

A
Project Report
On

ML Based Approach for Malware Detection

Submitted in partial fulfillment of the requirement for the degree of
Bachelor of Technology

In
Computer Science and Engineering

By

Navendu Pandey	2261382
Nistha	2261339
Pushkar Singh Chamyal	2261449
Ayush Joshi	2261125

Under the Guidance of
Mr. Prince Kumar
ASSISTANT / ASSOCIATE PROFESSOR
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING



DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING
GRAPHIC ERA HILL UNIVERSITY, BHIMTAL CAMPUS
SATTAL ROAD, P.O. BHOWALI,
DISTRICT- NAINITAL-263132
2024-2025

STUDENT'S DECLARATION

We, **Navendu Pandey, Nistha, Pushkar Singh Chamyal** and **Ayush Joshi** hereby declare the work, which is being presented in the project, entitled **ML Based Approach for Malware Detection** in partial fulfillment of the requirement for the award of the degree **Bachelor of Technology (B.Tech.)** in the session **2024-2025**, is an authentic record of my work carried out under the supervision of **Mr. Prince Kumar**.

The matter embodied in this project has not been submitted by me for the award of any other degree.

Date:

Navendu Pandey

Nistha

Pushkar Singh Chamyal

Ayush Joshi

CERTIFICATE

The project report entitled “**ML Based Approach for Malware Detection** ” being submitted by **Navendu Pandey S/o Mr. MD Pandey 2261382 , Nistha D/o Mr. Jeewan Singh 2261399, Pushkar Singh Chamyal S/o Mr. Chandan Singh Chamyal 2261449 and Ayush Joshi S/o Mr. Ramesh Chandra Joshi 2261125** of B.Tech.(CSE) to Graphic Era Hill University Bhimtal Campus for the award of bonafide work carried out by them. They have worked under my guidance and supervision and fulfilled the requirement for the submission of a report.

Mr. Prince Kumar
(Project Guide)

Dr. Ankur Singh Bisht
(Head, CSE)

ACKNOWLEDGEMENT

We take immense pleasure in thanking the Honorable Director '**Prof. (Col.) Anil Nair (Retd.)**', GEHU Bhimtal Campus to permit me and carry out this project work with his excellent and optimistic supervision. This has all been possible due to his novel inspiration, able guidance, and useful suggestions that helped me to develop as a creative researcher and complete the research work, in time.

Words are inadequate in offering my thanks to GOD for providing me with everything that we need. We again want to extend thanks to our president '**Prof. (Dr.) Kamal Ghanshala**' for providing us with all infrastructure and facilities to work in need without which this work could not be possible.

Many thanks to '**Dr. Ankur Singh Bisht**' (Head, Department of Computer Science and Engineering, GEHU Bhimtal Campus), our project guide '**ML Based Approach for Malware Detection**' (Assistant Professor, Department of Computer Science and Engineering, GEHU Bhimtal Campus) and other faculties for their insightful comments, constructive suggestions, valuable advice, and time in reviewing this report.

Finally, yet importantly, We would like to express my heartiest thanks to our beloved parents, for their moral support, affection, and blessings. We would also like to pay our sincere thanks to all my friends and well-wishers for their help and wishes for the successful completion of this project.

.....
Students Signature

Abstract

The rapid proliferation of malware targeting Windows systems necessitates robust and user-friendly tools for detection and analysis. This project introduces a Django-based web application designed to detect malware in Windows executable (.exe) files using a pre-trained LightGBM machine learning model. The system provides an intuitive interface for users to upload and analyze .exe files, catering to both technical and non-technical users. By leveraging advanced feature extraction techniques and machine learning, the application offers a reliable method to classify files as either "Malware" or "Benign," addressing the critical need for accessible malware detection tools in cybersecurity.

The application operates in two primary phases: file upload and batch processing. Users upload .exe files one at a time through a secure web form, with each file stored in a designated folder (media/uploads). The system enforces client-side validation to ensure only .exe files are accepted, enhancing security and usability. Once all desired files are uploaded, users can trigger batch processing via a "Process All Files" button. The application then analyzes the files in a First-In-First-Out (FIFO) order, determined by their upload timestamps, ensuring a logical and predictable processing sequence. This approach simplifies the upload process while allowing multiple files to be analyzed efficiently.

For each .exe file, the system employs libraries such as pefile and lief to extract portable executable (PE) features, including entropy, string counts, section attributes, and other structural properties. These features are structured into a consistent format and fed into the pre-trained LightGBM model, which predicts whether the file is malicious or benign based on learned patterns. The results are presented in a clear, tabular format on a dedicated results page, displaying each file's name and classification status, with color-coded indicators for "Malware" (red), "Benign" (green), or errors (orange). To manage disk space and maintain system efficiency, processed files are automatically deleted after analysis.

