MALWARE DETECTION USING MACHINE LEARNING

MAJOR PROJECT

UNDER SUPERVISION OF PROF. AMJAD SIR

20BCS087 NABEEL MOHAMMAD RIZWAN

20BCS081 ARSH ALI KHAN

CONTENTS

Introduction

- The persistent and evolving **threat of malware** poses a formidable challenge in today’s digital landscape. In an era defined by digital connectivity, cybersecurity has become a paramount concern.
- **Machine learning**, a subfield of artificial intelligence, has emerged as a formidable ally in the battle against malware.
- By leveraging advanced algorithms, data-driven insights, and pattern recognition, machine learning empowers cybersecurity experts to detect and combat malware with unprecedented accuracy and agility.
- We will examine how machine learning techniques enable the identification of malware based on behavioral patterns, anomalous activities, and code analysis, ultimately enhancing our ability to thwart cyber threats and protect the digital realm from malicious incursions.

Literature Review

The impact of malicious software are getting worse day by day. Malicious software or malwares are programs that are created to harm, interrupt or damage computers, networks and other resources associated with it. Malwares are transferred in computers without the knowledge of its owner.	[8] A Study on Malware and Malware Detection Techniques, , DOI: 10.5815, 08 March 2018
Viruses: "A computer program usually hidden within another seemingly innocuous program that produces copies of itself and inserts them into other programs or files, and that usually performs a malicious action (such as destroying data)". **Worms**: Spafford (1989) defines a worm as "a program that can run independently and can propagate a fully working version of itself to other machines." This type of harmful code is predominantly prevalent in networks such as the Internet. **Spyware**: Spyware refers to software that access confidential information from the user and pass it on to another entity without informing the user of this action. Thus it takes control over the user's system without asking for his/her consent regarding the matter. **Backdoor**: The method of evading usual authentication procedures, particularly over connections such as the Internet, is known as backdoor. One or more backdoors may be installed into a system without the user's knowledge to make the system susceptible to outside attacks.	[7] Introduction to Malware and Malware Analysis: A brief overview, ISSN: 2321-7782, October 2016

Literature Review

Static Analysis - **Analyzing malicious software without executing it**, but by merely inspecting the program is called static or code analysis. It is usually performed by taking each part of the binary file and studying each component thoroughly without **actually executing it**. **Dynamic Analysis** - In dynamic analysis, **the process is executed and the system is observed while the changes that occur are noted**. It is to be kept in mind that the malware needs to be executed in a safe environment, preferably in a virtual machine. It is also wise to take a snapshot of the virtual machine before the malware binaries are executed so as to ensure that the safe state can be returned to and the proper changes can be noted.	[7] Introduction to Malware and Malware Analysis: A brief overview, ISSN: 2321-7782, October 2016
Signature based Technique When a malware is written, a sequence of bit generally known as **signature** is embedded in its code which can later use to identify from which family this malware belongs to. The signature based detection technique is used by most of the antivirus programs. The antivirus program disassemble the code of the infected file and search for the pattern that belong to a malware family. Signatures of the malwares are maintained in database and then further used for comparison in detection process. This kind of detection technique is also known as string or pattern scanning or matching. It can be static, dynamic or hybrid as well.	[8] A Study on Malware and Malware Detection Techniques, DOI: 10.5815, 08 March 2018

Literature Review

Understanding how **LSTM**'s can be used for malware detection overall. For this they had to disassemble the malware and benign files first to properly generate the datasets.	[1] Malware Detection with LSTM using Opcode Language: Renjie Lu: arXiv:1906.04593v1
This paper has combined **1-D CNN and bi-directional LSTM** to classify android **Dalvik bytecodes** as either malware or non-malware. They have shown that one-dimensional CNNs detect local features and reduce the feature size and then have combined that with bi-directional long-short term memory network (LSTM) to detect malware.	[2] SP.W., Hwang, YS. (2021). Malware Detection by Merging 1D CNN and Bi-directional LSTM Utilizing Sequential Data .Springer 10.1007/978-981-33-6385-4_16
This paper has used a **1-D CNN** instead of the standard 2-D CNN to classify files into malware or benign. It argues that the **1-D CNN is better than 2-D** as the 2-D CNN suffers from problematic semantic interpretations.	[3] Malware Detection Using 1-Dimensional Convolutional Neural Networks:10.1109/EuroSPW.2019.00034
This paper has shown a technique called as **Instruction2vec to generate instruction embeddings for assembly instructions**. It is based on Word2Vec itself, and they have provided an implementation as well to generate embeddings for a given corpus of assembly instructions. They have shown their embeddings to be more accurate than those generated by standard Word2Vec and Binary2Img.	[4] Lee, & Kwon, Hyun & Choi, Sang-Hoon & Lim, Seung-Ho & Baek, Sung Hoon & Park(2019). Instruction2vec: Efficient Preprocessor of Assembly Code 9. 4086. 10.3390/app9194086.

Why Machine Learning for Malware Detection?

- **Adaptability to new threats-**

➢ Machine learning models can **adapt to new and previously unseen malware threats** without requiring manual updates. They learn from historical data and can identify patterns indicative of malicious behavior.

- **Complex Pattern Recognition-**

➢ Malware often **exhibits complex patterns** that are challenging to capture with traditional rule-based methods. Machine learning models, particularly deep learning and ensemble techniques, excel at identifying intricate patterns in large datasets.

Why Machine Learning for Malware Detection?

- **Behavioral Analysis-**

➢ Machine learning models can **analyze the behavior** of files and processes, **distinguishing between normal and malicious activities**. This is particularly useful for detecting novel or evolving malware that may not have known signatures.

- **Reduced False Positives-**

➢ Machine learning models can help compared to rule-based systems.

