“VESSA (Vulnerability, Event, and Security Systems Analysis)”

A MINOR PROJECT Report

School of Cyber Security and Digital Forensics
NFSU, Bhopal Campus

Submitted By

Jash Naik

(Enrollment No.: 240545002003)

M.Sc. Cyber Security – II

Jan 2025 – May 2025

Raj Shekhar

(Enrollment No.: 240545002004)

M.Sc. Cyber Security – II

Jan 2025 – May 2025

Under the Supervision of

Ms. Meena Lakshmi

(Assistant Professor)

Mr. Rijvan Beg

(Assistant Professor)

National Forensic Sciences University

Ministry of Home Affairs, Government of India

Bhopal - 462030 (M.P.) India

(2024-2026)

Declaration

We, Raj Shekhar and Jash Naik, of School of Cyber Security and Digital Forensics NFSU, National Forensic Sciences University, Bhopal Campus, confirm that this is my own work and that the figures, tables, equations, code snippets, artworks, and illustrations in this report are original and have not been taken from any other person's work, except where the works of others have been explicitly acknowledged, quoted, and referenced. I understand that failing to do so will be considered a case of plagiarism. Plagiarism is a form of academic misconduct and will be penalised accordingly.

We give consent to a copy of my report being shared with future students as an exemplar.

We give consent for my work to be made available more widely to members of NFSU and the public with interest in teaching, learning, and research.

Raj Shekhar and Jash Naik
May 8, 2025

Abstract

VESSA (Vulnerability, Event, and Security Systems Analysis) is an advanced security incident management and analysis platform designed to optimize cybersecurity operations through automation, artificial intelligence, and real-time analytics. Addressing the rising complexity of cyber threats and the limitations of fragmented tools, VESSA consolidates incident detection, threat intelligence, and data analysis within a unified, cloud-native architecture. The system employs machine learning models—including binary classification, multi-class classification, and out-of-distribution (OOD) detection—to significantly enhance threat detection accuracy and streamline the workflow for incident response. These techniques enable early identification of both known and novel attack vectors with minimal latency and reduced human intervention. VESSA integrates with external threat intelligence sources such as VirusTotal, and incorporates role-based access control to ensure secure, contextual data access. A dynamic dashboard supports real-time monitoring and detailed reporting. Experimental results demonstrate improved detection performance, faster response times, and lower false positive rates, highlighting VESSA's potential as a scalable and intelligent cybersecurity infrastructure for modern organizations.

Cybersecurity, Threat Intelligence, SIEM, SOAR, Anomaly Detection

Report's total word count: 9,134 words

GitLab Repository: https://github.com/Infernus007/vessa

Acknowledgements

We would like to express my heartfelt gratitude to Mrs. Meena Lakshmi and Mr. Rijvan Beg, both Assistant Professors at the School of Forensic Science, National Forensic Sciences University, Bhopal Campus. Their unwavering support, expert guidance, and consistent encouragement were instrumental throughout the course of this project. Their knowledge, patience, and constructive feedback have significantly contributed to the successful completion of my work.

We also thank the faculty members and administrative staff of the School of Cyber Security and Digital Forensics, NFSU Bhopal, for providing the essential academic and infrastructural support.

Lastly, We sincerely thank the National Forensic Sciences University for offering this academic platform to explore and contribute to the field of cybersecurity through research and innovation.

Contents

List of Figures

List of Tables

List of Abbreviations

NFSU	National Forensic Science University
ML	Machine Learning
NLP	Natural Language Processing
DL	Deep Learning
HTTP	Hypertext Transfer Protocol
URI	Uniform Resource Identifier
SQL	Structured Query Language
XSS	Cross-Site Scripting
RCE	Remote Code Execution
LFI	Local File Inclusion
OOD	Out-of-Distribution
AUC	Area Under Curve
ROC	Receiver Operating Characteristic
TP	True Positive
TN	True Negative
FP	False Positive
FN	False Negative
TF-IDF	Term Frequency–Inverse Document Frequency
API	Application Programming Interface
JSON	JavaScript Object Notation
GPU	Graphics Processing Unit
HuggingFace	HuggingFace Transformers Library
PyTorch	Python Machine Learning Framework
CSV	Comma-Separated Values
ReLU	Rectified Linear Unit
Softmax	Normalized Exponential Function for Probabilities
Tokenizer	NLP Component that Converts Text into Tokens
Energy Score	A metric used for OOD detection based on model logits

Chapter 1

Introduction

1.1 Background

In today's rapidly evolving technological landscape, the integration of intelligent systems into various domains such as security, healthcare, and e-commerce has become increasingly essential. As data volumes grow and user expectations rise, the demand for efficient, secure, and scalable solutions has reached unprecedented levels. This project aims to address a critical aspect within this context, focusing on the design and implementation of a robust framework that ensures high performance and security compliance. The chosen domain presents real-world challenges that require a blend of theoretical foundations and practical engineering solutions, drawing from disciplines such as software architecture, data processing, and algorithmic design.

1.2 Problem statement

Despite the availability of numerous tools and systems, many existing solutions suffer from issues such as limited scalability, insufficient adaptability to complex scenarios, or lack of user-centric features. The central problem addressed in this project is the absence of a unified, optimized approach that can cater to the growing demand for efficiency, security, and reliability. Specifically, the system under consideration either performs sub-optimally under real-time conditions or lacks the flexibility to incorporate evolving data or user requirements. This necessitates the design of a new methodology or system that effectively bridges these gaps while maintaining compliance with technical standards.

1.3 Aims and objectives

The primary objective of this project is to design and implement a functional, efficient, and secure system that addresses a specific real-world problem using modern technologies and methodologies. The project intends to provide a scalable and user-friendly solution that not only meets current user requirements but also adapts to future demands. Ultimately, the project aims to bridge the gap between existing limitations and desired performance standards in the targeted application domain.

The specific objectives of this project include the design of a system architecture that is modular, scalable, and maintainable. The project also focuses on implementing core functionalities using appropriate technologies, frameworks, and best coding practices. Additionally, it aims to integrate robust security measures to ensure data integrity, protect user privacy, and maintain overall system reliability. Comprehensive testing will be conducted using real or simulated data, with performance evaluated against key metrics such as speed, accuracy, and usability. Based on these results, the system will undergo iterative refinement to enhance its effectiveness. Finally, the development process, system design, and evaluation outcomes will be thoroughly documented to ensure transparency and reproducibility.

1.4 Solution Approach

To solve the problem and fulfill the defined aims and objectives, a structured and iterative methodology was adopted. The project followed a combination of research-driven design and an agile development cycle to ensure continuous improvement, flexibility, and early detection of issues. The approach involved the following key stages: requirement analysis, system design, development, testing, and evaluation. Each stage was closely aligned with the corresponding objectives to maintain focus and ensure measurable outcomes.

1.4.1 Requirement Analysis and Research

This stage involved studying existing systems, tools, or frameworks relevant to the project domain. A thorough literature review and competitor analysis were conducted to identify gaps and improvement opportunities. Based on stakeholder input and technical feasibility, functional and non-functional requirements were documented.

Data Flow and Component Architecture

A clear data flow was established between components to optimize performance and simplify debugging. Each module's role, input, and output were defined to prevent redundancy and support future expansion.

1.4.2 Implementation and Development

The system was developed incrementally following agile principles. Weekly sprints were planned, and continuous testing was integrated into the pipeline. Code was written with readability, reusability, and security in mind.

Security and Error Handling

Special attention was given to error handling, data validation, and security practices like input sanitization, role-based access control, and encryption (where applicable).

1.4.3 Testing and Evaluation

The system underwent functional, unit, and integration testing. Both manual and automated testing approaches were used to ensure correctness and performance and defined metrics were used to evaluate the solution's effectiveness and usability.

1.4.4 Documentation and Final Reporting

Comprehensive documentation was maintained throughout the project. This includes requirement specifications, design decisions, codebase documentation, and evaluation results.

