🔐 From Detection to Defense: Building an AI-Driven Hybrid Cybersecurity System

Author: [Chimdessa Tesfaye, Getachewu Getu, Rediet Bekele, Meseret Ghebresilassie]

Introduction: A Modern Threat Landscape Needs Smarter Defenses

In today’s hyper-connected world, malware attacks and cyber threats are becoming increasingly sophisticated and harder to detect using traditional tools alone. As cybercriminals adopt advanced techniques, security analysts and institutions must also evolve their strategies to stay ahead.

This blog post presents a full-cycle AI-powered hybrid cybersecurity system that combines **malware detection from executable files** and **threat intelligence analysis** from unstructured text using advanced Natural Language Processing (NLP) models. Our goal is to automate the two core pillars of defense: **real-time threat detection** and **strategic threat insight**.

By promoting cybersecurity education and awareness, the project supports **Sustainable Development Goal 4 (Quality Education)**. At the same time, it contributes to **SDG 9 (Industry, Innovation, and Infrastructure)** by strengthening digital infrastructure through cutting-edge AI technologies. Furthermore, by enhancing institutional resilience and safeguarding systems from cybercrime, it aligns with **SDG 16 (Peace, Justice, and Strong Institutions)**.

By developing a robust AI-based cyber defense system, this project contributes to a safer digital environment, reduces cyber threats, and fosters cybersecurity innovation. Whether you're an engineer, researcher, or cyber enthusiast, this post walks you through our journey of building a production-ready system that leverages machine learning, BERT, spaCy, and Streamlit to defend against evolving cyber threats.

Figure 1: Malware Detection https://tresorit.com

I. The Problem: Fragmented Security Approaches

Cyber defense often splits into two isolated functions:

- **Reactive Detection**: Signature-based or behavior-based detection systems that scan executables.
- **Proactive Intelligence**: Human analysts manually reading threat reports and trying to identify patterns.

Both methods are resource-intensive and non-scalable. Malware detection requires understanding static binary features, while extracting threat behavior from texts needs context-aware analysis — both of which demand automation to be practical.

Part 1: Intelligent Malware Detection Using PE Header Features

Here we want to classify malware and legitimate software by analyzing extracted features from PE including collecting, cleaning and comprehending data through EDA, along with feature selection and transformation to improve machine learning models like Linear regression, random forest classifier, XGBoost, and Deep Neutral Network.

Step 1: Data Collection

Malware Dataset is available in various public domain and is provided by "*Virusshare.com*" to provide security researchers. The size of dataset is [56645, 54] of (mal=41323, ben=15321). This file included legitimate columns used as binary labels - 0 for legitimate software and 1 for malware - this served as the target variable for classification tasks.

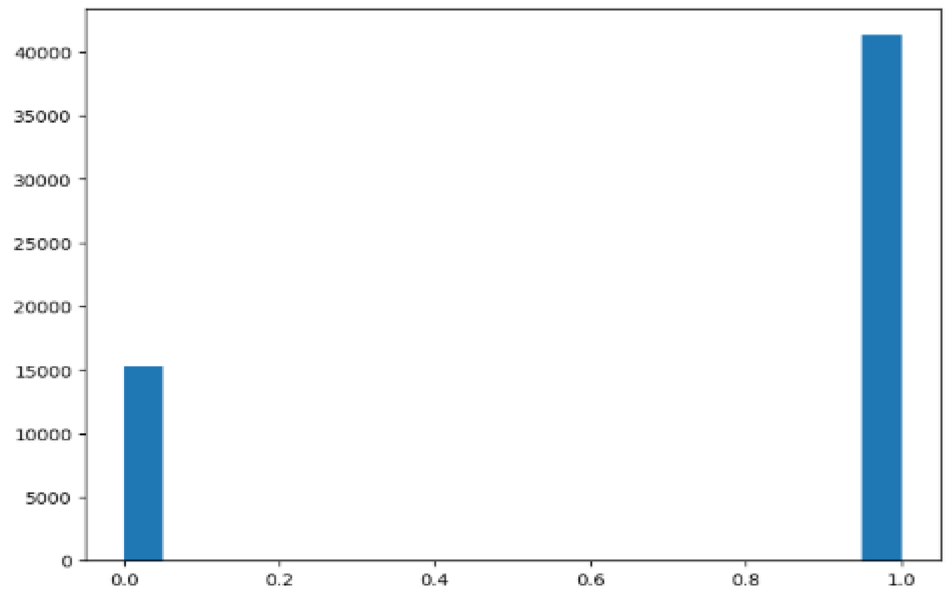


Figure 2: Target class distribution (0-legitimate and 1-malware)

Step 2: Data Cleaning

In order to ensure the quality and integrity of the data collected for accurate model prediction, cleaning of the data is essential.