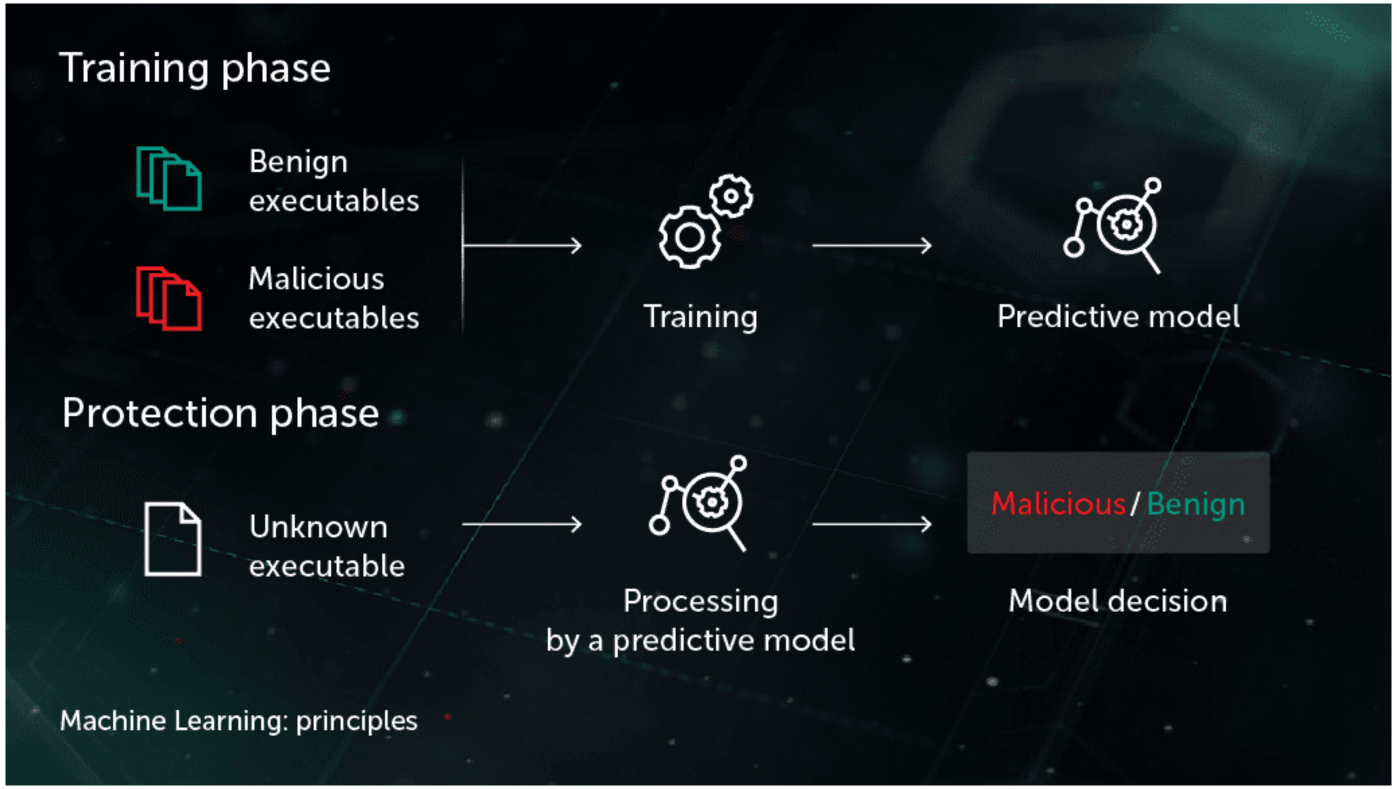
- **Scalability-reduce false positives**

➢ **ML models can efficiently process and analyze large datasets**, making them suitable for handling the vast amounts of data generated in the cybersecurity domain.

- **Continuous Learning-**

➢ In the field of Malware and System virus, It is important that the **product keeps learning unknown and new malwares**.

Malware Detection Process



The Impact Of AI and ML on Cybersecurity (Source: acte.in)

Algorithms and Techniques

- **Static Technique –**
    - In this technique, **we do not run the PE file** (.exe) in the system or virtual environment.
    - Here **We extract important features from the PE file and apply analysis** from database of known malware features and come to a decision whether it is malicious or not.
    - In our major project, we will train model on large dataset of malware features which may include signature (md5), Machine, size of code, sizeofoptionalheader, etc. to predict whether a given file is malicious or not.
- **PE file –**
    - These are Portable executable files on windows with **.exe format**.

Techniques

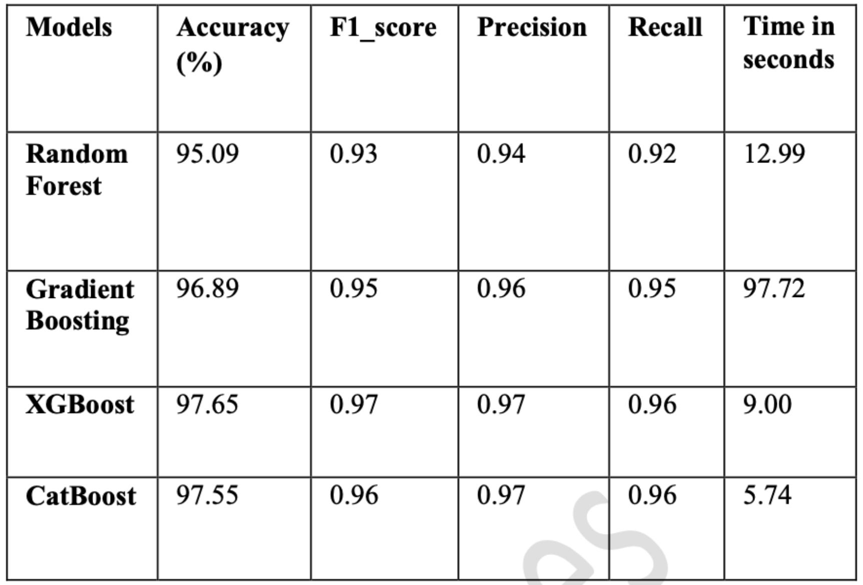
1. Combining the power of **Machine Learning and Deep Learning**, our approach leverages **XGBoost**, a state-of-the-art ML algorithm, to classify data stored in the widely accessible CSV format, unlocking the benefits of rapid processing and interpretation.
2. Simultaneously, our **Deep Learning** solution harnesses the minor details of assembly instructions, stored in text files, to unveil insights that were previously obscured, providing a holistic understanding of complex systems and processes utilizing **1D-CNN & LSTM** Algorithm.

XGBoost (eXtreme Gradient Boosting)

- It is an implementation of **Gradient Boosting**. In gradient boosting, each predictor corrects its predecessor's error.
- XGBoost is an **optimized distributed gradient boosting library** designed for efficient and scalable training of machine learning models. It is an ensemble learning method that combines the predictions of multiple weak models to produce a stronger prediction.
- XGBoost stands for "Extreme Gradient Boosting" and it has become one of the most popular and widely used machine learning algorithms due to its ability to **handle large datasets** and its ability to achieve state-of-the-art performance in many machine learning tasks such as classification and regression.
- In this algorithm, decision trees are created in sequential form. **Weights play an important role in XGBoost**. Weights are assigned to all the independent variables which are then fed into the decision tree which predicts results. The weight of variables predicted wrong by the tree is increased and these variables are then fed to the second decision tree. These individual classifiers/predictors then ensemble to give a strong and more precise model. It can work on regression, classification, ranking, and user-defined prediction problems. [1]

Why use XGBoost?

- One of the key features of XGBoost is its **efficient handling of missing values**, which allows it to handle real-world data with missing values **without requiring significant pre-processing**.
- Additionally, XGBoost has **built-in support for parallel processing**, making it possible to **train models on large datasets in a reasonable amount of time**.