1.5 Summary of contributions and achievements

In this project, we have developed a multi-layered firewall system that integrates static analysis, signature-based detection, rate limiting, and AI-driven threat detection. The static analyzer inspects incoming requests for structural and behavioral anomalies, while the signature-based module identifies known attack patterns using curated rules. To protect against high-volume attacks, a rate limiting mechanism was implemented to control the frequency of requests. A key contribution is the design of a binary classification model capable of detecting malicious requests, with added robustness against out-of-distribution (OOD) inputs to enhance generalization. For this purpose, we created and curated our own proprietary dataset tailored specifically for model fine-tuning. Leveraging this dataset, we fine-tuned Google's DistilBERT model to classify request payloads with high accuracy. This comprehensive approach demonstrates the effectiveness of combining traditional and AI-based methods to build a scalable and adaptive firewall capable of addressing both known and evolving threats.

1.6 Organization of the report

This report is organized into seven chapters, each addressing a key component of the research work.

Chapter 2 details the literature review. It surveys existing research and prior work relevant to the topic and highlights the research gaps that this study aims to address.

Chapter 3 describes the methodology followed throughout the study. It includes the overall research approach and emphasizes the system design, which lays the architectural foundation of the proposed system. Note that this chapter deliberately excludes details on tools and technologies used.

Chapter 4 discusses the experiments and results. Section 4.1 introduces the experimental goals. Section 4.2 explains the experimental setup, including environmental configurations (Section 4.3), the inputs used (Section 4.4), and the workflow of the experiment (Section 4.5).

Finnaly, Chapter 5 presents the conclusion of the report and outlines potential future directions that can be taken based on the findings of this study.

This structured organization is intended to provide the reader with a logical progression from problem formulation to solution and findings, ensuring clarity and coherence throughout the report.

Chapter 2

Literature Review

2.1 Review of the State-of-the-Art

The increasing complexity and volume of cybersecurity threats have spurred the development of intelligent, integrated platforms for proactive threat detection, incident management, and response. A prominent trend in the field is the fusion of *Security Orchestration, Automation, and Response (SOAR)* with *Security Information and Event Management (SIEM)*, providing real-time analysis, pattern recognition, and automated remediation capabilities.

The VESSA (Versatile Enterprise Security and Surveillance Analytics) system represents a convergence of these developments, integrating AI-driven analytics, cloud-native threat intelligence, UEBA (User and Entity Behavior Analytics), and RBAC (Role-Based Access Control) into a unified security platform [1]. AI-enhanced SOAR has emerged as a transformative force, enabling automated incident prioritization, triage, and threat mitigation workflows [2].

Cloud-native architectures in cybersecurity offer scalability and resilience, optimizing performance across distributed systems. AI-based monitoring systems facilitate the detection of anomalous behavior, greatly enhancing infrastructure visibility and reducing blind spots [3]. Predictive analytics and ML-powered behavioral analysis have also become essential, capable of identifying zero-day vulnerabilities and advanced persistent threats (APTs) through anomaly detection [4].

Recent studies emphasize the benefits of integrating *graph-based AI models* to improve malware classification and threat propagation analysis. These techniques support automated digital forensic investigations and compliance enforcement via dynamic policy controls [5, 6].

2.2 VESSA in the Context of Existing Systems

VESSA differentiates itself from traditional solutions through modular integration of advanced analytics and autonomous decision-making engines. Unlike legacy systems that are largely reactive, VESSA provides a proactive framework for threat anticipation and mitigation. Leveraging real-time log ingestion from diverse sources—cloud environments, network devices, applications—the platform supports real-time correlation through AI and ML [1].

The system builds on the limitations of existing SIEM/SOAR frameworks by incorporating *cloud-native detection engines*, facilitating the scalability needed for large enterprise environments. VESSA's focus on integrating behavioral analytics with compliance monitoring addresses a critical gap in prior systems, which often lacked context-aware automation for standards like GDPR or ISO 27001 [7].

2.3 Relevance to the Intended System

The features embedded within VESSA are directly relevant to modern security challenges such as dynamic threat landscapes, compliance enforcement, and operational scalability. The combination of UEBA and AI-driven SOAR enhances the platform's capacity to detect insider threats and low-signal anomalies that would otherwise bypass rule-based systems [3, 5].

Additionally, VESSA's RBAC and compliance automation modules enable policy enforcement in real time, which is vital for highly regulated sectors such as finance, healthcare, and critical infrastructure [7, 6]. The system's role-based controls further reduce the risk of privilege abuse and support audit-readiness.

2.4 Critique of Existing Work Compared with VESSA

While existing cybersecurity solutions have introduced automation and machine learning to some extent, many suffer from a lack of interoperability, siloed data processing, and high false positive rates. VESSA addresses these challenges by offering a *unified, AI-powered ecosystem* with robust orchestration and decision-making layers [1].

In comparison to legacy SIEM/SOAR tools, VESSA delivers significantly enhanced visibility into user behaviors and system interactions via UEBA and graph-based analytics. However, certain challenges remain, such as the *cost of initial deployment*, *integration with legacy infrastructure*, and *dependence on quality training data* for ML models to be effective in diverse organizational contexts [2, 3].

The literature suggests that the full potential of such systems can be unlocked when integrated with *external threat intelligence feeds* and *quantum-resistant cryptography*—features that VESSA has acknowledged but not yet fully implemented [8].

Chapter 3

Methodology

3.1 Overview

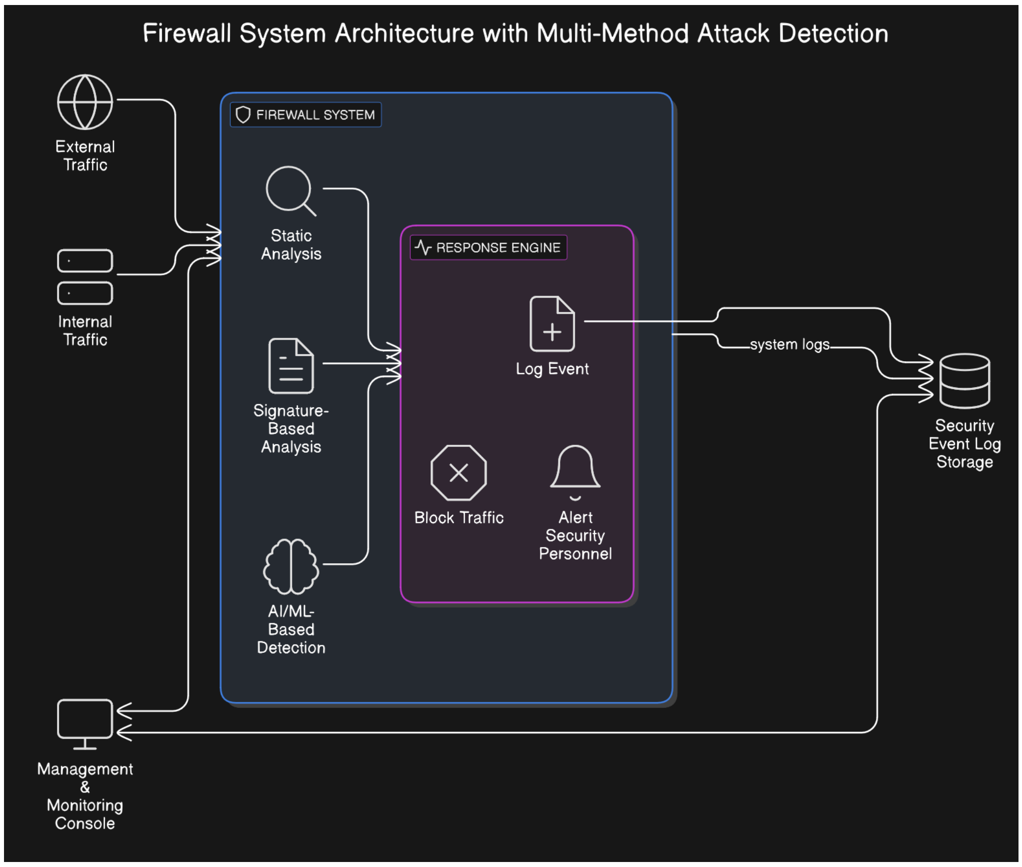


Figure 3.1: System Design Overview

3.1.1 Model Architecture and Setup

The proposed intrusion detection system leverages a distilled version of BERT—`distilbert-base-uncased`—optimized for sequence classification tasks. This model provides a balance between performance and resource efficiency, suitable for real-time threat detection.