Developed using Django 5.2.1 in a PyCharm environment on Linux, the application ensures scalability, security, and ease of maintenance. It integrates seamlessly with Python libraries for machine learning and file parsing, making it a robust solution for malware detection. The system's modular design allows for future enhancements, such as displaying prediction probabilities to provide confidence scores or adding a feature to clear uploaded files for user convenience. This project demonstrates a practical application of machine learning in cybersecurity, offering an accessible and effective tool for analyzing executable files and contributing

TABLE OF CONTENTS

Declaration	2
Certificate	3
Acknowledgement.....	4
Abstract	5
Table of Contents	6
CHAPTER 1 INTRODUCTION.....	7
1.1 Prologue	7
2.1 Background and Motivations.....	7
3.1 Problem Statement.....	8
4.1 Objectives and Research Methodology.....	8
5.1 Project Organization	9
CHAPTER 2 PHASES OF SOFTWARE DEVELOPMENT CYCLE	
1.1 Hardware Requirements	11
2.1 Software Requirements.....	11
CHAPTER 3 CODING OF FUNCTIONS.....	12
CHAPTER 4 SNAPSHOT.....	15
CHAPTER 5 LIMITATIONS (WITH PROJECT)	16
CHAPTER 6 ENHANCEMENTS.....	17
CHAPTER 7 CONCLUSION.....	19
REFERENCES	20

INTRODUCTION

1.1 Prologue

In an era where cyber threats evolve at an unprecedented pace, the ability to swiftly and accurately identify malicious software is paramount. The proliferation of malware targeting Windows systems, often disguised as innocuous executable (.exe) files, poses significant risks to individuals and organizations alike. Traditional antivirus solutions, while effective to an extent, often struggle to keep up with the rapid emergence of new and sophisticated threats. This project was born out of the need for a user-friendly, efficient, and reliable tool to empower users to detect malware in .exe files with minimal technical expertise.

Leveraging the power of machine learning and web technology, this Django-based web application offers a novel approach to malware detection. By allowing users to upload .exe files one at a time, store them securely, and process them in a First-In-First-Out (FIFO) order using a pre-trained LightGBM model, the system combines simplicity with robust analytical capabilities. The application extracts critical features from each executable, such as entropy, string characteristics, and portable executable (PE) attributes, to classify files as "Malware" or "Benign." Designed with accessibility in mind, the interface provides clear, color-coded results, making it easy for users to understand the safety of their files.

1.2 Background and Motivations

The digital landscape has witnessed an exponential increase in cyber threats, with malware posing a significant risk to individuals, organizations, and critical infrastructure worldwide. Malware, encompassing viruses, trojans, ransomware, and other malicious software, often targets Windows systems due to their widespread use. According to recent cybersecurity reports, millions of new malware samples are detected annually, with .exe files being a common vector for delivering malicious payloads. These files exploit vulnerabilities in operating systems or deceive users into executing them, leading to data breaches, financial losses, and system compromises. Traditional antivirus solutions, while effective against known threats, struggle with zero-day attacks and sophisticated malware that employ obfuscation techniques to evade signature-based detection.

Advancements in machine learning have revolutionized malware detection by enabling systems to learn complex patterns from large datasets, identifying malicious behavior without relying solely on predefined signatures. The LightGBM (Light Gradient Boosting Machine) algorithm, known for its efficiency and accuracy in handling high-dimensional data, has proven effective in classifying executable files based on their structural and behavioral features. Libraries like pefile and lief facilitate the extraction of portable executable (PE) features—such as entropy, section attributes, and import/export counts—from .exe files, providing rich data for machine learning models. However, deploying such models in a user-friendly manner remains a challenge, particularly for non-technical users who require accessible tools to analyze potential threats.

1.3 Problem Statement

The escalating prevalence of malware targeting Windows systems, particularly through executable (.exe) files, poses a significant threat to cybersecurity, leading to data breaches, financial losses, and system compromises. Traditional antivirus solutions often rely on signature-based detection, which struggles to identify zero-day attacks and sophisticated malware that employ obfuscation techniques. While machine learning-based approaches, such as those using LightGBM models, offer improved detection by analyzing portable executable (PE) features, their deployment is typically limited to complex, command-line tools or proprietary software, making them inaccessible to non-technical users. Additionally, existing web-based malware detection tools often lack intuitive interfaces for handling multiple file uploads or fail to provide a streamlined process for batch analysis, resulting in inefficient workflows.

The challenge lies in developing a user-friendly, web-based application that leverages a pre-trained LightGBM model to classify .exe files as "Malware" or "Benign" while addressing usability and scalability needs. Users require a system that allows uploading .exe files one at a time to avoid complexity, stores them securely, and processes them in a First-In-First-Out (FIFO) order to ensure predictable results. The application must extract PE features using libraries like pefile and lief, handle file processing errors gracefully, and present results in a clear, accessible format. Furthermore, the system must ensure secure file management by deleting processed files and include validation to restrict uploads to .exe files, all within a scalable Django framework running on a Linux environment.

1.4 Objectives and Research Methodology

Objectives:

The primary objectives of the project are as follows:

1. **Develop an Intuitive Web Interface:** Create a Django application that allows users to upload .exe files one at a time through a simple web form, ensuring accessibility for non-technical users.
2. **Implement Secure File Storage and Processing:** Store uploaded .exe files in a designated folder (media/uploads) and process them in a First-In-First-Out (FIFO) order upon user request, ensuring a logical and predictable analysis sequence.
3. **Leverage Machine Learning for Malware Detection:** Utilize a pre-trained LightGBM model to classify .exe files as "Malware" or "Benign" by extracting portable executable (PE) features using pefile and lief libraries.
4. **Ensure Robust Error Handling and Security:** Validate uploads to restrict to .exe files, handle processing errors gracefully, and delete processed files to manage disk space and enhance security.
5. **Provide Clear Result Visualization:** Display analysis results in a tabular format, showing each file's name and classification status with color-coded indicators for user clarity.