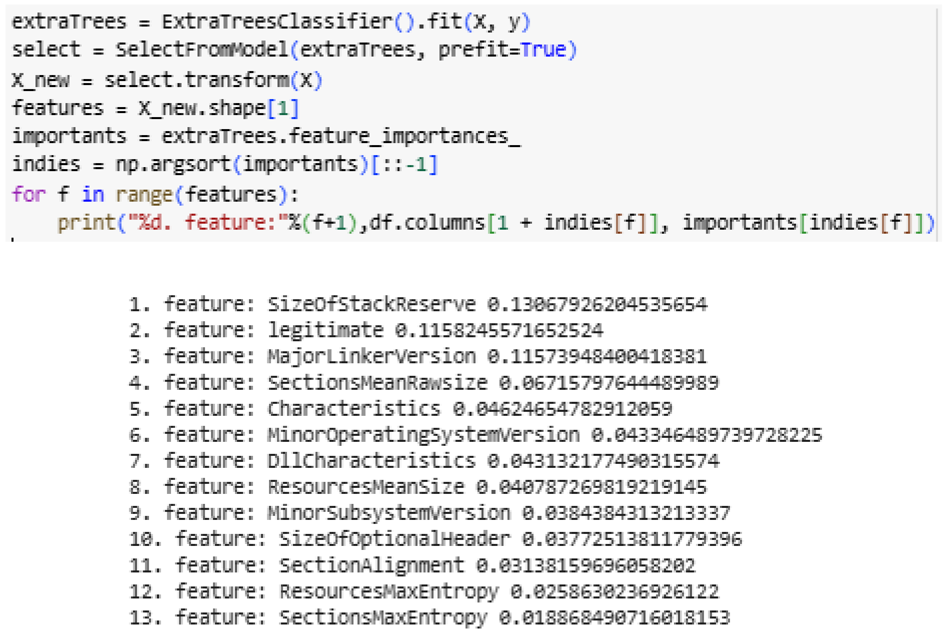
Missing values: Checks performed showed no missing values in any of the 54 columns.
Label Balance: we checked for the skewed distribution of malware vs benign samples and made sure it was reasonable for model training. In this case unnecessary data are removed like Name of the file, and ID.

Step 3: Feature Extraction from .exe Files

Using the Python **pefile** library, we extracted 54 critical features from Windows executables (PE files), including:

- Header details: **Machine**, **Characteristics**, **ImageBase**, etc.
- Section entropy and size: Indicators of packing or obfuscation
- Resource and import table metadata

Here we perform feature selection using the Extra-Trees Classifier to identify and select the most important features from our dataset. So, based on this method we get 13 features that are more importance for our model development.



```
extraTrees = ExtraTreesClassifier().fit(X, y)
select = SelectFromModel(extraTrees, prefit=True)
X_new = select.transform(X)
features = X_new.shape[1]
importants = extraTrees.feature_importances_
indies = np.argsort(importants)[::-1]
for f in range(features):
    print("%d. feature:"%(f+1),df.columns[1 + indies[f]], importants[indies[f]])
```

```
1. feature: SizeOfStackReserve 0.13067926204535654
2. feature: legitimate 0.1158245571652524
3. feature: MajorLinkerVersion 0.11573948400418381
4. feature: SectionsMeanRawsize 0.06715797644489989
5. feature: Characteristics 0.04624654782912059
6. feature: MinorOperatingSystemVersion 0.043346489739728225
7. feature: DllCharacteristics 0.043132177490315574
8. feature: ResourcesMeanSize 0.040787269819219145
9. feature: MinorSubsystemVersion 0.0384384313213337
10. feature: SizeOfOptionalHeader 0.03772513811779396
11. feature: SectionAlignment 0.03138159696058202
12. feature: ResourcesMaxEntropy 0.0258630236926122
13. feature: SectionsMaxEntropy 0.018868490716018153
```

Figure 3: Feature selection code snapshot

Step 4: Model Building and Training

To build an effective malware detection system, **we experimented with a variety of machine learning and deep learning models** on the extracted PE header features. Each model was chosen based on its strengths and typical application in classification tasks. Below is a detailed explanation of the models we used:

Logistic Regression (LR)

We used logistic regression as a foundational baseline due to its simplicity and interpretability. LR models the probability of the binary outcome (malware or benign) as a logistic function of the input features. Despite being a linear classifier, it provides a strong benchmark for comparison with more complex models.