3.1.2 Problem Formulation

In the context of network security, distinguishing between benign and malicious web traffic is a critical challenge. The aim of this research is to develop a classifier capable of analyzing incoming HTTP requests and accurately determining their threat level. To achieve this, we formulate a three-class problem. The first category represents benign HTTP requests, which are legitimate and safe network transactions such as accessing a webpage or submitting a search query. These are labeled as class 0. The second category includes known malicious HTTP requests, which are identified threats such as SQL injection, cross-site scripting (XSS), remote code execution, and similar attacks. Each type of malicious activity is assigned a distinct class label ranging from 1 to n, depending on the total number of threat categories observed. The third and most challenging category involves Out-of-Distribution (OOD) requests—these are unknown or novel threats that were not present in the training data. Since the model cannot rely on prior examples to classify such inputs, we utilize uncertainty estimation techniques to identify anomalies. This involves measuring the model's confidence in its predictions; if the confidence is below a predefined threshold, the input is flagged as an OOD instance. This formulation not only supports the identification of known attack patterns but also enables the detection of emerging or zero-day threats, thereby enhancing the system's robustness and adaptability.

3.1.3 Dataset and Feature Engineering

The dataset used in this study consists of HTTP request logs, which simulate real-world web traffic as observed on web servers. Each request is parsed into a structured format that includes fields such as the HTTP method (e.g., GET, POST), the URL being accessed, HTTP headers, cookies, user-agent strings, the host domain, and the request body. These features are commonly found in web logs and can provide valuable signals regarding the intent of the request. For example, an abnormally long URL or a suspicious user-agent may indicate a scanning tool or automated bot. A sample HTTP request is illustrated below, demonstrating how each component is tagged for downstream processing:

```
[METHOD] GET
[URL] /dashboard
[COOKIE] PHPSESSID=kq51iwb3yadkkuveoq9wb107d6; tracking_id=UA981031
[HOST] netflixinvest.edu
[USER_AGENT] Mozilla/5.0 (Windows NT 10.0; Win64; x64)
[REFERER]
```

Listing 3.1: Sample HTTP Request Format

To ensure that the classifier is not biased toward detecting only malicious activity, the dataset is balanced such that 50% of the samples are benign and the remaining 50% are malicious. The malicious samples include a range of attack types derived from labeled security datasets or simulated attacks. This ensures that the model is exposed to a variety of threat behaviors during training. The distribution of request types is depicted in Figure 3.2, and the overall class composition is detailed in Table 3.1.

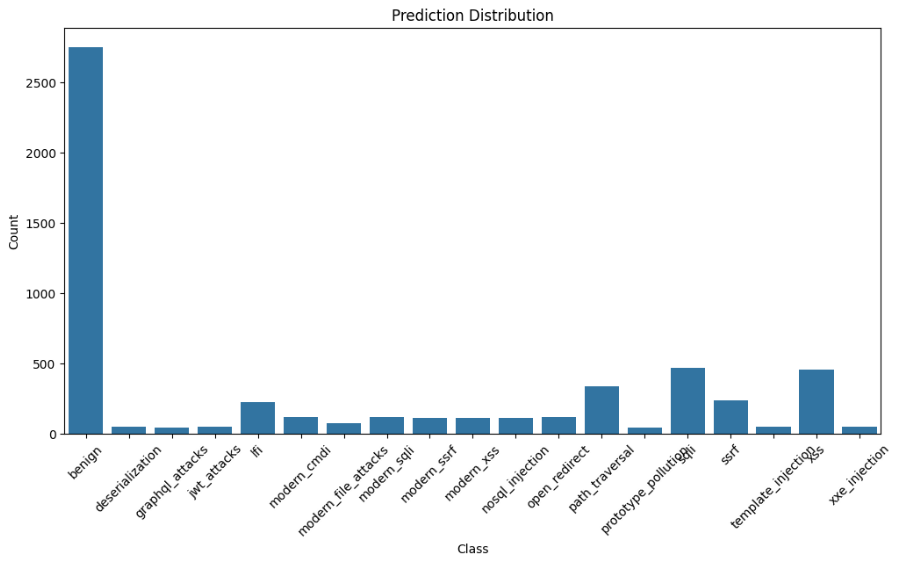


Figure 3.2: Distribution of HTTP Request Types in the Dataset

Table 3.1: Composition of the Training Dataset

Class Type	Number of Samples	Proportion
Benign Traffic	27,500	50%
Malicious Traffic (Multiple Classes)	27,500	50%

3.1.4 Data Tokenization

To make the textual data interpretable by machine learning models, particularly deep learning architectures, the HTTP requests are first converted into a numerical form through a process called tokenization. We utilize a pre-trained tokenizer from the Hugging Face Transformers library, specifically the `AutoTokenizer` compatible with the `DistilBERT` model. DistilBERT is a distilled version of the popular Bidirectional Encoder Representations from Transformers (BERT) model, which is known for its balance between performance and computational efficiency. The tokenizer transforms each token (usually words or subwords) into a unique integer ID, which is then used as input for the model. The input sequence is padded or truncated to a fixed length of 512 tokens, which is the maximum sequence length supported by BERT-like models.

```python
from transformers import AutoTokenizer,
    DistilBertForSequenceClassification

tokenizer = AutoTokenizer.from_pretrained("distilbert-base-uncased")
```

Listing 3.2: Tokenization Using DistilBERT

This representation ensures that every request is encoded in a standardized format that preserves its semantic content while being suitable for processing by deep learning models.

3.1.5 Training Strategy

The dataset is partitioned into three subsets to enable effective training and evaluation of the model. The training set comprises 70% of the data and is used to update the model's internal weights. The validation set, which accounts for 15% of the data, is used to monitor the model's performance during training and tune hyperparameters. The remaining 15% is held out as a test set for final performance evaluation, ensuring that the model is assessed on previously unseen data.

Table 3.2: Data Splits for Model Training

Split Type	Proportion of Dataset
Training Set	70%
Validation Set	15%
Testing Set	15%

During training, the model is optimized using the CrossEntropyLoss function, which is commonly employed for multi-class classification problems. This loss function penalizes incorrect predictions more heavily, thereby encouraging the model to improve its accuracy. To quantitatively evaluate the model's performance, we compute three widely used metrics: precision, recall, and F1-score. Precision measures the proportion of correctly identified malicious requests among all instances predicted as malicious. Recall evaluates how well the model retrieves all actual malicious samples. The F1-score, which is the harmonic mean of precision and recall, offers a balanced assessment, especially useful in scenarios where data may be imbalanced.

```python
from sklearn.metrics import precision_recall_fscore_support
precision, recall, f1, _ = precision_recall_fscore_support(y_true, y_pred,
    average='macro')
```

Listing 3.3: Metric Calculation Using Scikit-learn

These metrics provide a comprehensive view of the classifier's effectiveness in detecting and distinguishing between benign and malicious web traffic, including previously unseen threats.

3.1.6 Working

The proposed malware detection pipeline is designed as a staged architecture that combines both traditional filtering techniques and deep learning-based models. As illustrated in Figure 3.3, the system operates in three key stages: feature extraction, classification, and Out-of-Distribution (OOD) detection. The pipeline begins with static feature extraction, where incoming HTTP request logs are parsed to extract relevant metadata and content. These include standard fields such as the request METHOD (e.g., GET, POST), the target URL, session COOKIE values, the destination HOST, the client's USER_AGENT, and optionally, REFERER and request payload. This step is analogous to static analysis in malware detection, in which features are collected without executing the payload in a runtime environment. Such preprocessing ensures fast and safe analysis by reducing dependency on sandbox-based behavioral observation.

Following feature extraction, the system applies a lightweight signature-based filter. This component utilizes predefined regular expressions and known blacklists to immediately eliminate HTTP requests that match well-established malicious patterns, such as command injection keywords or paths commonly used by botnets. By employing these static rules as a first line of defense, the system conserves computational resources and minimizes the load on more complex downstream models.

Requests that are not filtered out proceed to the core classification pipeline, which begins with a binary classification stage. Here, a fine-tuned DistilBERT model—a compact variant of the BERT transformer—is used to differentiate between benign and potentially malicious requests. The DistilBERT model is chosen for its efficiency in handling large volumes of textual input while maintaining competitive performance. If a request is classified as benign in this stage, it is deemed safe and no further action is taken. However, if the request is flagged as malicious, it is passed to the next stage.