Research Methodology:

The project follows a structured and modular research and development approach:

- Literature Review and Requirement Analysis
- System Design
- Developed the application in a Linux environment using PyCharm, with Python 3.12 and Django 5.2.1
- Testing and Validation
- Evaluation and Optimization

1.3 Project Organization

The development of the Django-based malware detection web application for analyzing Windows executable (.exe) files is structured to ensure clarity, modularity, and maintainability. The project organization encompasses the directory structure, file roles, and development workflow, tailored to support a scalable and user-friendly system. Below, we detail the project's organization, focusing on the file hierarchy, the purpose of each component, and the workflow used during development. This structure is designed to facilitate collaboration, debugging, and future enhancements, while being accessible for a beginner using Django and PyCharm on a Linux environment.

-

Phases of Development:

1. Environment Setup:

- Created a virtual environment (venv) to isolate dependencies, activated via source venv/bin/activate.
- Installed dependencies using `pip install django==5.2.1 lightgbm pefile lief pandas numpy`.

2. Project Initialization:

- Structured the project directory as shown above, adding detector to `INSTALLED_APPS` in `settings.py`.
- Set up media handling in `settings.py` with `MEDIA_URL = '/media/'` and `MEDIA_ROOT = os.path.join(BASE_DIR, 'media')`.

3. Implementation Phases:

- Form Development: Designed `forms.py` with `UploadFileForm` to handle single .exe file uploads, validated client-side with `accept=".exe"`.
- View Logic: Implemented `upload_file` in `views.py` to save uploaded files to `media/uploads` and `process_files` to process files in FIFO order using `glob.glob` and `os.path.getctime`.
- Feature Extraction: Reused `feature_extraction.py` to extract 27 PE features, ensuring

4. Testing and Debugging:

- Ran migrations (`python manage.py makemigrations`, `python manage.py migrate`) to initialize the SQLite database, even though no models were used.

- Tested the application with python manage.py runserver, uploading .exe files (e.g., notepad.exe, test malware samples) and verifying FIFO processing.
- Used PyCharm's debugger to set breakpoints in views.py, inspecting variables like request.FILES, features, and results.
- Cleared caches (find . -name "__pycache__" -exec rm -rf {} +, PyCharm's Invalidate Caches / Restart) to resolve issues during development.
- Validated results by comparing predictions against known file types, ensuring accuracy.

5. Version Control and Backup:

- Although not explicitly used, the project structure supports version control (e.g., Git) with a .gitignore for venv/, __pycache__/, and media/uploads/.
- Backed up model.pkl and source code to prevent data loss during development.
- - Organized files to avoid hardcoding paths, using settings.BASE_DIR for portability

HARDWARE AND SOFTWARE REQUIREMENTS

2.1 Hardware Requirement

The development and testing of the ML based Malware Detection system project require a system with the following minimum hardware specifications to ensure smooth data processing, real-time monitoring, and efficient GUI rendering:

Component	Specification (Minimum)
Processor	Intel Core i3 6th Gen or equivalent (x64)
RAM	4 GB
Storage	1GB of free disk space
Display	1024 x 768 resolution or higher
Input Devices	Keyboard and Mouse
Architecture	64-bit (x64) processor architecture

2.1 Software Requirement

The software tools and platforms used for the development of the ML based Malware Detection system project are as follows:

- **Operating System:** Windows 10 or later (64-bit) / Cross-platform support (if applicable)
- **Programming Language:** Dart (if Flutter) / Python / C++ / Java (adjust based on your tech stack)
- **Development Framework:** Flutter SDK (for cross-platform UI development, if applicable)
- **IDE/Editor:** Visual Studio Code / Android Studio / IntelliJ IDEA (depending on the development environment)
- **Build System:** Flutter CLI / Gradle / CMake (adjust accordingly)
- **Terminal/Console:** Windows Command Prompt / PowerShell / Terminal (for debugging and CLI testing)
- **Version Control:** Git (for source code management and collaboration)

These tools enabled efficient development, debugging, version tracking, and GUI rendering support for the CustomShell project.