Decision Tree (DT)

We used decision trees to model non-linear relationships by recursively partitioning the feature space based on optimal split points. Decision trees are easy to visualize and interpret, but they tend to overfit, especially when grown deep.

Random Forest (RF)

To overcome decision tree overfitting, we used a random forest, which aggregates the predictions of multiple decision trees trained on random subsets of data and features. This ensemble approach reduces variance and improves robustness and accuracy.

XGBoost

We used XGBoost, a highly optimized gradient boosting framework, due to its ability to efficiently build strong predictive models by sequentially correcting errors from weak learners. Its handling of feature interactions and regularization makes it highly accurate and resistant to overfitting.

Deep Neural Network (DNN)

We implemented a fully connected feedforward neural network with multiple hidden layers. The DNN learns complex, non-linear feature representations through backpropagation, enabling it to capture subtle patterns in the data.

Convolutional Neural Network (CNN) We applied a 1D CNN to leverage spatial relationships within the feature vector by convolving filters across the features. CNNs are effective at extracting local patterns and hierarchies from structured data, often used in image and signal processing.

Long Short-Term Memory (LSTM) We used LSTM networks, a type of recurrent neural network, to explore whether sequential dependencies existed in the PE feature vectors. LSTMs are designed to capture long-term dependencies in sequences and can model temporal or ordered relationships.

Hybrid Model: CNN-LSTM to combine the strengths of convolutional and recurrent architectures, we developed a hybrid CNN-LSTM model. This model leverages the CNN layers for spatial feature extraction from the PE header features, followed by LSTM layers to capture any sequential or temporal dependencies within those features.

• CNN Layers: These apply convolutional filters that automatically learn important spatial patterns and local feature interactions.

• LSTM Layers: These sequentially process the CNN-extracted features to learn longer-range dependencies and contextual relationships.

The hybrid model aimed to improve classification accuracy by integrating spatial and sequential learning mechanisms. Although the CNN-LSTM model performed better than standalone CNN or LSTM models, it did not surpass the classical XGBoost in overall accuracy or training efficiency.

- ***Split Train/Test:***

The classification data sets are split into 80% training and 20% testing while ensuring that the class distributions remain balanced and result in good model performance.

Step 5: Model Evaluation

In this study, various experiments were conducted and evaluated using state-of-the-art evaluation metrics including accuracy, precision, recall, F1 score, ROC AUC, loss, and training time. These metrics help provide a comprehensive understanding of each model's performance in malware detection and threat intelligence classification tasks. The results are summarized in the tables below, followed by explanatory notes.

Model used	F1 score on testing data	Accuracy score on testing dataset	Accuracy on training dataset	Training time (sec)
XGBoost	99.11%	99.52%	99.88%	1.03
RF	99.05%	98.62%	98.44%	5.39
LR	96.03%	94.45%	94.32%	3.24
DNN	58.18%	72.4%	73.11%	10.8

Experiment 1: Malware Detection Using Traditional Machine Learning Models

Traditional machine learning algorithms, especially ensemble methods like XGBoost and Random Forest, demonstrated excellent accuracy and F1 scores, indicating strong capability in classifying malware based on extracted PE file features. Logistic Regression, while faster, showed relatively lower accuracy, and the deep neural network struggled possibly due to insufficient feature engineering or model complexity.

Model used	Accuracy	Precision	Recall	F1 Score	ROC AUC	Loss	Training Time
CNN	98.9%	98.3%	98.05%	98.17%	99.88%	0.04	5.4
LSTM	97%	94.14%	96.08%	95.1%	99.51%	0.08	5.1
Hybrid CNN_LSTM	98.98%	98.26%	98.37%	98.32%	99.88%	0.03	5.3