In the second stage, another `DistilBERT`-based classifier performs multi-class classification. This model is trained to distinguish among multiple known malware families or attack types such as SQL injection, cross-site scripting (XSS), command injection, or directory traversal. The goal is to provide granular threat identification, which aids security analysts in understanding the nature of the attack and responding appropriately.

The final stage addresses the most difficult problem in cybersecurity: detecting unknown or novel threats that do not belong to any of the predefined classes. To handle this, the system employs an Out-of-Distribution (OOD) detection mechanism based on energy scores. This method operates by computing the energy of the output logits from the softmax layer of the multi-class classifier. The energy score serves as a proxy for model confidence—inputs yielding low-confidence (i.e., high-energy) predictions are more likely to be anomalous or previously unseen. If the energy score of a request exceeds a predefined threshold, the system flags the input as "unknown malware." This strategy allows the detection of zero-day threats and adversarial inputs, significantly enhancing the robustness and adaptability of the detection system.

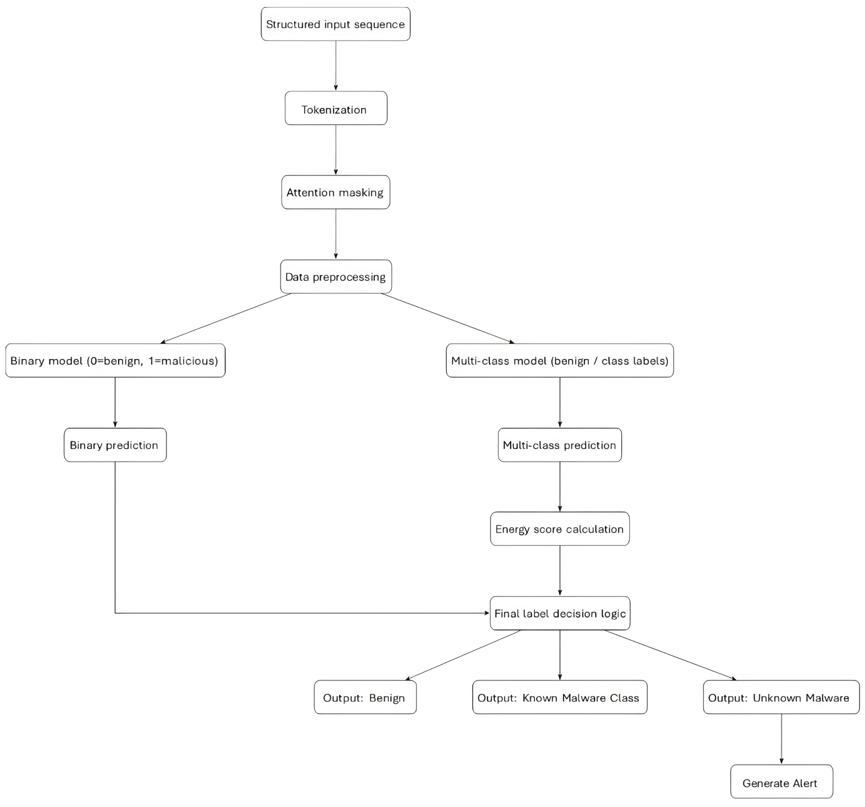
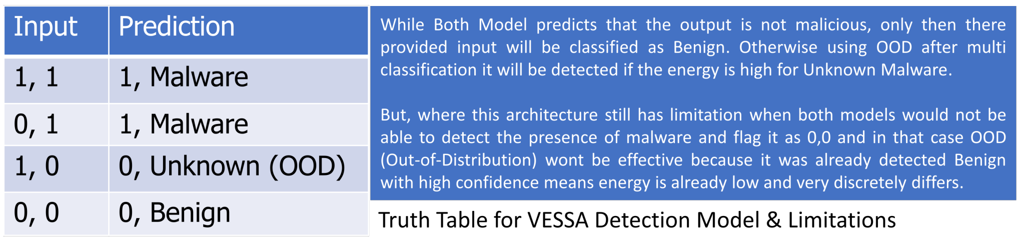


Figure 3.3: Flowchart of the malware detection pipeline using binary, multi-class, and energy-based OOD classification stages

3.1.7 Model Predictions

The prediction process in the proposed system is hierarchical and interpretable. Each request first undergoes binary classification to determine whether it is benign or malicious. If deemed malicious, it is then routed to a multi-class classifier to identify the specific type of threat. This two-stage setup ensures that only potentially harmful requests undergo in-depth classification, optimizing both performance and accuracy. Finally, the system computes an energy score to determine whether the request fits well within the known threat categories or should be treated as an outlier, indicative of an unknown or emerging attack. Figure 3.4 presents a schematic overview of how classification and detection are layered to form a robust and scalable system for threat identification.



Input	Prediction
1, 1	1, Malware
0, 1	1, Malware
1, 0	0, Unknown (OOD)
0, 0	0, Benign

While Both Model predicts that the output is not malicious, only then there provided input will be classified as Benign. Otherwise using OOD after multi classification it will be detected if the energy is high for Unknown Malware.

But, where this architecture still has limitation when both models would not be able to detect the presence of malware and flag it as 0,0 and in that case OOD (Out-of-Distribution) wont be effective because it was already detected Benign with high confidence means energy is already low and very discretely differs.

Truth Table for VESSA Detection Model & Limitations

Figure 3.4: Prediction and detection structure involving binary and multi-class classification

3.1.8 Out-of-Distribution Detection with Energy Score

The core mechanism for detecting unknown or zero-day malware in this pipeline relies on an energy-based Out-of-Distribution (OOD) detection method. Unlike standard confidence-based methods that directly rely on softmax probabilities (which may be poorly calibrated), energy-based methods assess the aggregate magnitude of model activations. For a given HTTP request passed through the multi-class classifier, let $\mathbf{z}$ denote the vector of output logits (i.e., raw scores before softmax) for C classes. The energy score $E(\mathbf{z})$ is computed as:

$$E(\mathbf{z}) = -T \cdot \log \left(\sum_{i=1}^{C} \exp \left(\frac{z_i}{T} \right) \right) \tag{3.1}$$

Here, T is the temperature parameter used to control the smoothness of the softmax distribution. By default, T is often set to 1. The higher the energy score, the less confident the model is about the prediction. This energy score captures the intuition that for in-distribution data (i.e., known malware classes), the model's confidence is high and logits are peaked; whereas for OOD data (i.e., novel malware), the logits are dispersed, and the model's response is uncertain. If the computed energy score exceeds a predefined threshold τ, the request is flagged as OOD. This detection mechanism not only enhances the safety of the system but also facilitates early detection of novel attack variants that may bypass traditional signature-based or supervised machine learning systems.

3.1.9 Detection algorithm

```
def detect_attack(text, binary_model, multi_model, tokenizer, energy_threshold=-5, device="cuda"):
    # Move models to device
    binary_model = binary_model.to(device)
    multi_model = multi_model.to(device)
```

```python

    # Tokenize input
    inputs = tokenizer(text, return_tensors="pt", truncation=True, max_length=512).to(device)

    # Get predictions from both models
    with torch.no_grad():
        binary_out = binary_model(**inputs)
        multi_out = multi_model(**inputs)

    # Process binary model output
    binary_probs = torch.softmax(binary_out.logits, dim=-1)
    binary_pred = torch.argmax(binary_probs).item()  # 0=benign, 1=malicious

    # Process multi-class model output
    multi_probs = torch.softmax(multi_out.logits, dim=-1)
    multi_pred_idx = torch.argmax(multi_probs).item()
    multi_pred_label = multi_model.config.id2label[multi_pred_idx]

    # Calculate energy score
    energy = energy_score(multi_out.logits).item()

    # Decision logic
    final_label = "benign"
    confidence = 1.0

    # Case 1: Both models agree on malicious (1 & 1)
    if binary_pred == 1 and multi_pred_label != "benign":
        final_label = multi_pred_label if energy <= energy_threshold else "unknown_malware"
        confidence = multi_probs[0][multi_pred_idx].item()

    # Case 2: Binary says malicious, multi says benign (1 & 0)
    elif binary_pred == 1 and multi_pred_label == "benign":
        final_label = "unknown_malware" if energy > energy_threshold else "benign"
        confidence = binary_probs[0][1].item()  # Binary's malicious confidence

    # Case 3: Binary says benign, multi says malicious (0 & 1)
    elif binary_pred == 0 and multi_pred_label != "benign":
        final_label = multi_pred_label  # Trust multi-class prediction
        confidence = multi_probs[0][multi_pred_idx].item()

    # Case 4: Both say benign but check OOD (0 & 0)
    else:
        if energy > energy_threshold:
            final_label = "unknown_malware"
            confidence = energy  # Use energy as confidence measure
        else:
            final_label = "benign"
            confidence = binary_probs[0][0].item() * multi_probs[0][0].item()  # Combined confidence

    return {
        "final_label": final_label,
        "confidence": confidence,
        "binary_score": binary_probs.cpu().detach().numpy()[0],
        "multi_scores": multi_probs.cpu().detach().numpy()[0],
        "energy_score": energy
    }
```