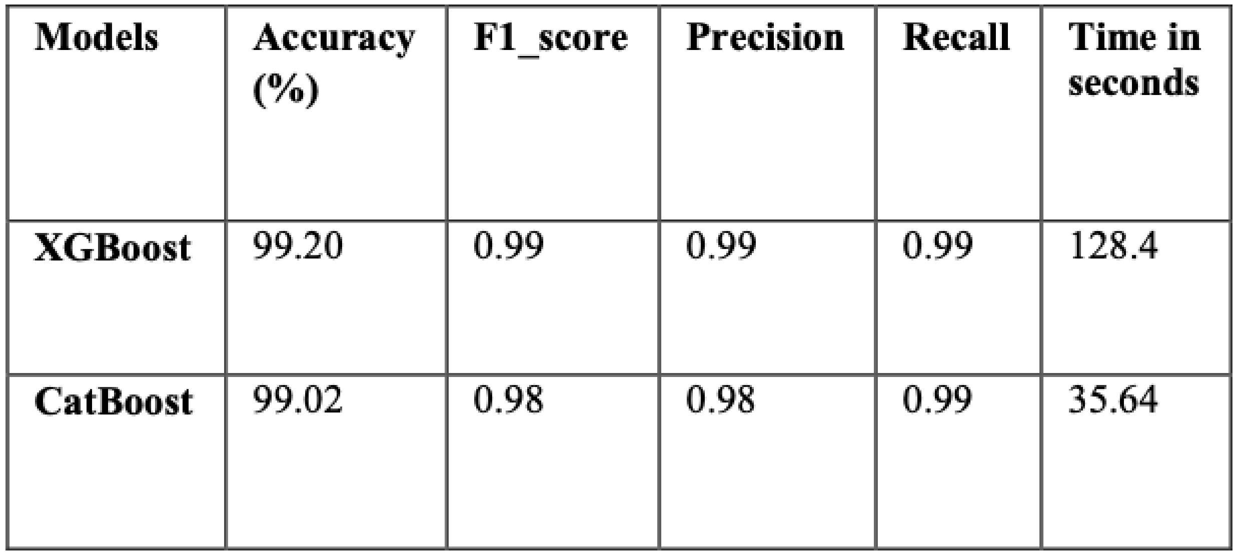


Models	Accuracy (%)	F1_score	Precision	Recall	Time in seconds
Random Forest	95.09	0.93	0.94	0.92	12.99
Gradient Boosting	96.89	0.95	0.96	0.95	97.72
XGBoost	97.65	0.97	0.97	0.96	9.00
CatBoost	97.55	0.96	0.97	0.96	5.74

- XGBoost with n_estimators = 100 have better accuracy than the rest of the models. [1]

- **Data taken from Research Paper [1]**. Malware Prediction using XGBoost and CATBoost, ISSN NO:0377-9254, 05, May/2022.

Why use XGBoost?



Models	Accuracy (%)	F1_score	Precision	Recall	Time in seconds
XGBoost	99.20	0.99	0.99	0.99	128.4
CatBoost	99.02	0.98	0.98	0.99	35.64

- XGBoost with n_estimators = 700 achieved nearly 99% accuracy. [1]

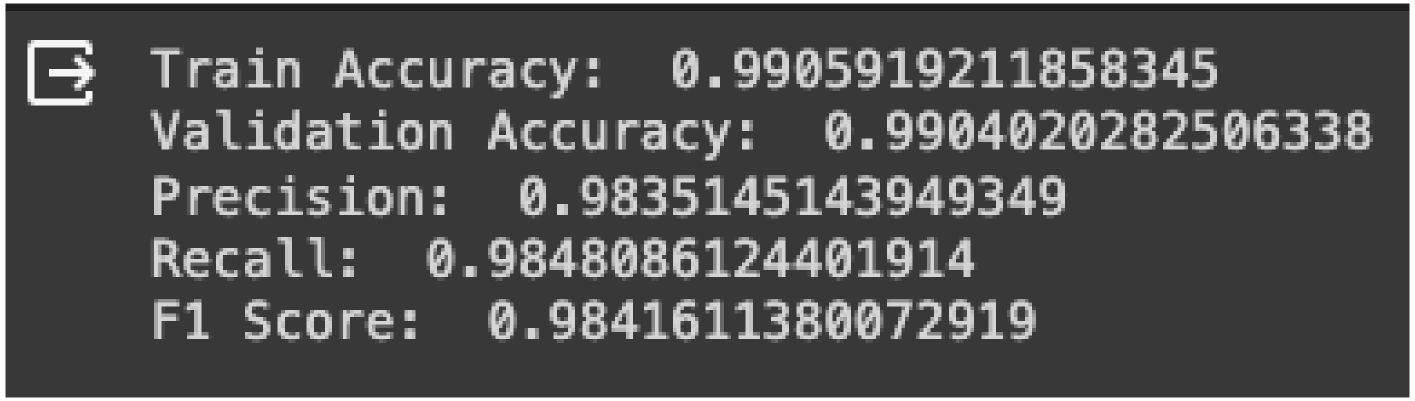
- However, increasing depth of each tree and increasing n_estimators may cause **overfitting**.

Implementation of XGBoost

- Dataset (50.67 MB) is taken from Kaggle (https://www.kaggle.com/datasets/dasarijayanth/pe-header-data)
  - This Dataset contains **Header names of Portable Executable (PE)** files as its features (columns). And the PE files used for this dataset are .exe, .dll files. PE files are most commonly used file types mainly in windows Operating System.
  - Number of instances with **label 0**: **96724**
  - Number of instances with **label 1**: **41323**
  - Shape of Dataset: **(138047, 54)**
- We train the model using XGBoost Algorithm

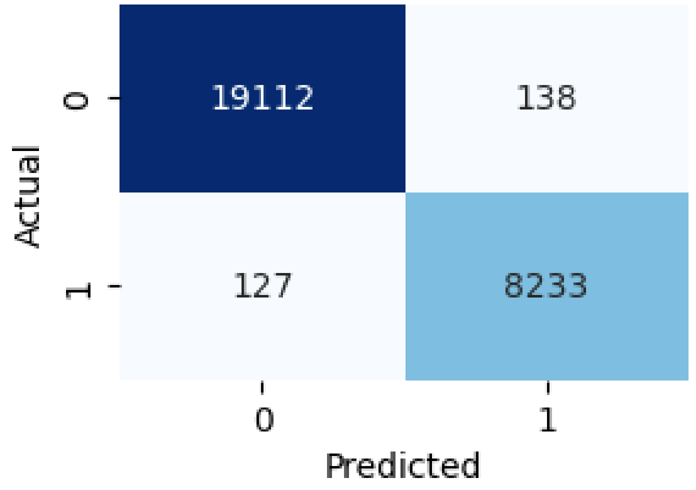
Results of XGBoost

- Max-depth: 4
- No. of Estimators: 40

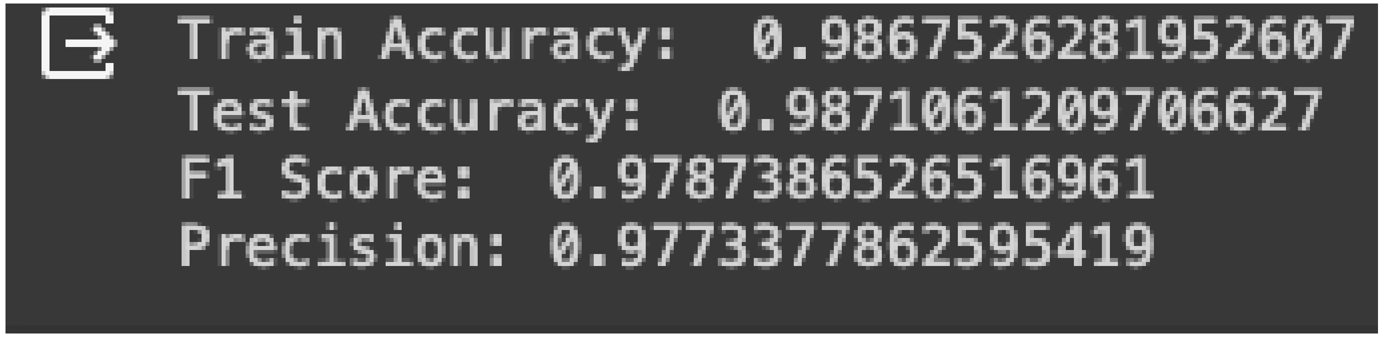


```
Train Accuracy:  0.9905919211858345
Validation Accuracy:  0.9904020282506338
Precision:  0.9835145143949349
Recall:  0.9848086124401914
F1 Score:  0.9841611380072919
```