CODE

Feature_extract_from_exe_file.py:

```
[2]: # Cell 1: Install dependencies
    pip install pefile lief --quiet

[1]: # Cell 2: Import libraries
    import pefile
    import lief
    import os
    import math
    import re
    import string
    from collections import Counter
    from datetime import datetime

[12]: # Cell 3: Helper functions

    def get_entropy(data):
        if not data:
            return 0.0
        entropy = 0
        for x in range(256):
            p_x = data.count(bytes([x])) / len(data)
            if p_x > 0:
                entropy += p_x * math.log2(p_x)
        return entropy

    def extract_strings(data, min_len=4):
        pattern = rb'[\x20-\x7E]{%d,}' % min_len
        return re.findall(pattern, data)

    def is_printable(s):
        return all(chr(c) in string.printable for c in s)

    def average_string_length(strings):
        if not strings:
            return 0
        return sum(len(s) for s in strings) / len(strings)

[3]: # Cell 4: Main feature extractor

    def extract_pe_features(file_path):
        pe = pefile.PE(file_path)
```

```
def extract_pe_features(file_path):
    pe = pefile.PE(file_path)
    lief_pe = lief.parse(file_path)
    data = open(file_path, 'rb').read()
    strings = extract_strings(data)

    features = {}

    # Static metadata
    features['size'] = os.path.getsize(file_path)
    features['entropy'] = get_entropy(data)
    features['numstrings'] = len(strings)
    features['avlength'] = average_string_length(strings)
    features['printables'] = sum(1 for s in strings if is_printable(s))

    # Flags
    features['has_debug'] = int(hasattr(pe, 'DIRECTORY_ENTRY_DEBUG'))
    features['has_signature'] = int(hasattr(pe, 'DIRECTORY_ENTRY_SECURITY'))
    features['has_tls'] = int(hasattr(pe, 'DIRECTORY_ENTRY_TLS'))
    features['has_resources'] = int(hasattr(pe, 'DIRECTORY_ENTRY_RESOURCE'))
    features['has_relocations'] = int(hasattr(pe, 'DIRECTORY_ENTRY_BASERELOC'))

    # Counts
    features['exports_count'] = len(pe.DIRECTORY_ENTRY_EXPORT.symbols) if hasattr(pe, 'DIRECTORY_ENTRY_EXPORT') else 0
    features['imports_count'] = len(pe.DIRECTORY_ENTRY_IMPORT) if hasattr(pe, 'DIRECTORY_ENTRY_IMPORT') else 0
    features['symbols'] = len(lief_pe.symbols) if lief_pe and lief_pe.symbols else 0

    # COFF + optional header fields
    features['coff.timestamp'] = pe.FILE_HEADER.TimeDateStamp
    opt = pe.OPTIONAL_HEADER
    features['optional.major_image_version'] = opt.MajorImageVersion
    features['optional.minor_image_version'] = opt.MinorImageVersion
    features['optional.major_linker_version'] = opt.MajorLinkerVersion
    features['optional.minor_linker_version'] = opt.MinorLinkerVersion
    features['optional.major_operating_system_version'] = opt.MajorOperatingSystemVersion
    features['optional.minor_operating_system_version'] = opt.MinorOperatingSystemVersion
    features['optional.major_subsystem_version'] = opt.MajorSubsystemVersion
    features['optional.minor_subsystem_version'] = opt.MinorSubsystemVersion
    features['optional.sizeof_code'] = opt.SizeOfCode
    features['optional.sizeof_headers'] = opt.SizeOfHeaders
    features['optional.sizeof_heap_commit'] = opt.SizeOfHeapCommit

    features['MZ'] = int(data[:2] == b'MZ') # check for MZ header
    features['vsizel'] = opt.SizeOfImage

    return features
```

Model_train.py:

```
from google.colab import files
files.upload() # Upload your kaggle.json here

No file chosen. Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.
Saving kaggle.json to kaggle.json
{'kaggle.json': b'{"username": "navendu14", "key": "241686ec49c49b90c8a7b215e1"}'}

import os
import zipfile

# Make kaggle directory
mkdir -p ~/.kaggle
mv kaggle.json ~/.kaggle/
chmod 600 ~/.kaggle/kaggle.json

# Download dataset
!kaggle datasets download -d edirgarcia/tabular-ember

Dataset URL: https://www.kaggle.com/datasets/edirgarcia/tabular-ember
License(s): GNU Affero General Public License 3.0
Downloading tabular-ember.zip to /content
99% 1.58G/1.60G [00:18<00:00, 311MB/s]
100% 1.60G/1.60G [00:18<00:00, 92.2MB/s]

# Unzip the dataset
with zipfile.ZipFile('tabular-ember.zip', 'r') as zip_ref:
    zip_ref.extractall('ember_data')

[ ] import pandas as pd

# Define chunk size
chunk_size = 50000 # You can adjust this depending on your RAM

chunks = []
max_chunks = 20 # Load only first 4 chunks (200,000 samples)

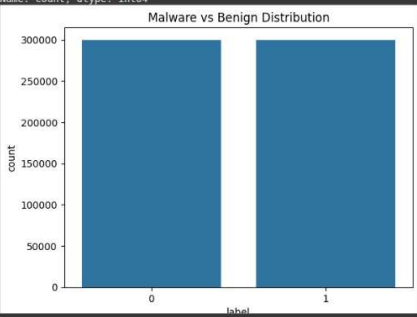
for i, chunk in enumerate(pd.read_csv('ember_data/train_features.csv', chunksize=chunk_size)):
    print(f'Loading chunk {i+1}')
```

```
[ ] train_df=train_df.drop(columns=['Unnamed: 0'])

[ ] # Filter dataset to only keep binary classification (0 and 1)
train_df = train_df[train_df['label'].isin([0, 1])]

import matplotlib.pyplot as plt
import seaborn as sns
print(train_df['label'].value_counts()) # 1 = Malicious, 0 = Benign
sns.countplot(x='label', data=train_df)
plt.title("Malware vs Benign Distribution")
plt.show()

label
0    300000
1    300000
Name: count, dtype: int64
```