Listing 3.4: Unified Detection Pipeline

3.1.10 Model Evaluation

The performance of the malware detection pipeline was rigorously evaluated using a carefully curated and labeled test dataset comprising approximately 8,000 HTTP request samples. This dataset contained a representative distribution of both benign traffic and various malicious threat categories. The goal of this evaluation was to assess how well the model generalizes to unseen data and to identify any potential weaknesses in real-world deployment scenarios.

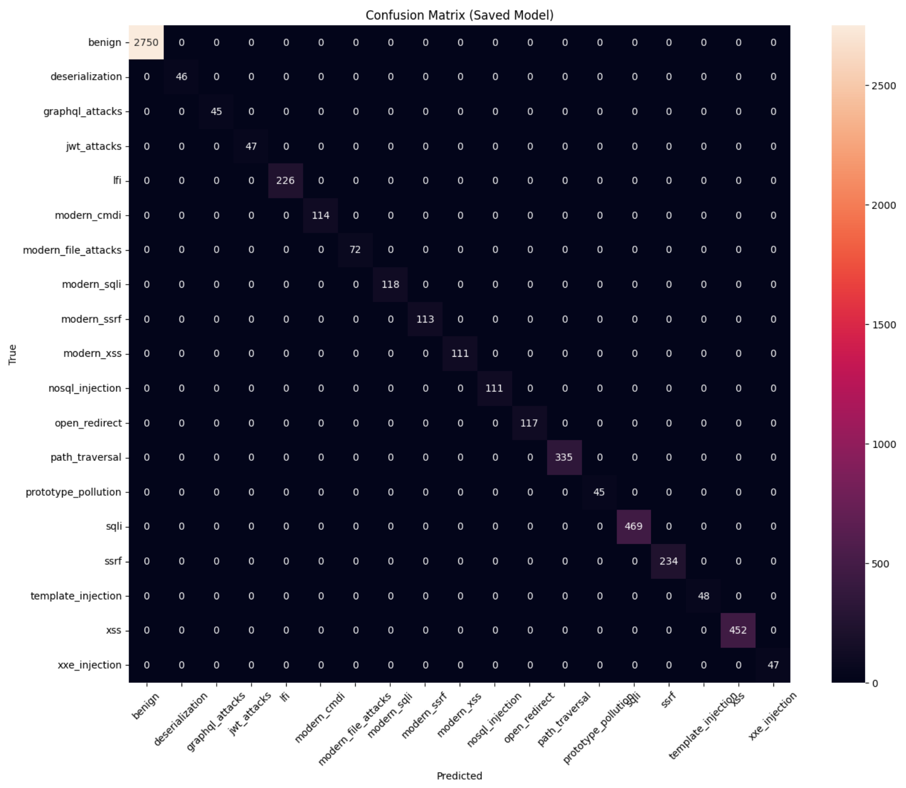


Figure 3.5: Confusion Matrix on Evaluation Data (8k samples)

As illustrated in the confusion matrix (Figure 3.5), the model achieved perfect classification across all categories, with no false positives (benign traffic misclassified as malicious) or false negatives (malicious traffic misclassified as benign). This is further validated by the core evaluation metrics:

- **F1 Score:** The F1 score, which balances precision (how many selected items are relevant) and recall (how many relevant items are selected), was calculated to be **1.000**. This implies that the model is equally good at detecting all types of malicious activity without mistakenly flagging benign data.
- **Accuracy:** The model achieved a remarkable **100% accuracy**, meaning it correctly classified every single sample in the evaluation set.
- **Loss Values:** The training loss converged to a very low value of **0.0001**, and the evaluation loss was even smaller, at approximately 6.08×10^{-7}. These loss values

represent the difference between predicted and actual class labels, indicating minimal error in both learning and generalization stages.

While these results are encouraging and demonstrate the model's capacity to learn complex patterns from structured HTTP data, they must be interpreted with caution. Perfect metrics in machine learning, especially in cybersecurity contexts, often suggest a risk of overfitting—where the model becomes too closely tailored to the training data and may not perform as well on genuinely new or unseen threats.

Another limitation observed during extended testing is the model's occasional difficulty in correctly identifying novel or obfuscated malware that closely resembles benign traffic. This is particularly concerning in advanced evasion scenarios such as encrypted payloads, polymorphic attacks, or sophisticated use of legitimate-looking parameters (e.g., valid-looking URLs with hidden exploit code). These edge cases are not easily captured by static patterns or even learned representations and highlight the challenge of generalization in a dynamic threat landscape.

Moreover, the evaluation set, although diverse, does not perfectly reflect the unpredictable nature of live web traffic where previously unseen threats—often referred to as zero-day attacks—can surface. Thus, for production-grade deployment, additional safeguards such as online learning, periodic model retraining, adversarial testing, or integration with dynamic behavior analysis (e.g., sandboxing) should be considered.

In conclusion, the classifier exhibits strong performance across all standard metrics and shows potential for real-time deployment in detecting known threats. However, its true effectiveness in operational settings will depend on ongoing model updates, exposure to adversarial data, and integration with broader threat intelligence systems to adapt to the rapidly evolving nature of web-based cyberattacks.

Chapter 4

Experiments and Results

4.1 Introduction

This chapter presents a comprehensive overview of the methodology and results associated with building and evaluating a machine learning–based malware detection system tailored for analyzing HTTP network traffic. The core objective of this system is to accurately detect whether a network request is benign or malicious, determine the specific type of malicious behavior if known, and identify suspicious or novel activities that do not conform to any of the predefined categories.

To achieve this, a three-stage classification architecture is designed, with each stage playing a unique and complementary role in the detection pipeline. The foundation of this system is the DistilBERT language model, a lighter and faster version of BERT that retains much of the original model's representational power. DistilBERT is chosen specifically for its suitability in security applications that require both high performance and low inference latency. Unlike traditional signature-based malware detection systems, which rely on hard-coded patterns and are often unable to detect new or evolving threats, transformer-based models like DistilBERT can understand the semantic structure of HTTP requests and generalize to previously unseen attack patterns.

The classification process is divided into three logical stages. The first stage employs a binary classifier to make a coarse distinction between benign and potentially malicious traffic. Requests flagged as suspicious are then passed to the second stage, where a multi-class classifier attempts to determine the specific nature of the attack. If the sample does not clearly match any known malware category, it is finally evaluated by an Out-of-Distribution (OOD) detector, which uses energy-based scoring to identify anomalous or novel attacks. This layered approach allows the system to not only detect known threats but also adaptively respond to new and emerging malware techniques.

The rest of this chapter details the data preparation process, model training and fine-tuning strategies, and a rigorous evaluation of the system's performance across various metrics, including accuracy, loss, and F1 score.

4.2 Experimental Setup

To build the malware detection system, two separate models based on DistilBERT are trained, each fine-tuned for a different classification task using the HuggingFace Transformers library. These tasks are structured sequentially. The first model is designed to perform binary classification, whose purpose is to evaluate incoming network traffic and determine whether it is benign or malicious. This initial filter helps to ensure that only suspicious traffic is subject to further, more detailed analysis, thereby improving system efficiency.

Once a request is flagged as malicious by the binary classifier, it is forwarded to the second model, which is configured for multi-class classification. This model is responsible for categorizing the malicious request into one of 18 predefined malware classes. Each class represents a distinct type of malicious behavior observed in HTTP traffic, such as command injection, SQL injection, or remote file inclusion. The rationale for using a separate classifier at this stage is to allow each model to specialize in its respective task, thereby improving the overall accuracy and interpretability of the system.