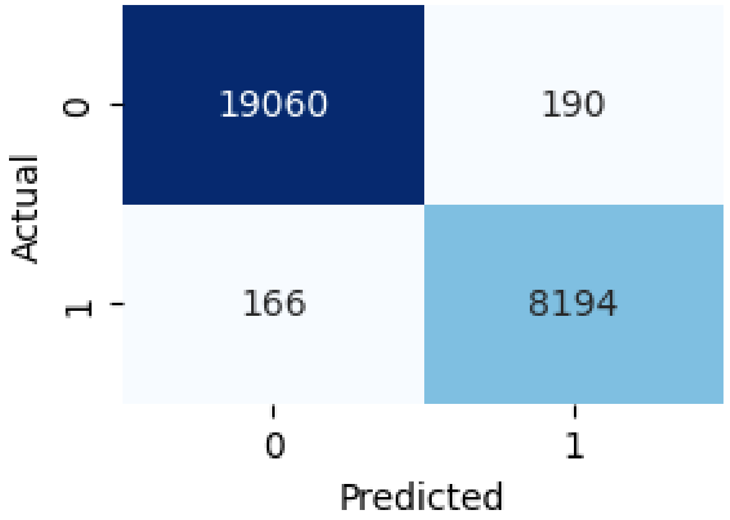
Confusion Matrix



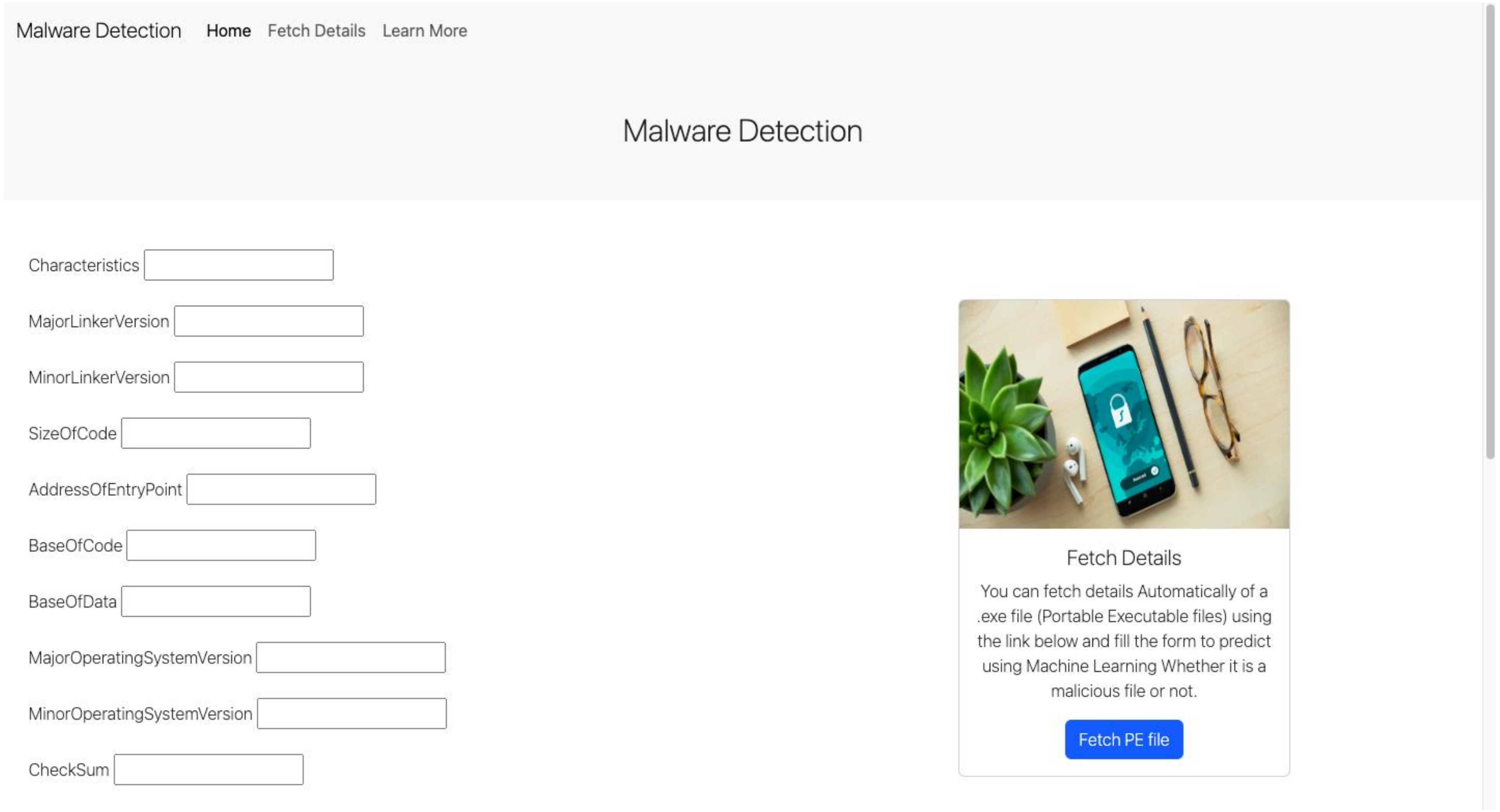
Comparison With Random Forest



```
Train Accuracy:  0.9867526281952607
Test Accuracy:  0.987106120970627
F1 Score:  0.9787386526516961
Precision: 0.9773377862595419
```



Practical Implementation of Malware Detection



Malware Detection   Home   Fetch Details   Learn More

Malware Detection

Characteristics	
MajorLinkerVersion	
MinorLinkerVersion	
SizeOfCode	
AddressOfEntryPoint	
BaseOfCode	
BaseOfData	
MajorOperatingSystemVersion	
MinorOperatingSystemVersion	
CheckSum	

Fetch Details

You can fetch details Automatically of a .exe file (Portable Executable files) using the link below and fill the form to predict using Machine Learning Whether it is a malicious file or not.