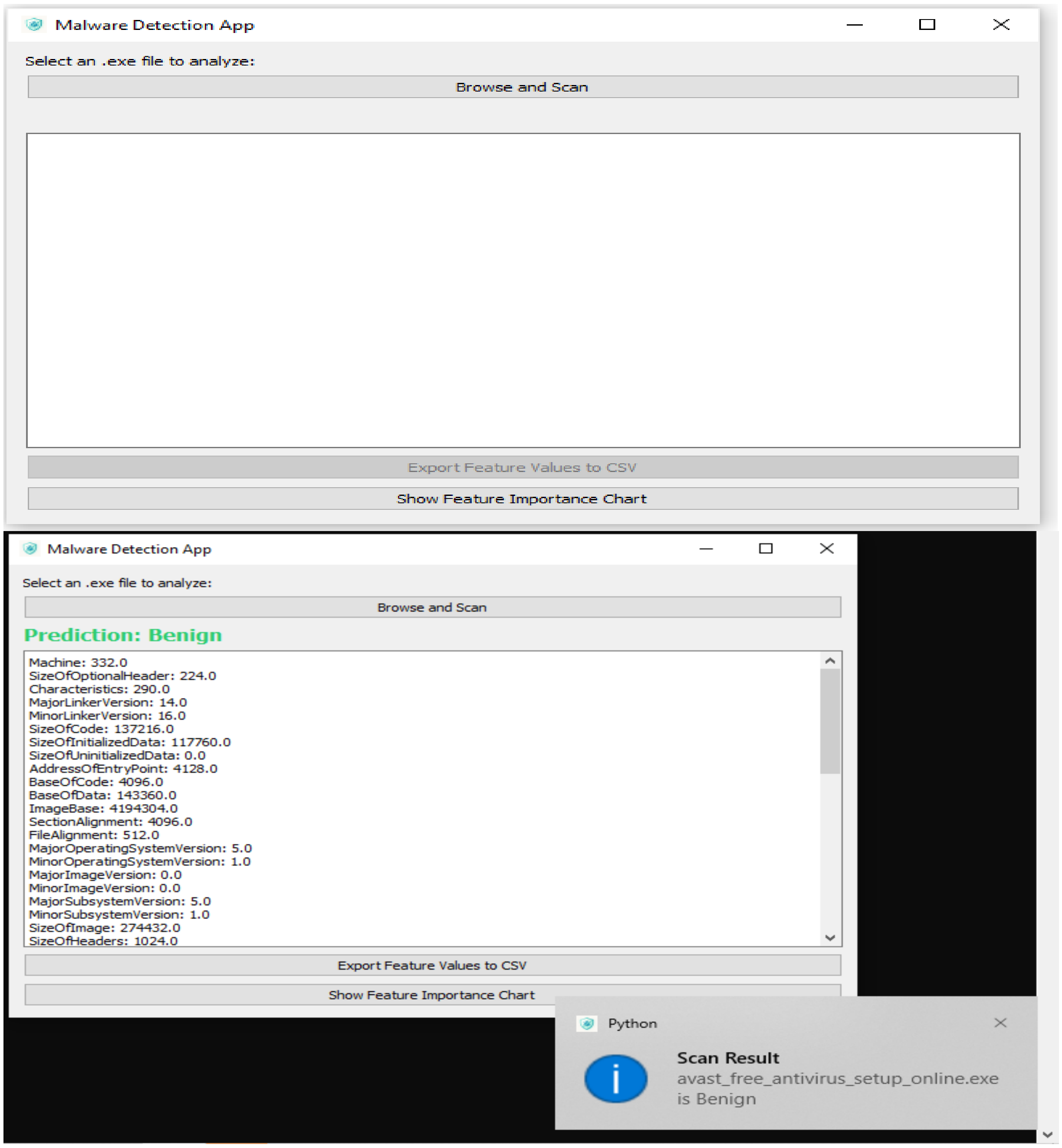
Experiment Two: Hybrid CNN-LSTM Model for Malware Detection

In this experiment, we applied a hybrid CNN-LSTM model to classify malware from PE header features. The CNN layers captured spatial patterns, while LSTM handled sequential dependencies across features. This deep learning approach delivered strong results with an accuracy of 98.98%, precision of 98.26%, recall of 98.37%, and an F1 score of 98.32%. The ROC AUC score reached 99.88%, indicating excellent classification performance. The model also showed efficient training with a low loss of 0.0327 and training time of 5.3 seconds. These results show the hybrid model's effectiveness in detecting malware with both precision and speed.

Here after doing these two experiments, we selected XGBoost classifier selected for its performance and interpretability trained on a labelled dataset of malicious and benign executables classification.

Step 6: Model Deployment

We deployed this in a PyQt5 desktop app that shows real-time notifications, predictions (Benign or Malware).