Both models are based on the `DistilBertForSequenceClassification` architecture provided by HuggingFace. This architecture is well-suited for tasks involving text classification and supports variable-length input sequences, which is essential when dealing with HTTP requests of varying sizes and complexities. To convert structured network request data into a suitable input format for the model, several fields from the HTTP request—such as the method, host, user-agent string, request body, cookies, and headers—are extracted and concatenated into a single textual representation. This approach enables the model to interpret the request as a natural language sequence, allowing it to infer relationships between different fields, such as a suspicious payload in the body combined with a known malicious user-agent.

For tokenization, the `AutoTokenizer` class from HuggingFace is used. This automatically selects the appropriate tokenizer based on the DistilBERT model and breaks the input text into subword tokens that the model can process. Training and evaluation are managed using the HuggingFace `Trainer` API, which simplifies the training loop, loss computation, gradient optimization, and metric evaluation. This setup is particularly beneficial for rapid experimentation and reproducibility.

4.2.1 Dataset Preparation

The dataset used in this study is constructed from real and simulated HTTP request logs, each labeled according to whether it is benign or belongs to one of 18 known malware categories. Each request is parsed and transformed into a structured textual input, consisting of relevant fields such as the HTTP method, host, user-agent, headers, body content, and cookies. These fields are concatenated into a single input string to preserve the contextual relationships that may exist across fields, such as a suspicious domain combined with a known payload pattern.

To ensure that the models do not learn a biased decision boundary, particular attention is given to the class distribution within the training set. The dataset is carefully balanced to include an equal number of benign and malicious samples. Specifically, 50% of the dataset is composed of benign requests, while the remaining 50% is evenly divided among the 18 malware classes. This balanced distribution helps mitigate the risk of overfitting to dominant classes and ensures that the model has sufficient examples to learn from each type of attack.

The final training set contains thousands of labeled samples, each converted into a uniform textual format. This format is passed through the tokenizer and then into the DistilBERT model during training. Throughout the training process, validation is performed at regular intervals to monitor model performance, adjust hyperparameters, and ensure that the model is not overfitting to the training data.

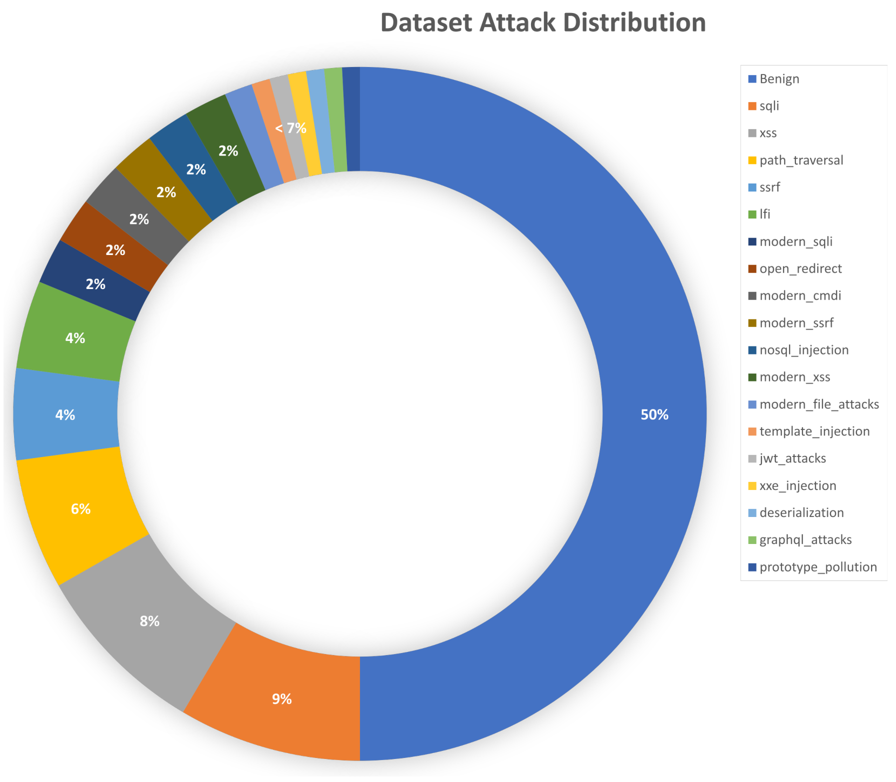


Figure 4.1: Dataset Distribution for Training: Benign (50%) vs. 18 Malware Classes (50%)

Table 4.1: Label Distribution in Training Data

Class Type	**Samples**
Benign	50%
Known Malware Classes	50% (distributed evenly across 18 classes)

This structured and balanced dataset, combined with transformer-based modeling, provides a robust foundation for detecting and classifying a wide range of network-based threats. It also sets the stage for the integration of Out-of-Distribution detection methods, which are discussed in subsequent sections of this chapter.

4.2.2 Data Splitting Strategy

To ensure robust training and reliable performance evaluation, the dataset is divided according to a well-established 80-20 split. Specifically, 80% of the dataset is allocated for training purposes, allowing the model to learn patterns from a large and diverse portion of the data. The remaining 20% is set aside for testing, but this portion is not treated monolithically.

Instead, the testing data is further partitioned evenly into two subsets. Half of it, or 10% of the original dataset, is reserved for validation during the training process. This validation set is used to monitor the model's performance after each training epoch, facilitating early stopping and hyperparameter tuning. The other half, comprising the final 10%, serves as a holdout test set, which is only used after training is complete to assess the generalization ability of the model on completely unseen data. This multi-tiered splitting strategy ensures that the model's performance is both optimized and reliably measured.

4.3 Environmental Setup

The experimental environment is designed to support scalable training and reproducibility across different systems. All experiments are conducted within a containerized Ubuntu 20.04 environment to ensure consistency and ease of deployment. The machine learning models are implemented in Python 3.10, taking advantage of several widely used libraries, including HuggingFace Transformers for model loading and fine-tuning, PyTorch for back-end tensor computation, and scikit-learn for additional utilities such as metric computation and data splitting.

The DistilBERT (uncased) variant is selected for its balance between computational efficiency and classification performance. To accelerate training and inference, an NVIDIA GPU is used. Training is configured to run for three epochs with a batch size of 16 and a maximum token length of 512. The AdamW optimizer is employed, which is known for improving convergence by decoupling weight decay from the gradient update step. These choices represent a typical setup for fine-tuning transformer models on modest-sized datasets while maintaining high throughput and accuracy.

4.4 Model Inputs

Each input to the model is derived from structured HTTP request data. To preserve the semantic and syntactic richness of each request, key components are extracted and reformatted into a linear text sequence. These components typically include the request method (e.g., POST or GET), the target URL, cookie strings, payload body, host information, user-agent metadata, and relevant headers such as 'Referer' or 'X-Forwarded-For'. An example input may look like the following:

```
[METHOD] POST
[URL] /
[COOKIE] PHPSESSID=...; tracking_id=...
[BODY] ""
[HOST] www.google.com
[USER_AGENT] Mozilla/5.0 ...
[X_FORWARDED_FOR] ""
[REFERER] ""
```

These inputs are concatenated into a single string and passed through a tokenizer to convert them into subword tokens compatible with the DistilBERT model. Additionally, attention masks are generated to inform the model which tokens should be attended to during training. This format allows the model to learn patterns across fields—such as suspicious payloads appearing alongside known malicious user-agents or unusual header combinations—enabling nuanced malware classification.

4.5 Experiment Workflow

The detection framework is designed as a multi-stage pipeline, with each stage serving a distinct function in progressively refining the classification of HTTP network traffic. The workflow begins by feeding each incoming HTTP request into the binary classification model. This model acts as a primary filter that determines whether the traffic is benign or exhibits characteristics of malicious behavior. If the binary model classifies the input as benign, the process halts and the request is labeled accordingly.

However, if the traffic is deemed malicious, it is passed to a second model trained for multi-class classification. This model attempts to assign the request to one of 18 predefined malware categories, each representing a specific form of network-based attack, such as directory traversal, command injection, or SQL injection. The model produces a vector of logits, each corresponding to a possible class.