Aim: Malware Detection

Tools: Flask, Colab

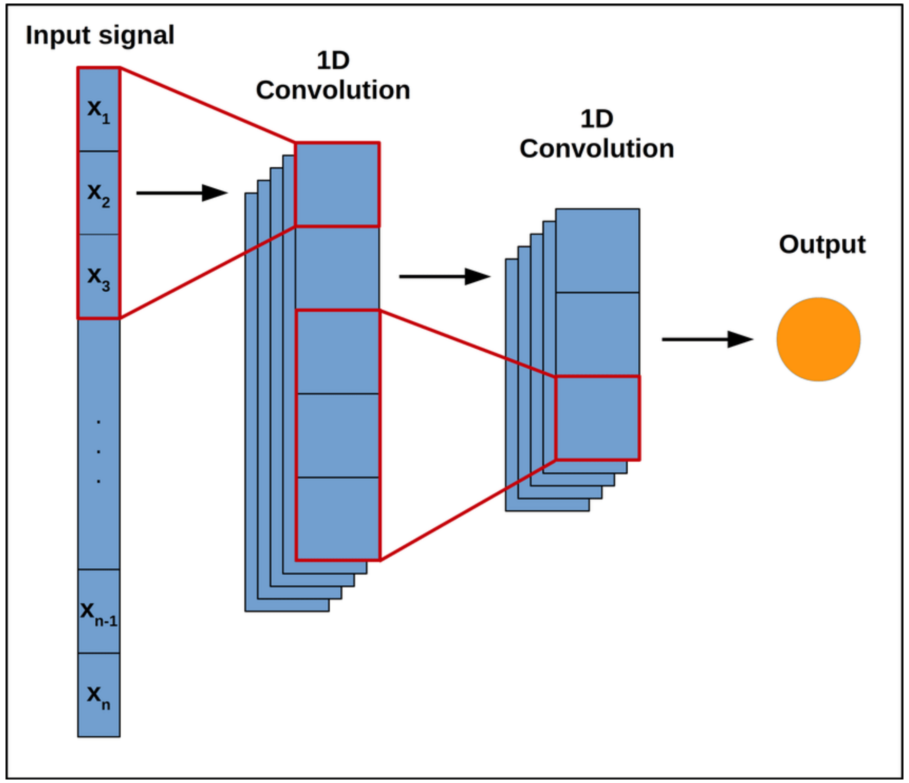
Algorithm: XGBoost

Programming Languages:

Python, HTML, CSS

1D-CNN

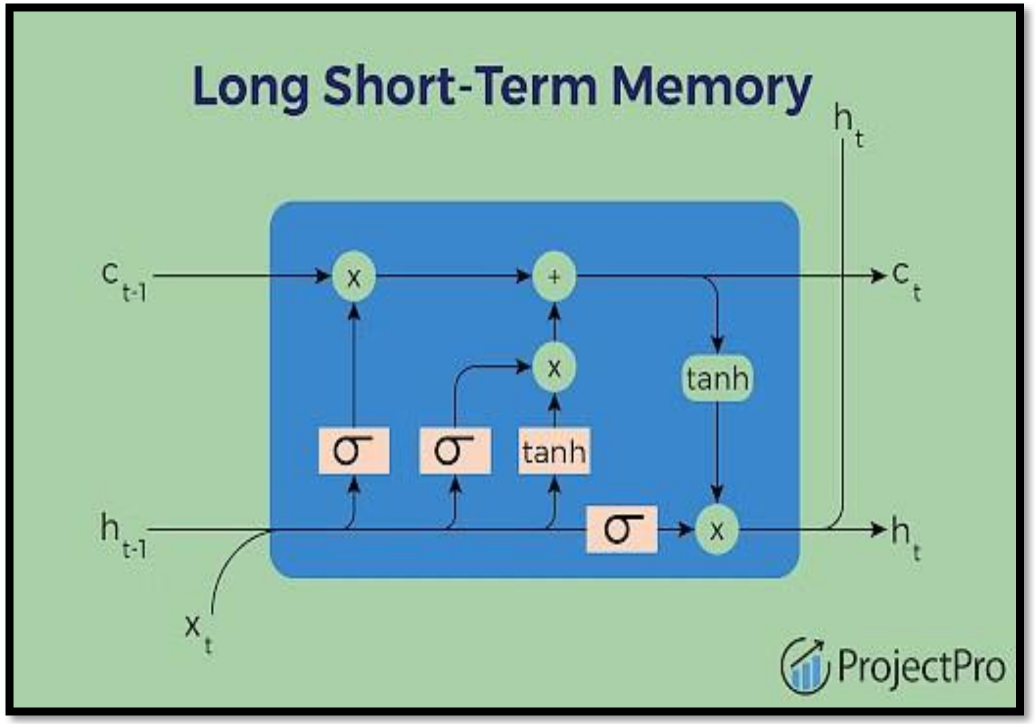
- 1D CNNs analyse **sequential data** like time series or text.
- They capture **local patterns** within the sequence, similar to how images have spatial patterns.
- Benefits include **efficient feature extraction** and high speed.