Malware Detection App
Select an .exe file to analyze:
Browse and Scan
Export Feature Values to CSV
Show Feature Importance Chart
Malware Detection App
Select an .exe file to analyze:
Browse and Scan
Export Feature Values to CSV
Show Feature Importance Chart
Python

Prediction: Benign

```
Machine: 332.0
SizeOfOptionalHeader: 224.0
Characteristics: 290.0
MajorLinkerVersion: 14.0
MinorLinkerVersion: 16.0
SizeOfCode: 137216.0
SizeOfInitializedData: 117760.0
SizeOfUninitializedData: 0.0
AddressOfEntryPoint: 4128.0
BaseOfCode: 4096.0
BaseOfData: 143360.0
ImageBase: 4194304.0
SectionAlignment: 4096.0
FileAlignment: 512.0
MajorOperatingSystemVersion: 5.0
MinorOperatingSystemVersion: 1.0
MajorImageVersion: 0.0
MinorImageVersion: 0.0
MajorSubsystemVersion: 5.0
MinorSubsystemVersion: 1.0
SizeOfImage: 274432.0
SizeOfHeaders: 1024.0
```

Scan Result
avast_free_antivirus_setup_online.exe is Benign

Figure 4: PyQt5 app for malware detection

Part 2: Natural Language Processing (NLP) Pipeline for Threat Intelligence

Cyber threat reports contain high-level intelligence about vulnerabilities, attack vectors, and adversaries. However, manually reading hundreds of articles isn’t scalable.

We developed an NLP pipeline that can:

- Extract named entities like CVEs, malware names, and organizations using spaCy.
- Classify text as threat-relevant or not using a fine-tuned BERT model.
- Detect behavior tags such as *initial access*, *data exfiltration*, or *credential access*.

The Process: Fine-Tuning BERT on Threat Reports

Step 1: Data Collection

We collected open-source cyber threat intelligence articles from reputable sources such as *The Hacker News* using RSS scraping and saved them in a CSV format. The CSV file includes over 90 entries with the following fields:

- title: Headline of the article
- summary: Core description of the threat activity
- published: Publication date
- link: Original article URL

We combined the title and summary fields to form a single input text for NLP processing.

Step 2: Text Preprocessing

To prepare the unstructured threat intelligence texts for training and classification, we performed the following preprocessing steps:

1. Text Cleaning

Although BERT can handle raw text directly, minimal cleaning was applied to improve consistency:

- Removed newline characters (`\n`), tabs, and excessive whitespaces.
- Replaced malformed symbols (e.g., smart quotes like â€™, â€‟) with standard characters.
- Lowercased the text for normalization (optional with BERT, but useful for consistency across models).

2. Label Normalization

The classification labels (`LABEL_0`, `LABEL_1`) were mapped to integers:

- `0`: Non-threat-related text
- `1`: Threat-related intelligence

3. Tokenization using BERT Tokenizer

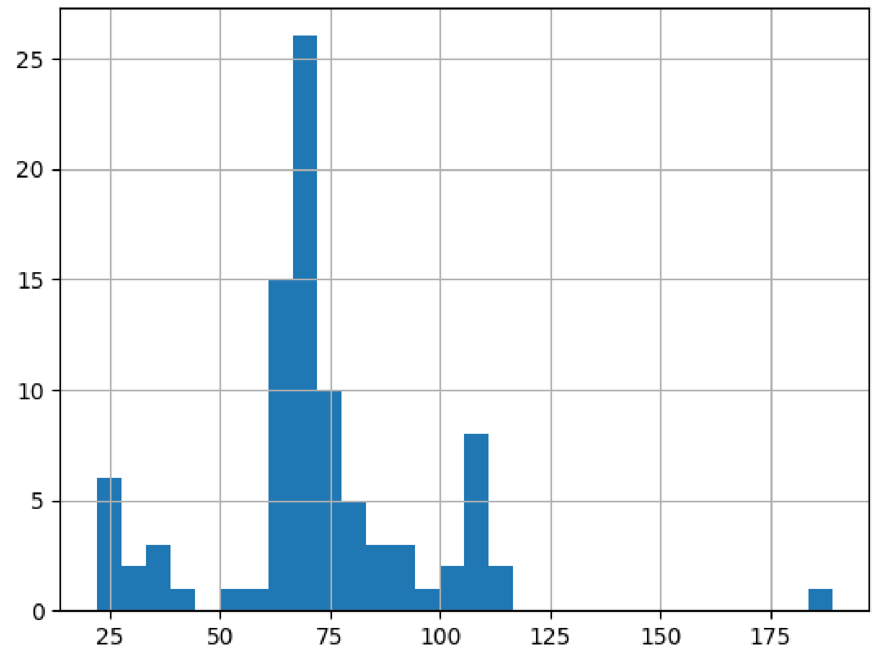
We used the Hugging Face `BertTokenizer` to tokenize and encode the text:

- **Subword tokenization**: Breaks words into smaller units, improving vocabulary coverage.
- **Special tokens**: Added `[CLS]` at the beginning and `[SEP]` at the end of each input.
- **Padding and truncation**: Ensured each input sequence had a fixed length (max 512 tokens).
- **Attention masks**: Generated to distinguish real tokens from padding.