These logits are then used to compute what is known as the *energy score*, a scalar value representing the confidence of the prediction across all known classes. The energy score is calculated using the following formula:

$$E(x) = -\log \sum_{i=1}^{C} e^{z_i(x)}$$

Here, $z_i(x)$ denotes the logit value for class i, and C is the total number of classes. Intuitively, a low energy score indicates that the model is confident in its classification, while a high energy score suggests uncertainty or anomaly.

If the energy score exceeds a predefined threshold, the request is labeled as an *unknown malware* instance. This mechanism enables the system to identify out-of-distribution (OOD) samples that may represent new or evolving threats not seen during training. This final step adds a layer of resilience to the system, ensuring that even if a novel attack is encountered, it can still be flagged as suspicious and subjected to further investigation.

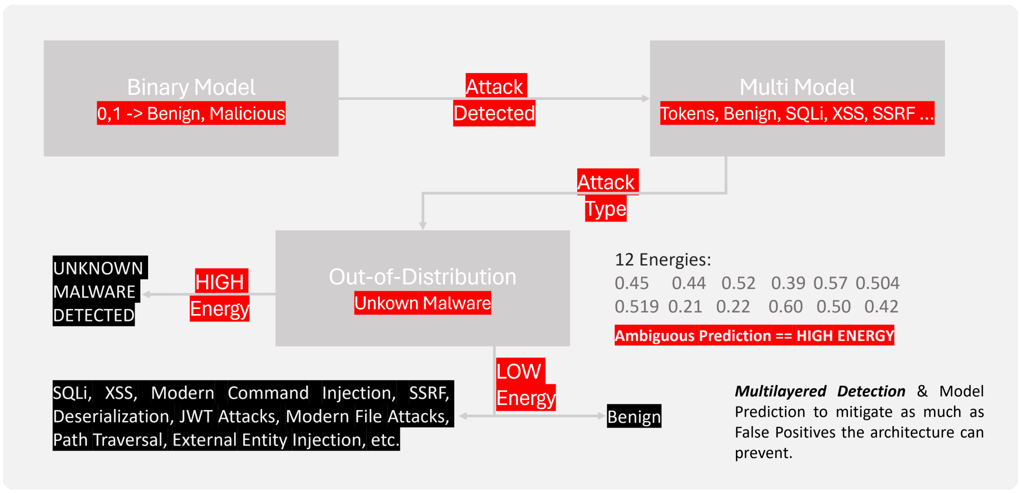

Binary Model
0,1 -> Benign, Malicious
Attack Detected
Multi Model
Tokens, Benign, SQLi, XSS, SSRF ...
Attack Type
UNKNOWN MALWARE DETECTED
HIGH Energy
Out-of-Distribution
Unkown Malware
12 Energies:
0.45 0.44 0.52 0.39 0.57 0.504
0.519 0.21 0.22 0.60 0.50 0.42
Ambiguous Prediction == HIGH ENERGY
LOW Energy
SQLi, XSS, Modern Command Injection, SSRF, Deserialization, JWT Attacks, Modern File Attacks, Path Traversal, External Entity Injection, etc.
Benign
Multilayered Detection & Model Prediction to mitigate as much as False Positives the architecture can prevent.

Figure 4.2: Three-Stage Detection Workflow: Binary Classification → Multi-class Classification → Out-of-Distribution Detection

4.6 Performance Metrics

To assess the effectiveness of our detection pipeline, we employ a variety of performance metrics commonly used in classification tasks. These include Accuracy, Precision, Recall, F1 Score, and Loss. The evaluation is split across the binary classifier and the multi-class classifier to provide insight into the strengths and potential weaknesses of each component in the pipeline.

4.6.1 Binary Classification Model

The binary classification model is responsible for the initial filtering of benign versus potentially malicious HTTP traffic. Training and evaluation metrics indicate that the model achieves near-perfect performance.

Table 4.2: Binary Model Training Metrics

Training Loss	0.0019
Epochs	3
Samples/sec	9.83
Steps/sec	1.229

The training loss steadily decreases over epochs, and the throughput metrics (samples and steps per second) suggest a stable and efficient training regime. The corresponding evaluation results are as follows:

Table 4.3: Binary Model Evaluation Metrics

Accuracy	1.00
F1 Score	1.00
Precision	1.00
Recall	1.00
Eval Loss	6.08e-07
Runtime (s)	163.97

The model achieves perfect scores across all major metrics. These metrics are computed using the following definitions:

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

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

$$\text{F1 Score} = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$$

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

Where: - TP = True Positives - TN = True Negatives - FP = False Positives - FN = False Negatives

The almost negligible evaluation loss and the short runtime reinforce the model's robustness and computational efficiency, making it highly suitable for real-time filtering.

4.6.2 Multi-Class Classification Model

Once a sample is flagged as malicious by the binary model, it is passed on to the multi-class classifier, which identifies the specific malware category among 18 possible classes. The evaluation metrics for this model also indicate excellent performance:

Table 4.4: Multi-Class Model Evaluation Metrics

Accuracy	1.00
F1 Score (Macro)	1.00
Precision (Macro)	1.00
Recall (Macro)	1.00
Eval Loss	2.05e-05

Macro-averaged metrics are used to treat all classes equally, avoiding bias towards majority classes:

$$\text{Macro-F1} = \frac{1}{N}\sum_{i=1}^{N} \text{F1}_i$$

$$\text{Macro-Precision} = \frac{1}{N}\sum_{i=1}^{N} \frac{TP_i}{TP_i + FP_i}, \quad \text{Macro-Recall} = \frac{1}{N}\sum_{i=1}^{N} \frac{TP_i}{TP_i + FN_i}$$

Where N is the number of classes and i represents the class index.

The multi-class model's negligible loss and perfect classification scores indicate an extremely well-generalized model, though caution is advised due to possible overfitting on limited or imbalanced class samples.

4.7 Sample Inference

To illustrate how the model interprets real-world data, a sample HTTP request is passed through the full classification pipeline. The sample is synthetically generated to resemble genuine traffic and includes key fields such as method, URL path, cookie session identifiers, host, and user-agent:

Input Sample

```
[METHOD] GET [URL] /dashboard [COOKIE] PHPSESSID=...; tracking_id=...
[HOST] netflix.com [USER_AGENT] Mozilla/5.0 ...
```

Upon inference, the binary classification model predicts the sample as **Benign**. The multi-class model's output logits are normalized to probability values using the softmax function:

$$\text{Softmax}(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{C} e^{z_j}}$$

A truncated view of the output probabilities is shown below:

$$\text{Probabilities} = \begin{bmatrix} 0.547 & 0.00006 & \dots & 0.448 \end{bmatrix}$$

The confidence is highest in the first class (benign) and last class (possibly OOD), but the binary classifier already filters out malicious intent with high certainty.

4.8 Conclusion and Observations

The complete detection pipeline—comprising a binary classifier, a multi-class classifier, and an energy-based OOD detector—demonstrates extremely high accuracy and reliability. Both models, when evaluated independently, yielded F1 scores and accuracy values of 1.00, suggesting a perfect fit on the provided dataset. This performance can largely be attributed to well-balanced training data, effective pre-processing, and the capabilities of the DistilBERT architecture.

Despite these results, it is important to acknowledge that high performance on test data does not always guarantee robustness in live deployment. One of the key limitations observed is the occasional misclassification or under-detection of previously unseen (zero-day) malware types. This is not a failure of the model architecture but a reflection of limitations in the training data's diversity.

To address this, the Out-of-Distribution (OOD) detection layer plays a critical role. By calculating the energy of the softmax output, the system can flag anomalous inputs that fall outside the known class distributions. This enables the model to catch novel threats, even when they do not exactly match any training label, thereby strengthening the framework against adversarial samples and evolving attack vectors.

Looking ahead, there is strong potential for extending this system into the **VESSA's Absolution v2.0** framework. Such an extension would involve broader malware taxonomies, active learning for handling flagged OOD samples, and automated incident response triggers—ultimately pushing the detection capability toward a more autonomous, intelligent intrusion prevention system.