Source [11] : 1D-CNN

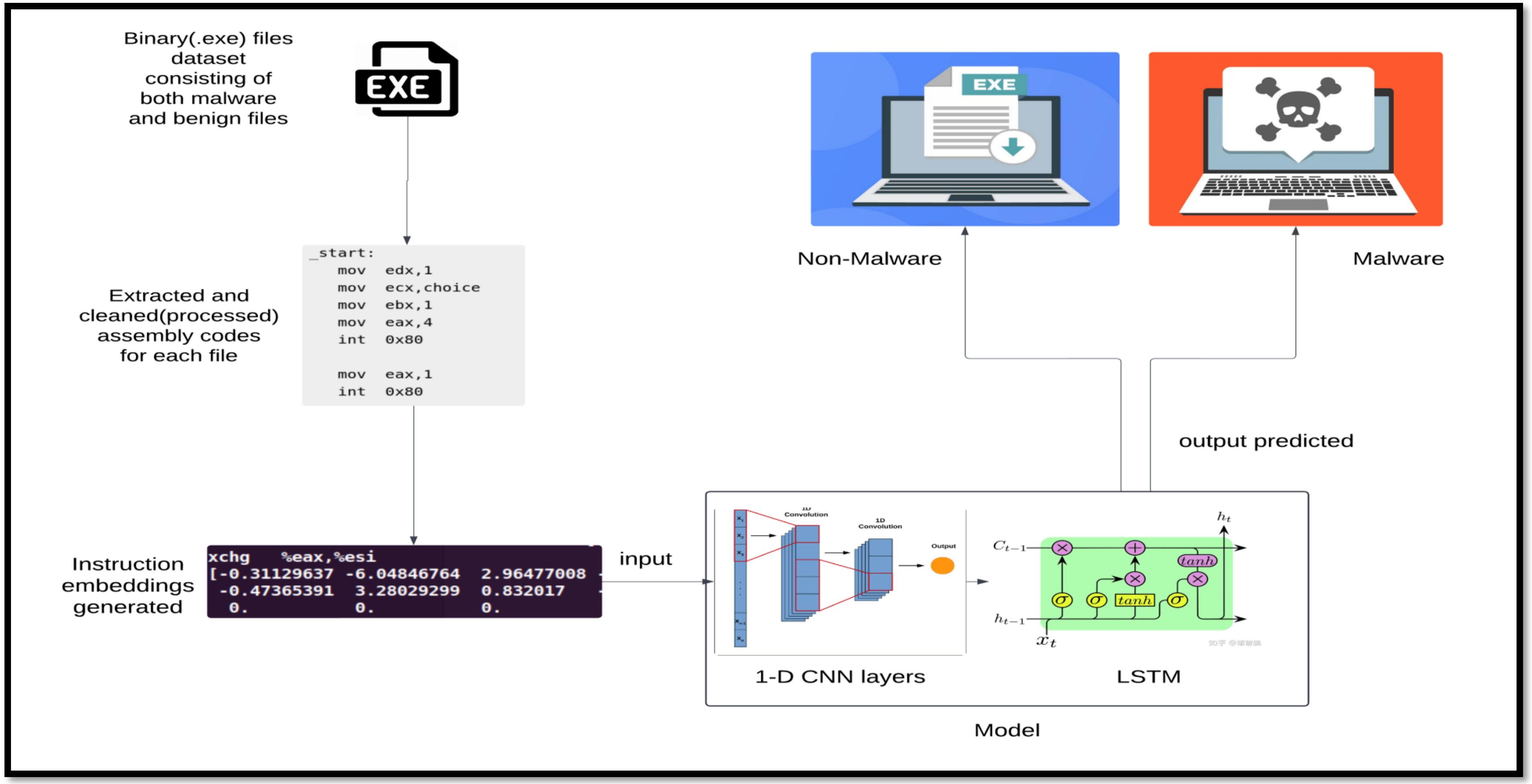
LSTM

- **Long Short-Term Memory (LSTM)** networks address the limitation of traditional neural networks: forgetting long-range dependencies in sequential data.
- Unlike their simpler counterparts, LSTMs incorporate **memory cells** that **retain relevant information** for extended periods, enabling them to process and **analyze sequences effectively**.



LSTM (Source: ProjectPro)

Proposed Architecture



Binary(.exe) files dataset consisting of both malware and benign files
EXE
Extracted and cleaned(processed) assembly codes for each file
Instruction embeddings generated
input
1-D CNN layers
LSTM
Model
output predicted
Non-Malware
Malware

```
_start:
    mov   edx,1
    mov   ecx,choice
    mov   ebx,1
    mov   eax,4
    int   0x80

    mov   eax,1
    int   0x80
```

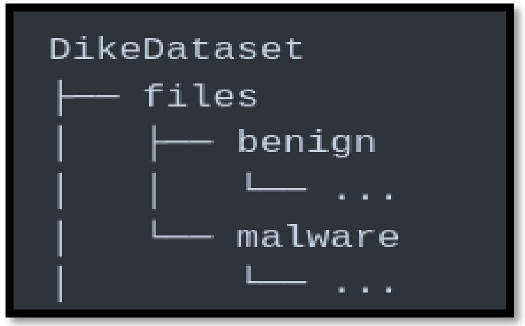
```
xchg    %eax,%esi
[-0.31129637 -6.04846764   2.96477008
 -0.47365391  3.28029299   0.832017
  0.           0.           0.
```

Dataset

- Dike Dataset
- Is a labeled dataset containing **benign and malicious binary(exe) files.**
- Malware files downloaded from MalwareBazaar.abuse.ch
- Benign files downloaded from softonic ,sourceforge and github

Malware	Benign
1949	1903

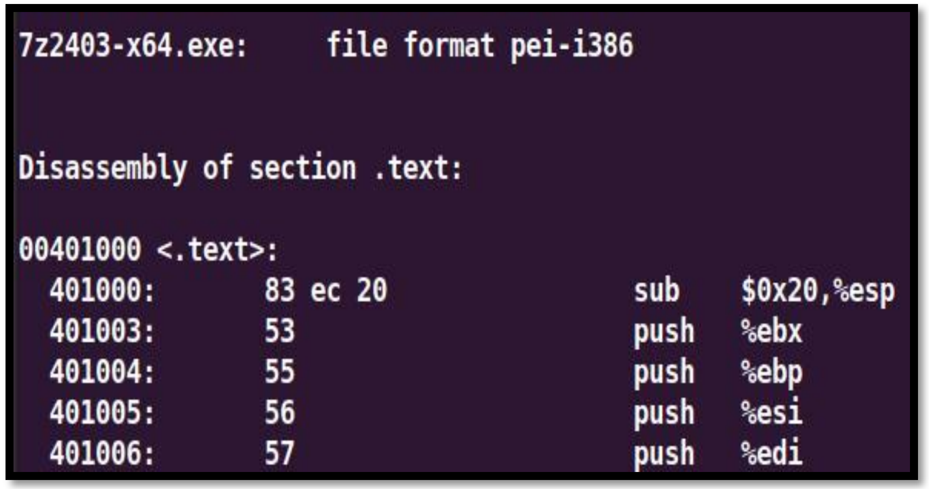
Dataset Distribution



```
DikeDataset
├── files
|   ├── benign
|   |   └── ...
|   └── malware
|       └── ...
```

Folder Structure

Dataset Cleaning (Preprocessing)



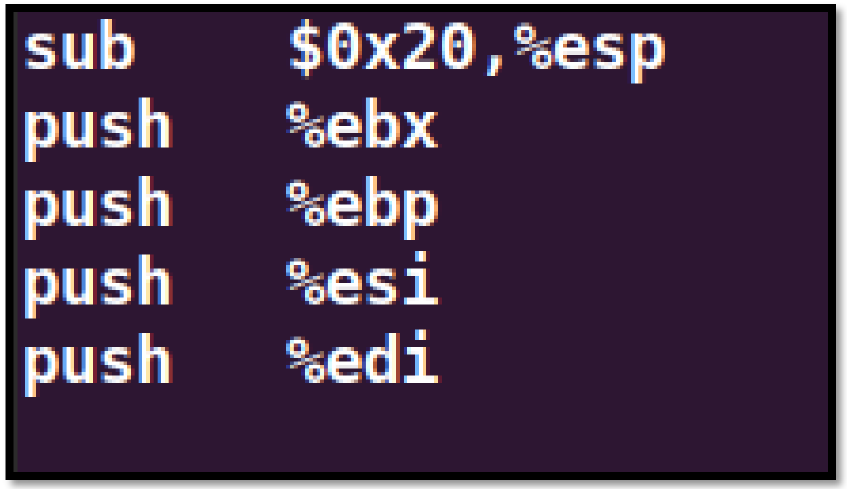
```
7z2403-x64.exe:     file format pei-i386


Disassembly of section .text:

00401000 <.text>:
  401000:       83 ec 20                sub    $0x20,%esp
  401003:       53                      push   %ebx
  401004:       55                      push   %ebp
  401005:       56                      push   %esi
  401006:       57                      push   %edi
```