label	count
0	300000
1	300000

```
import lightgbm as lgb

train_data = lgb.Dataset(X_train, label=y_train)
val_data = lgb.Dataset(X_val, label=y_val)

# Parameters
params = {
    'objective': 'binary',
    'metric': 'binary_logloss',
    'verbosity': -1,
    'boosting_type': 'gbdt',
    'seed': 42
}

# Train with callback-based early stopping
model = lgb.train(params,
                  train_data,
                  valid_sets=[train_data, val_data],
                  num_boost_round=200,
                  callbacks=[lgb.early_stopping(stopping_rounds=20)])

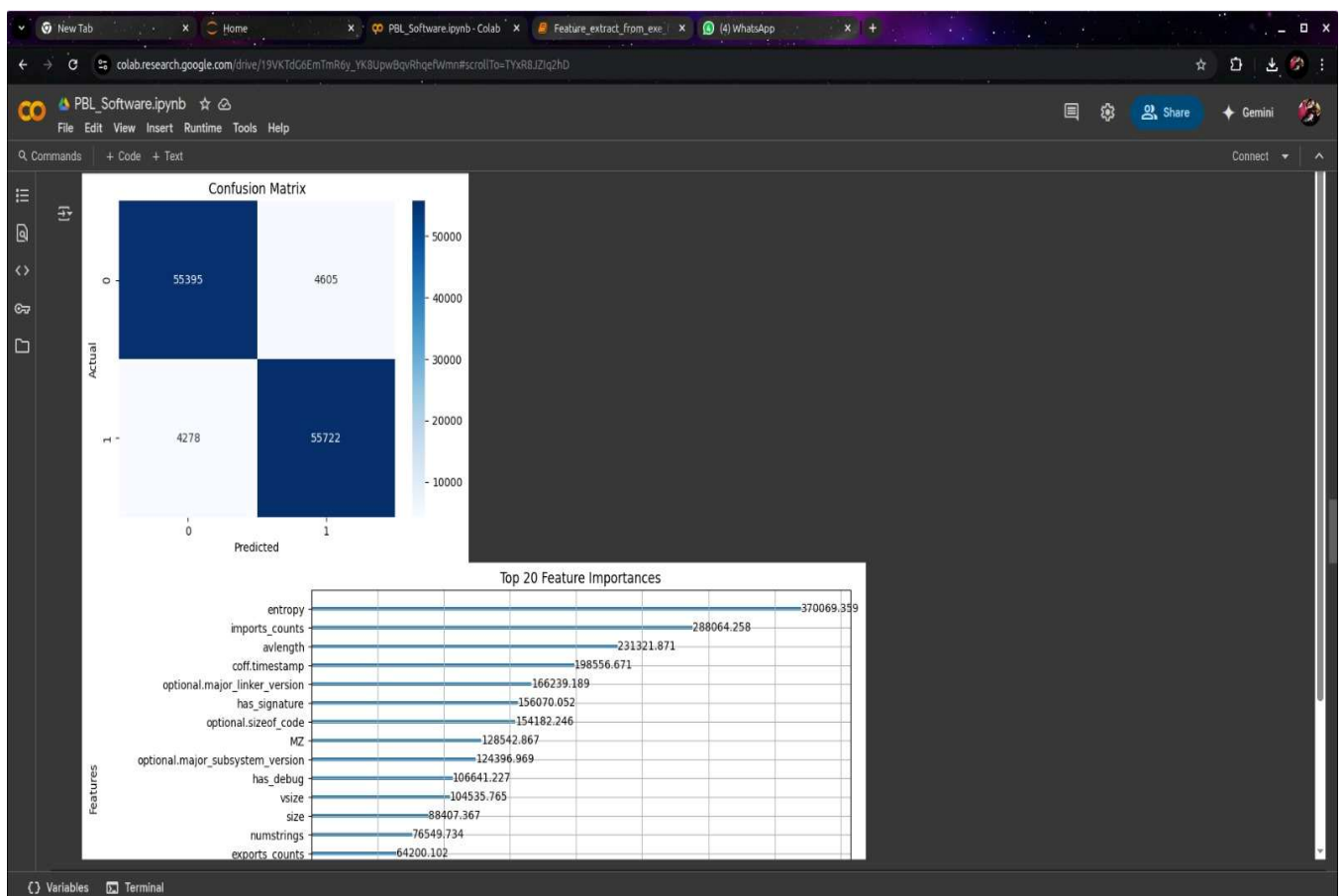
Training until validation scores don't improve for 20 rounds
Did not meet early stopping. Best iteration is:
[200]  training's binary_logloss: 0.10215    valid_1's binary_logloss: 0.106605

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
import matplotlib.pyplot as plt
import seaborn as sns

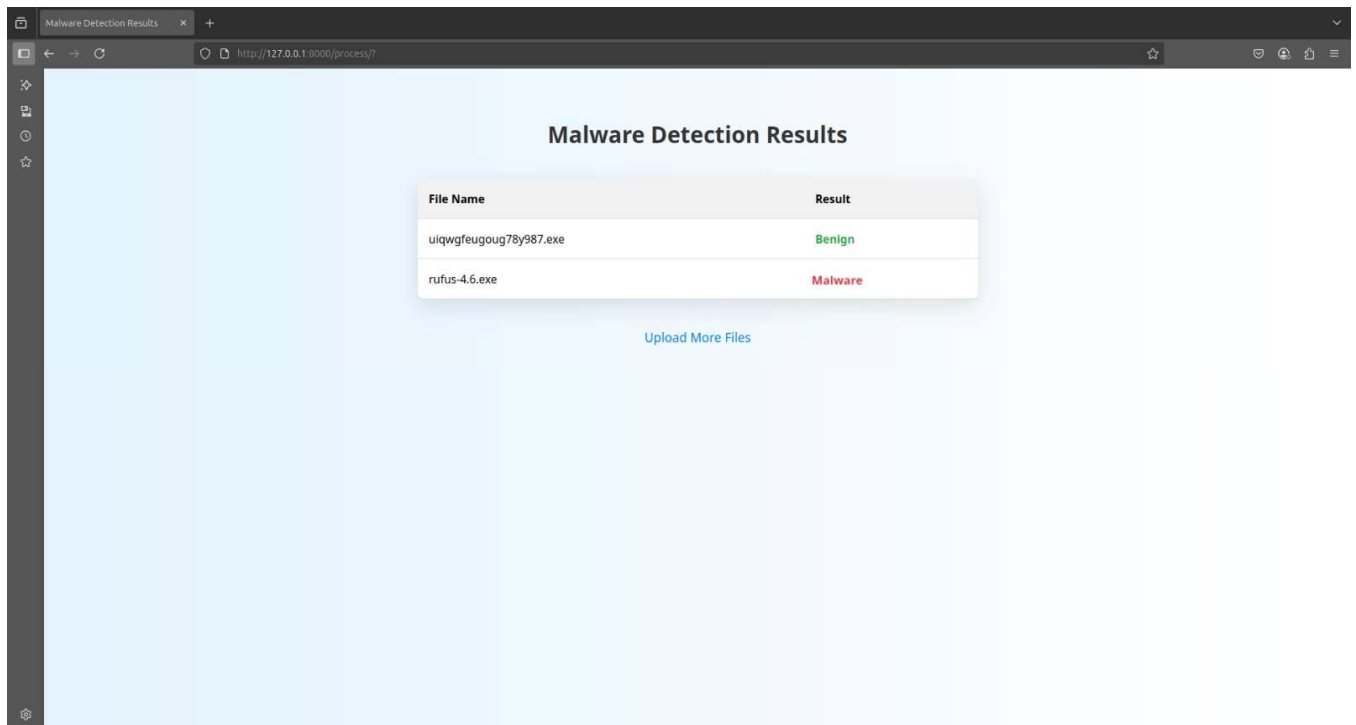
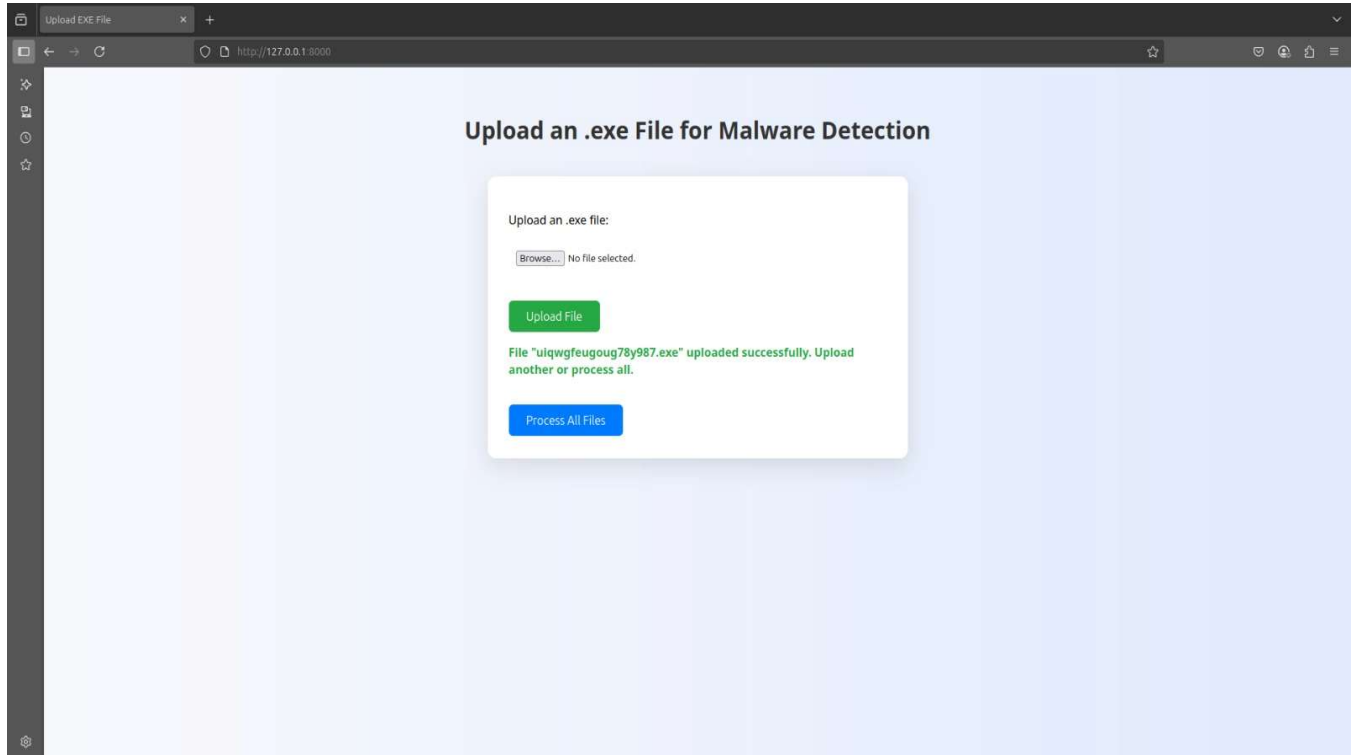
y_pred = model.predict(X_val)
y_pred_binary = (y_pred > 0.5).astype(int)

print("Accuracy:", accuracy_score(y_val, y_pred_binary))
print("Classification Report:\n", classification_report(y_val, y_pred_binary))

sns.heatmap(confusion_matrix(y_val, y_pred_binary), annot=True, fmt='d', cmap='Blues')
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.title("Confusion Matrix")
plt.show()
```