Step 3: Explanatory Data Analysis

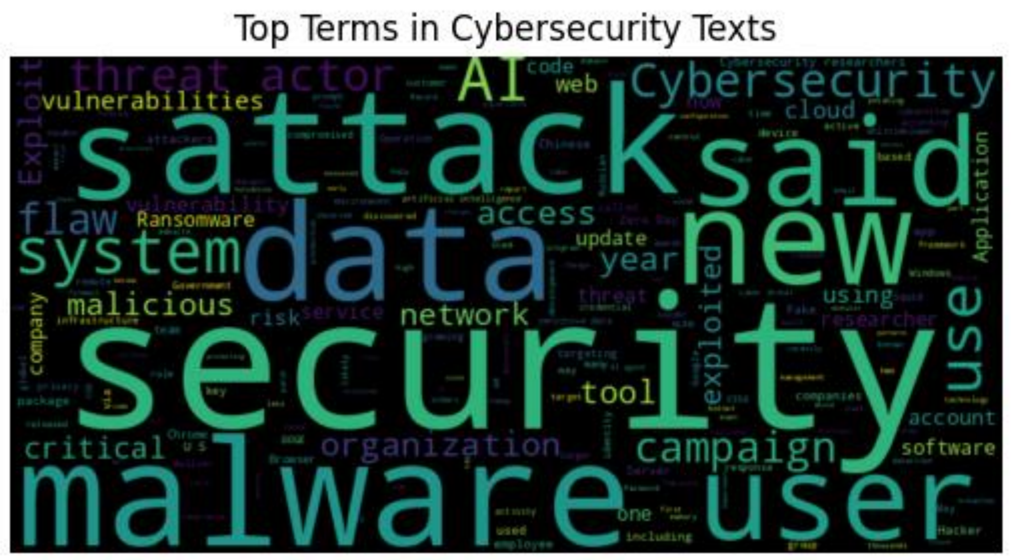
- **Text Length Distribution**

Found that most threat descriptions ranged between **50 to 125 words**, allowing us to fix a max token length of 512 for BERT input.



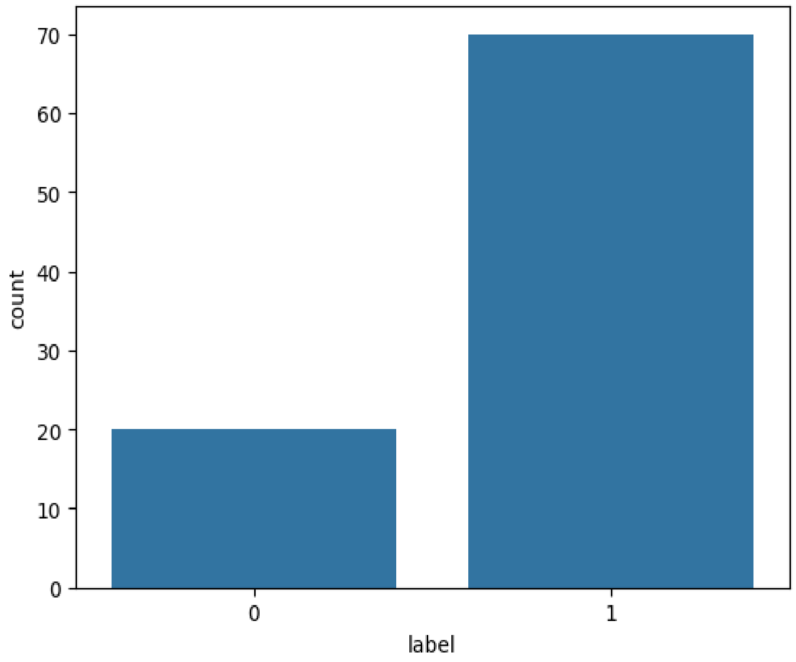
- **Word Cloud of Frequent Terms**

Common terms like "malware", "Malware", "security", and "attacks" appeared frequently, validating the threat-domain relevance of the dataset.



- **Label Distribution Visualization**

This helped confirm if the classes were balanced or if up/down-sampling was needed. As it appearing on the below graph the class is balanced.



Step 4: Named Entity Recognition (NER) and Threat Behavior Extraction

We utilized **spaCy**'s en_core_web_sm model for NER to detect and extract key cybersecurity-related entities such as:

- Threat actor names (ORG), malware names (PRODUCT), dates (DATE), and locations (GPE, LOC)

For behavior extraction, we used a keyword-matching approach against a predefined MITRE ATT&CK-inspired behavior lexicon (e.g., "initial access", "data exfiltration", "persistence"). This allowed detection of attacker tactics described within the summaries.