Raw assembly code generated using "objdump" command in linux

$\longrightarrow$

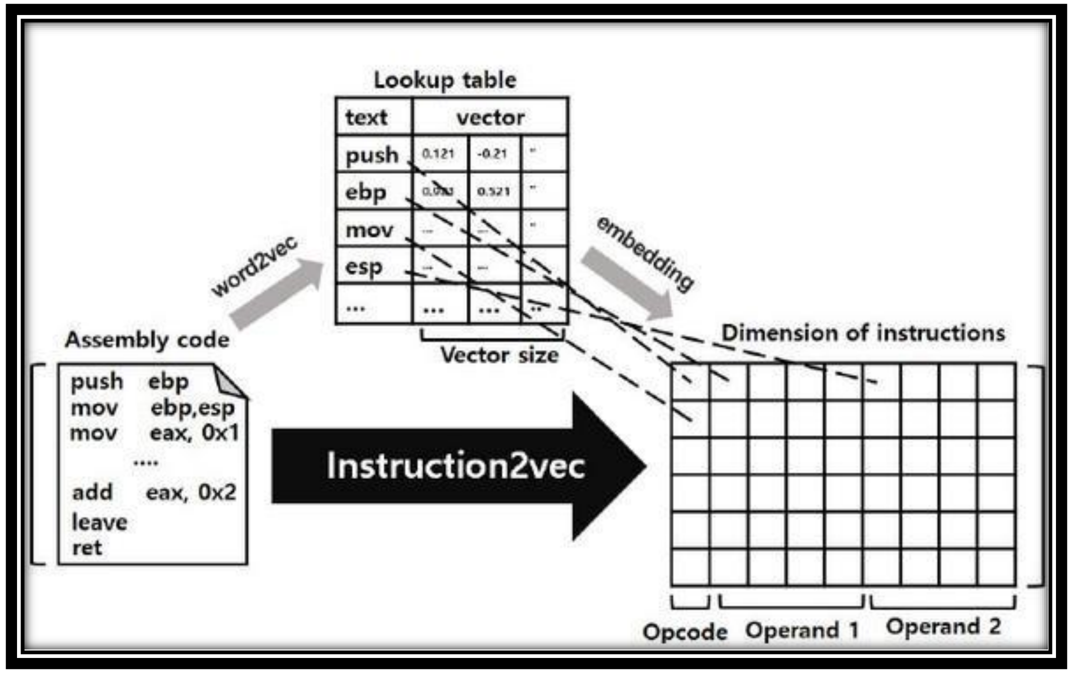


```
sub     $0x20,%esp
push    %ebx
push    %ebp
push    %esi
push    %edi
```

Removed memory addresses and opcodes to get the assembly instructions only

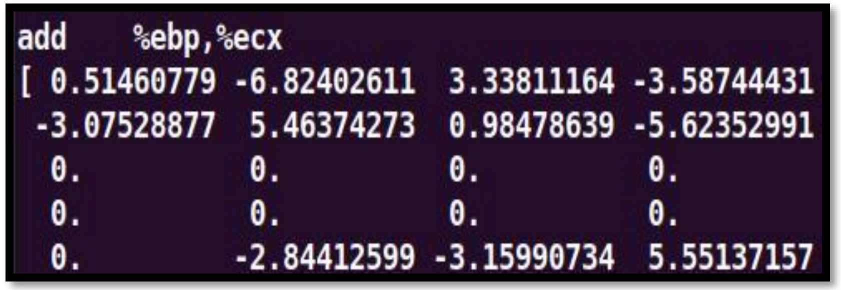
Instruction Embeddings- Instruction2Vec

- Generated using **Instruction2Vec**(based on Word2Vec), **specialized embedding generation technique** for assembly instructions.
- **Word2Vec alone doesn't consider the syntax** of assembly language. **Instruction2Vec outperforms both standard Word2Vec and Binary2Img** on generating accurate instruction embeddings. [4]
- Separates instructions, registers and memory addresses(pointers)
- **Rearranges the sequence of instructions** to match the execution order in the output embeddings file for an input assembly file.
- **Handles function calls** and **branches** by placing the actual function code in the middle of the sequence so that the meaning of the program is not lost.



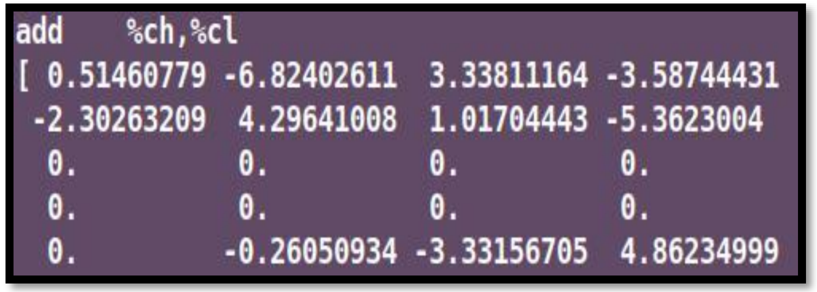
Source:[4]

Instruction2Vec embedding Example



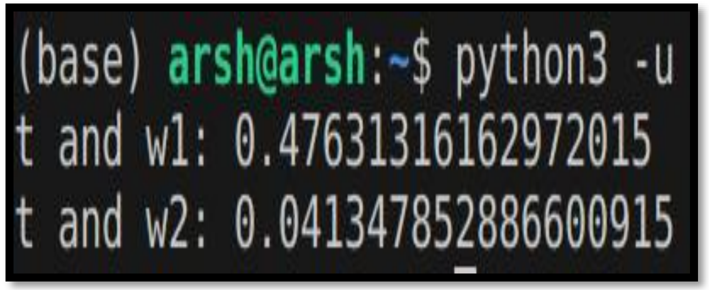
```
add    %ebp,%ecx
[ 0.51460779 -6.82402611  3.33811164 -3.58744431
 -3.07528877  5.46374273  0.98478639 -5.62352991
  0.          0.          0.          0.
  0.          0.          0.          0.
  0.         -2.84412599 -3.15990734  5.55137157
```

Example similar instruction "w1"



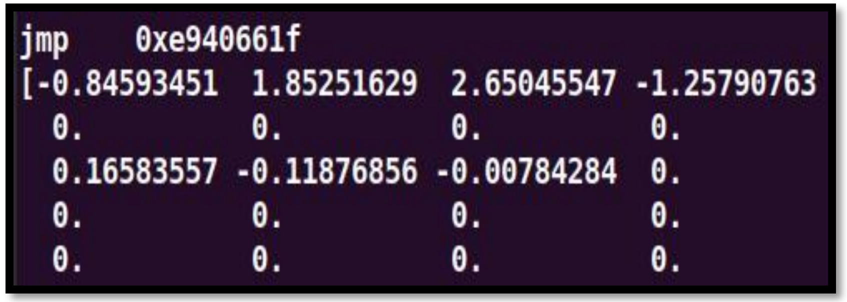
```
add    %ch,%cl
[ 0.51460779 -6.82402611  3.33811164 -3.58744431
 -2.30263209  4.29641008  1.01704443 -5.3623004
  0.          0.          0.          0.
  0.          0.          0.          0.
  0.         -0.26050934 -3.33156705  4.86234999
```

Example target instruction "t"



```
(base) arsh@arsh:~$ python3 -u
t and w1: 0.47631316162972015
t and w2: 0.041347852886600915
```