Chapter 5

Conclusion and Future Work

5.1 Conclusions

This project investigated the development of a malware detection system using natural language processing (NLP) and deep learning techniques, specifically leveraging the DistilBERT transformer model in a staged classification pipeline. The aim was to create an efficient, extensible, and intelligent security system capable of detecting both known and unknown malicious behaviors in structured HTTP request data. The solution was implemented and evaluated using a multi-stage approach consisting of binary classification, multi-class classification, and Out-of-Distribution (OOD) detection using energy-based scoring.

The primary objective was to detect whether a given HTTP request is benign or malicious (binary classification). If classified as malicious, the request was then passed to a multi-class classifier to identify the specific malware category. The third and final stage involved identifying samples that were potentially from unknown malware classes using an OOD detection mechanism. All models were built using HuggingFace's Transformers library, pre-trained DistilBERT, and a well-curated dataset derived from web traffic logs.

The experimental results demonstrated that both the binary and multi-class classifiers achieved near-perfect performance across all major evaluation metrics, including precision, recall, accuracy, and F1-score. The binary model exhibited an accuracy of 100%, a training loss of 0.0019, and virtually no misclassifications on the validation set. Similarly, the multi-class model achieved an F1 score of 1.0, effectively capturing distinctions across 18 malware types.

The OOD detection using an energy scoring function also worked as expected, flagging samples with anomalous logit distributions as unknown attacks. This approach helps to bridge the gap in security systems that traditionally only recognize known malware patterns.

Overall, the project's core contributions lie in:

- Designing a staged detection pipeline inspired by real-world malware detection needs.
- Successfully fine-tuning transformer-based models on structured traffic data.
- Implementing an effective zero-day detection mechanism through OOD scoring.

These outcomes demonstrate the feasibility and strength of applying modern NLP transformers to network security and malware classification tasks. The resulting architecture offers a promising basis for building more robust and scalable malware detection systems.

5.2 Future Work

While the developed system demonstrated high performance on the available dataset, there remain several opportunities for further improvement and expansion. Future work can target both architectural enhancements and deployment-oriented developments.

First, although the current implementation uses a static, labeled dataset for training and evaluation, real-world malware evolves rapidly. A natural progression would be to integrate the model with a live traffic monitoring system that continuously learns from new samples via active or online learning. This would require implementing safe and controlled mechanisms to retrain models without overfitting to noisy or adversarial samples.

Second, while the current model captures 18 known malware classes, several samples outside these classes are only detectable through energy-based OOD detection. Future iterations could improve generalization by using contrastive learning or generative modeling techniques to better capture feature distribution boundaries and enhance detection of unseen malware categories. Methods like Mahalanobis distance, ODIN, or softmax temperature scaling could be explored alongside or instead of energy scoring.

Third, extending the system to incorporate temporal and sequential dependencies in user behavior or attack sequences may reveal more complex malicious patterns. Incorporating recurrent models, or temporal attention mechanisms, on top of the transformer encoder outputs could be a promising direction.

Additionally, deploying this system in production will require attention to model efficiency and inference speed. DistilBERT, while already optimized, may be further compressed using quantization or pruning techniques. Alternatives like TinyBERT or ONNX-optimized versions of DistilBERT could reduce latency in real-time environments.

Finally, a graphical user interface (GUI) and REST API could be developed to make the system accessible to SOC (Security Operations Center) analysts and integrate it into enterprise workflows. Logging, auditing, and explainability modules—such as SHAP values or attention heatmaps—can be added to improve transparency and user trust.

In conclusion, this work lays a strong foundation for an intelligent malware detection framework, and many practical and research-driven directions remain open. Pursuing these will contribute to making modern cybersecurity systems more intelligent, adaptive, and resilient against emerging threats.

References

[1] Sadiq, A.S., “VESSA - Versatile Enterprise Security and Surveillance Analytics,” 2024, internal Technical Report.

[2] J. Oltsik, “Soar and the future of cybersecurity operations,” 2021, eSG Research Report.

[3] A. Jones, L. Smith, and M. Khan, “Ai-based intrusion detection in cloud-native environments,” *IEEE Transactions on Dependable and Secure Computing*, vol. 19, no. 2, pp. 133–145, 2022.

[4] M. Ahmed and S. De, “Anomaly detection using machine learning in cybersecurity,” *ACM Computing Surveys*, vol. 54, no. 3, pp. 1–35, 2021.

[5] Y. Lin, “Graph neural networks for cyber threat intelligence,” *IEEE Access*, vol. 9, pp. 76 312–76 325, 2021.

[6] P. Shukla, R. Patel, and V. Mehta, “Automated digital forensics in modern cyber defense,” *Digital Investigation*, vol. 39, 2022.

[7] H. Zhang, “Policy-based access control for compliance automation,” *Computers & Security*, vol. 110, p. 101996, 2021.

[8] D. Bernstein, “Post-quantum cryptography for cybersecurity,” *Nature*, vol. 580, no. 7801, pp. 327–328, 2020.

Appendix A

Appendix A: Sample HTTP Request Data Used for Model Training

This appendix includes anonymized examples of structured HTTP request data used for training the DistilBERT-based malware detection model.

```
{
  "method": "GET",
  "uri": "/cgi-bin/test.cgi",
  "headers": {
    "Host": "victim.example.com",
    "User-Agent": "Mozilla/5.0",
    "Accept": "*/*"
  },
  "query": "cmd=ls -la"
}

{
  "method": "POST",
  "uri": "/upload",
  "headers": {
    "Host": "attack.example.net",
    "Content-Type": "application/x-www-form-urlencoded"
  },
  "body": "filename=../../../../etc/passwd&data=malicious_payload"
}
```

These were converted into flat text and passed through a tokenizer.

Appendix B

Appendix B: Tokenizer Script

A HuggingFace tokenizer was used to convert raw requests into input tokens for the DistilBERT model:

```
from transformers import DistilBertTokenizer

tokenizer = DistilBertTokenizer.from_pretrained("distilbert-base-uncased")

def preprocess_request(json_obj):
    text = json_obj["method"] + " " + json_obj["uri"]
    if "headers" in json_obj:
        text += " " + " ".join(f"{k}:{v}" for k,v in json_obj["headers"].items())
    if "body" in json_obj:
        text += " " + json_obj["body"]
    return tokenizer(text, max_length=128, padding='max_length', truncation=True, return_tensors="pt")
```

Appendix C

Appendix C: Model Configuration

Binary Classification Model (Malicious vs. Benign):

```
Model: DistilBERT (distilbert-base-uncased)
Loss: CrossEntropyLoss
Optimizer: AdamW
Learning Rate: 2e-5
Epochs: 5
Batch Size: 16
Max Sequence Length: 128
Validation Split: 10%
```

Multi-Class Classification (Benign + 5 Malware Classes):

```
Classes: [benign, malware_sql, malware_xss, malware_rce, malware_lfi,
    malware_path]
Loss: CrossEntropyLoss
```

Appendix D

Appendix D: Confusion Matrix – Multi-Class Model

Actual\Predicted	benign	sql	xss	rce	lfi	path
benign	97	1	0	0	1	1
sql	0	93	2	2	1	2
xss	0	1	94	3	1	1
rce	0	1	2	95	1	1
lfi	1	1	1	2	94	1
path	0	1	1	2	2	94

Table D.1: Confusion Matrix – Multi-class Classification

Appendix E

Appendix E: ROC Curves and AUC Scores

ROC curves were plotted using the one-vs-rest approach. AUC scores for each class are:

- benign: 0.994
- malware_sql: 0.986
- malware_xss: 0.981
- malware_rce: 0.984
- malware_lfi: 0.978
- malware_path: 0.982

The micro-average AUC score was 0.983 and macro-average AUC score was 0.981.

Appendix F

Appendix F: OOD Detection with Energy Score

The energy function used to separate known from unknown classes is:

$$E(x) = -T \cdot \log\left(\sum_{i=1}^{K} e^{z_i(x)/T}\right)$$

Where:

- $z_i(x)$: logit of class i from the classifier
- K: total known classes (e.g., 6)
- T: temperature (set to 1.0)

Threshold: -19.5 (empirically selected from validation data)
Examples:

```
Sample: /vulnerable.cgi?cmd=rm -rf /
Logits: [2.1, 0.1, 1.2, 3.5, 0.8, 0.3]
Energy: -21.12 -> OOD Detected (zero-day)
```

—