Step 5: Text Classification Dataset Preparation

Using the output of the previous step, we manually labelled each document as:

- LABEL_1: Texts describing active or potential cyber threats
- LABEL_0: Informational or non-threat related text

This labelled dataset was saved as **threat_classification_dataset.csv** with two columns: text and label.

Step 6: Model Fine-Tuning

We fine-tuned a **BERT (bert-base-uncased)** model for binary text classification using the Hugging Face Transformers library.

- **Tokenization**: Used BertTokenizer to tokenize and pad input texts
- **Training**: Used the Trainer class with:
    - 80/20 train/validation split
    - CrossEntropyLoss
    - Evaluation on each epoch
- **Evaluation Metrics**: Accuracy was the primary metric, with up to 88.8% accuracy after 10 epochs. Training loss reduced to 0.0023, while validation loss stabilized around 0.5.

Step 7: Evaluation

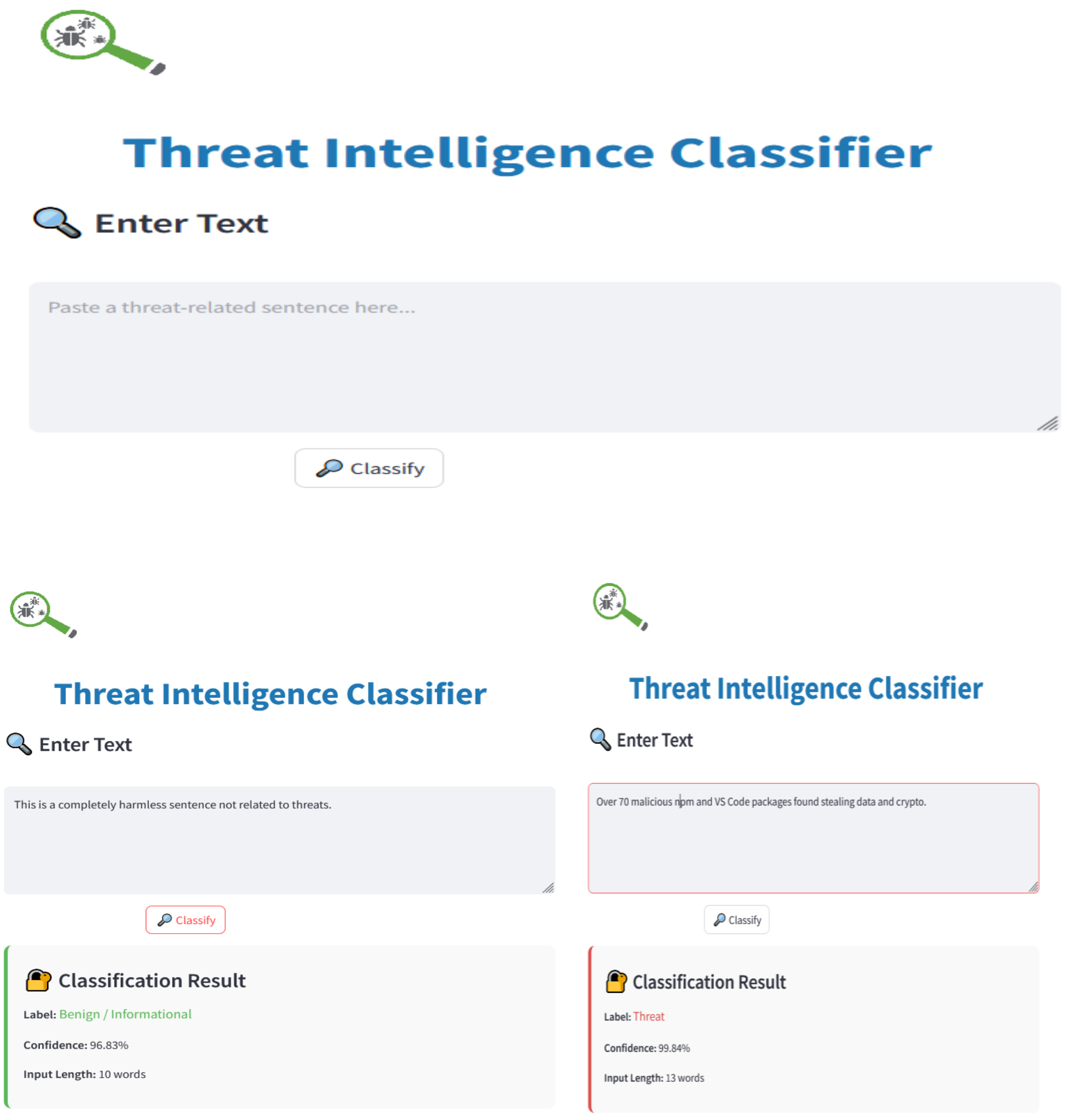
Here's a sample of the model's epoch-wise performance:

Epoch	Training Loss	Validation Loss	Accuracy
1	N/A	0.495	77.8%
3	0.387	0.405	83.3%
7	0.006	0.424	88.8%
10	0.002	0.518	88.8%

While training loss continued to decrease, validation loss plateaued after epoch 7, suggesting a good time to apply **early stopping** to prevent overfitting.

Step 8: Inference and Visualization
We developed a **Streamlit web app** that allows:

- Real-time classification of user-provided threat-related text
- Display of the predicted label and confidence score
- Future-ready design for integrating entity and behavior insights



Threat Intelligence Classifier

Enter Text

Paste a threat-related sentence here...

Classify

Threat Intelligence Classifier

Enter Text

This is a completely harmless sentence not related to threats.

Classify

Classification Result

Label: Benign / Informational

Confidence: 96.83%

Input Length: 10 words

Threat Intelligence Classifier

Enter Text

Over 70 malicious npm and VS Code packages found stealing data and crypto.

Classify

Classification Result

Label: Threat

Confidence: 99.84%

Input Length: 13 words

Figure 5: streamlit app for threat intelligence classifier

Step 9: Deployment and Integration

While the malware detection module runs as a **PyQt5 desktop app**, the NLP classifier runs in a separate **Streamlit dashboard** to ensure modularity and performance.

This hybrid system combining static binary analysis and textual threat intelligence enables a broader AI-driven cybersecurity posture.

Combined Power: Why Hybrid Systems Matter

Together, our project forms a **hybrid cyber defense system**:

Component	Technology Used	Purpose
Malware Detector	pefile, xgboost, PyQt5	Detect malicious executables
Threat Classifier	BERT, spaCy, Streamlit	Extract strategic threat insights
Integration Potential	REST APIs / Webhooks	Feed results to SIEM / SOC

This integration of **static binary analysis** and **semantic intelligence extraction** is powerful — combining frontline protection with strategic foresight.

Results and Evaluation

Module	Accuracy	Interface
XGBoost Malware Detector	99%	Desktop (PyQt5)
BERT Threat Classifier	88%	Web App (Streamlit)

Future Work

We plan to extend the system with:

- Integrate both tools into a unified dashboard.
- Use real-time threat feed APIs (e.g., VirusTotal, MISP).
- Extend threat classification to multi-class (APT, phishing, ransomware).
- Expand NER to custom cyber tags using spaCy or Flair.

Conclusion

In a world where threats evolve daily, a hybrid AI-driven defense system provides both precision and intelligence. By combining local malware detection with NLP-powered threat classification, we've built a system that can protect endpoints *and* help analysts see the bigger picture.

This project reflects the critical role of AI in shaping a safer digital future. We encourage others in academia and industry to build upon these foundations and innovate further.

📌 References

- pefile GitHub
- Breiman, L., Friedman, J., Olshen, R., & Stone, C. (1984). Classification and regression trees. Wadsworth International Group.
- Breiman, L. (2001). Random forests. *Machine Learning*, 45(1), 5-32.
- XGBoost Documentation
- HuggingFace Transformers
- The Hacker News
- Streamlit
- Pant, Y. (2022). Malware Detection in Executable Files using XGBoost Algorithm. National College of Ireland.
- Narayanan, M. E., & Muthukumar, B. (2021). Malware Classification Using XGBoost with Vote-Based Backward Feature Elimination. Turkish Journal of Computer and Mathematics Education.
- Kumar, R., & Geetha, S. (2020). Malware Classification Using XGBoost – Gradient Boosted Decision Tree. ASTES Journal.
- Arazzi, M. A., et al. (2023). NLP-Based Techniques for Cyber Threat Intelligence. arXiv.
- Ogundairo, O., & Broklyn, P. (2024). Natural Language Processing for Cybersecurity Incident Analysis. EasyChair Preprints.
- Demirol, D., Das, R., & Hanbay, D. (2025). A Novel Approach for Cyber Threat Analysis Systems Using BERT Model from Cyber Threat Intelligence Data. _Symmetry_.
- Aghaei, E., et al. (2022). SecureBERT: A Domain-Specific Language Model for Cybersecurity. arXiv.