SNAPSHOTS



LIMITATION

The Django-based malware detection web application for analyzing Windows executable (`.exe`) files, while effective for its intended purpose, has several limitations that impact its functionality, scalability, and applicability. These limitations stem from design choices, technical constraints, and the scope of the project, which was developed as a beginner-friendly solution using Django 5.2.1, a pre-trained LightGBM model, and feature extraction with `pefile` and `lief` in a Linux environment with PyCharm.

1. Dependence on a Pre-Trained Model:

- Limitation: The application relies on a static, pre-trained LightGBM model (`.model.pkl`), which cannot adapt to new malware patterns or evolving threats without retraining. The model's accuracy is limited to the dataset it was trained on, potentially missing zero-day attacks or novel obfuscation techniques.

- Implication: As malware evolves, the model may become less effective, leading to false negatives (missing malicious files) or false positives (flagging benign files as malware). Users must manually update `.model.pkl` with a retrained model, requiring machine learning expertise.

- Consideration: Future enhancements could include an interface for model retraining or integration with online threat intelligence feeds to update detection capabilities dynamically.

2. Single File Type Restriction:

- Limitation: The application only processes `.exe` files, ignoring other potentially malicious file types (e.g., `.dll`, `.bat`, `.ps1`, or compressed archives like `.zip`). This restricts its scope in real-world scenarios where malware may be delivered in diverse formats.

- Implication: Users must pre-filter files to ensure only `.exe` files are uploaded, limiting the application's utility for comprehensive malware analysis. Malicious non-`.exe` files will be rejected, potentially overlooking threats.

- Consideration: Extending support for additional file types would require updating `feature_extraction.py` and retraining the model to handle diverse feature sets, increasing complexity.

3. Lack of Real-Time Processing:

- Limitation: The application requires users to upload all `.exe` files before triggering batch processing in FIFO order, rather than analyzing files immediately upon upload. This two-step process (upload, then process) may be inconvenient for users needing instant results.

- Implication: Delays in analysis could hinder rapid response to potential threats, especially in time-sensitive scenarios. The FIFO processing also means users cannot prioritize specific files.

- Consideration: Adding an option for immediate processing per file, alongside batch processing, could improve flexibility, though it may complicate the interface and increase server load.

4. Limited Scalability for Large Batches:

- Limitation: The application's batch processing is CPU- and memory-intensive, particularly for large `.exe` files or high volumes (e.g., hundreds of files). Feature extraction with `pefile` and `lief` can be slow for complex executables, and temporary storage in `media/uploads` consumes disk space.

- Implication: Processing large batches may lead to performance bottlenecks, increased processing times, or disk space exhaustion, especially on minimum hardware (dual-core CPU, 4 GB RAM). The application is not optimized for high-throughput environments.

- Consideration: Implementing asynchronous processing with a task queue (e.g., Celery with Redis) or limiting batch sizes could enhance scalability, but this adds complexity.

ENHANCEMENTS

The Django-based malware detection web application can benefit from several enhancements to improve its functionality, usability, security, and scalability. These improvements aim to address existing limitations and make the tool more robust for diverse use cases while maintaining its accessibility for beginners. Below are key enhancement points outlined in detail:

- **Incorporate Prediction Probabilities:** Enhance the application by displaying the confidence scores of the LightGBM model's predictions alongside the "Malware" or "Benign" classifications. This would provide users with insight into the model's certainty, particularly for borderline cases, fostering greater trust in the results and aiding decision-making for further analysis or validation against other tools.
- **Add a Clear Uploads Option:** Introduce a feature that allows users to clear all files stored in the `media/uploads` folder with a single action, such as a dedicated button on the upload page. This would streamline the user experience by enabling easy resets of the upload queue, preventing clutter from incorrect uploads, and reducing manual effort, especially for users testing multiple files.
- **Implement User Authentication and Access Control:** Add user authentication to restrict access to the application, ensuring only authorized users can upload and process files. This enhancement would improve security by preventing unauthorized access in deployed environments, enable user-specific session management, and allow tracking of individual user activities for auditing purposes.
- **Expand Support for Additional File Types:** Extend the application's capability to process other potentially malicious file types beyond `.exe`, such as `.dll`, `.bat`, or even scripts like `.ps1`. This would make the tool more versatile, addressing a wider range of malware threats and better aligning with real-world scenarios where malware is delivered in various formats, though it would require retraining the model to handle diverse feature sets.
- **Introduce Asynchronous Processing for Scalability:** Incorporate asynchronous processing using a task queue system like Celery to handle file analysis in the background. This would improve the application's responsiveness by preventing delays during large batch processing, enhance scalability for multi-user environments, and ensure a smoother user experience, particularly when dealing with high volumes of files.
- **Enable Result Persistence for Historical Analysis:** Add functionality to store analysis results in a database, allowing users to access a history of past scans. This enhancement would enable users to revisit previous results, track trends over time, and perform audits, making the application more practical for ongoing monitoring and record-keeping in organizational settings.

- Integrate with External Threat Intelligence: Enhance the application by integrating with external threat intelligence platforms, such as VirusTotal, to cross-validate predictions or fetch additional context about analyzed files. This would improve detection accuracy, provide richer insights into potential threats, and allow the system to benefit from real-time global threat data, though it would require API access and additional configuration.
- Provide Detailed Error Reporting and Logging: Improve error handling by offering more detailed feedback to users when issues occur, such as during feature extraction failures, and implement comprehensive logging for debugging purposes. This would help users troubleshoot problems more effectively, enhance transparency, and make the application more maintainable, especially in production environments.
- Add Real-Time Processing Option: Allow users the option to process files immediately upon upload, in addition to the existing batch processing in FIFO order. This would cater to users needing quick results for individual files, improve flexibility, and reduce delays in time-sensitive scenarios, although it may increase server load and require careful resource management.
- Implement File Upload Limits and Quotas: Introduce restrictions on the number or size of files users can upload to prevent server overload and manage disk space more effectively. This would enhance the application's stability by avoiding resource exhaustion, particularly in multi-user or deployed scenarios, and ensure consistent performance under varying workloads.

These enhancements collectively aim to make the malware detection web application more user-friendly, secure, and adaptable to real-world cybersecurity needs. Prioritizing simpler improvements like prediction probabilities and the clear uploads feature can provide immediate benefits, while more advanced enhancements like asynchronous processing and external integrations can be pursued as the project evolves, ensuring a balance between functionality and development complexity for beginners.

CONCLUSION

The Django-based malware detection web application represents a significant step toward providing an accessible, user-friendly tool for analyzing Windows executable (`.exe`) files for potential malware threats. By leveraging a pre-trained LightGBM machine learning model and feature extraction libraries such as `pefile` and `lief`, the application effectively classifies files as "Malware" or "Benign" through a streamlined process of single-file uploads, secure storage in a `media/uploads` folder, and batch processing in a First-In-First-Out (FIFO) order. Developed in a Linux environment using PyCharm and Django 5.2.1, the system achieves its core objectives of simplicity, functionality, and clarity, making it suitable for beginners and non-technical users while demonstrating the practical application of machine learning in cybersecurity.

The project's modular design, with clear separation of concerns across forms, views, templates, and feature extraction logic, ensures maintainability and sets a foundation for future enhancements. Proposed improvements, such as adding prediction probabilities, a clear uploads feature, user authentication, support for additional file types, asynchronous processing, and result persistence, address key limitations and have the potential to transform the application into a more robust, scalable, and secure tool. These enhancements would improve transparency, usability, security, and versatility, making the system viable for broader use cases, including organizational or networked environments.

Despite its limitations, such as reliance on a static model, restricted file type support, and lack of real-time processing, the application successfully fulfills its goal of providing an intuitive platform for malware detection. It bridges the gap between advanced machine learning techniques and end-user accessibility, contributing to the open-source cybersecurity community. The project also serves as an educational tool, offering valuable insights into web development, machine learning integration, and secure file handling for beginners using Django. Moving forward, implementing the proposed enhancements and testing the application in diverse scenarios will further strengthen its capabilities, ensuring it remains a relevant and effective solution in the ever-evolving landscape of cybersecurity threats.

REFERENCES

1. LightGBM Documentation: The LightGBM documentation provides detailed information on using the LightGBM machine learning library, including model loading, prediction, and probability estimation, essential for understanding the classification logic and potential enhancements like prediction probabilities.
2. pefile Documentation: The pefile library documentation offers insights into extracting portable executable (PE) features from .exe files, such as entropy and section attributes, which are critical for the feature extraction process in the application.
3. LIEF Documentation: The LIEF library documentation details its capabilities for parsing and analyzing executable files, complementing pefile in extracting features like symbols and imports, used in the application's feature extraction module.
4. Python Documentation: The official Python 3.12 documentation provides resources on Python's standard library, including modules like os, glob, and pickle, which are used for file handling, sorting by creation time (FIFO logic), and model loading in the application.
5. Pandas Documentation: The Pandas documentation explains DataFrame operations, crucial for structuring extracted features into a format compatible with the LightGBM model for prediction.