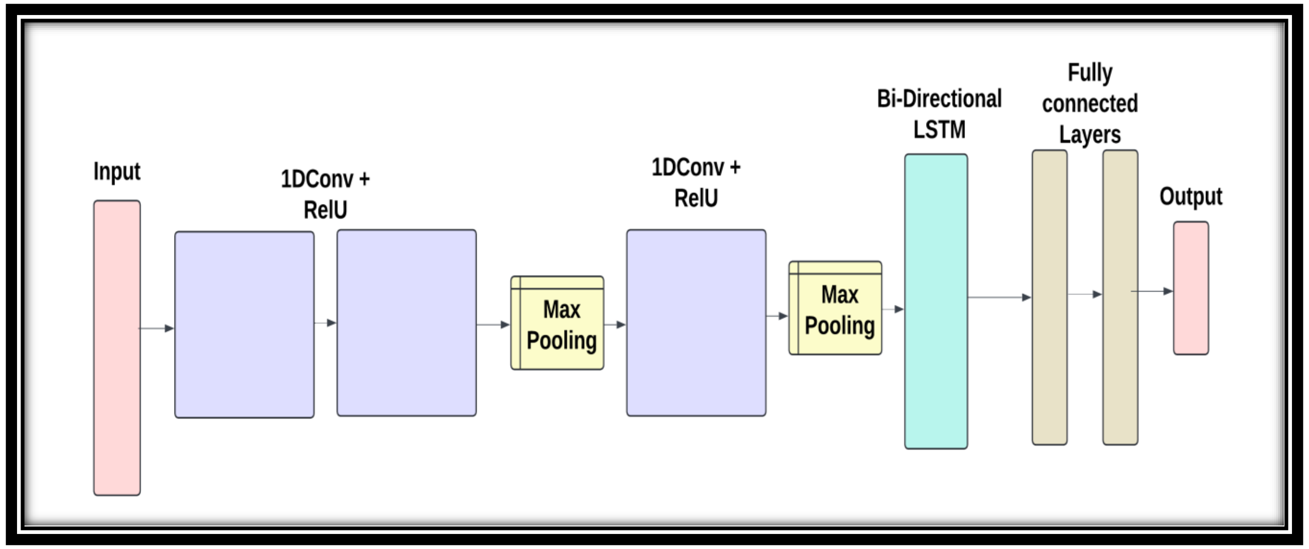
Cosine similarities



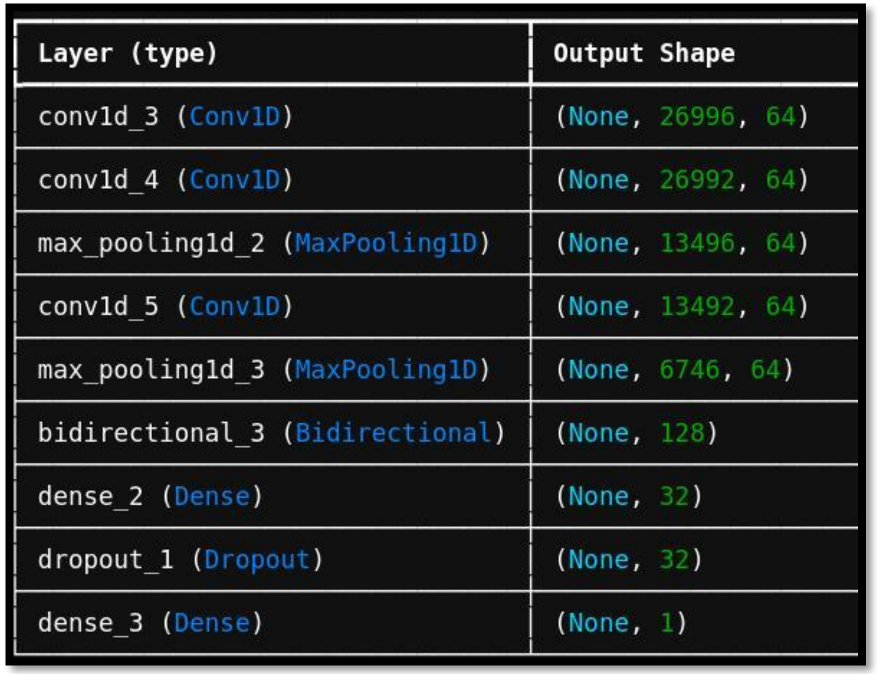
```
jmp    0xe940661f
[-0.84593451  1.85251629  2.65045547 -1.25790763
  0.          0.          0.          0.
  0.16583557 -0.11876856 -0.00784284  0.
  0.          0.          0.          0.
  0.          0.          0.          0.
```

Example non-similar instruction "w2"

Model Architecture



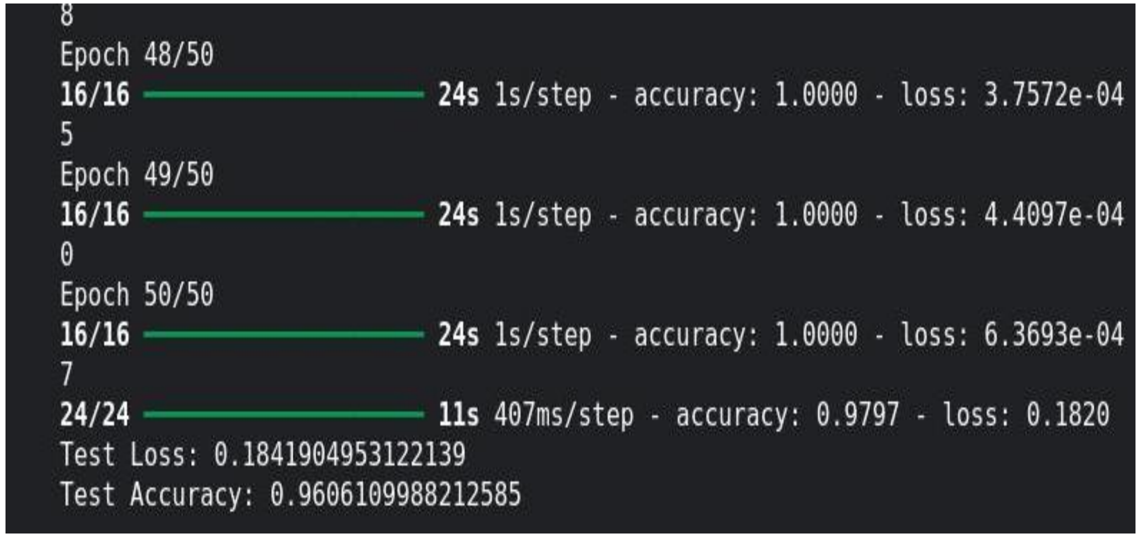
Reference: [2] SP.W., Hwang, YS. (2021). Malware Detection by Merging 1D CNN and Bi-directional LSTM .Springer 10.1007/978-981-33-6385-4_16



```
Layer (type)                          Output Shape
conv1d_3 (Conv1D)                     (None, 26996, 64)
conv1d_4 (Conv1D)                     (None, 26992, 64)
max_pooling1d_2 (MaxPooling1D)        (None, 13496, 64)
conv1d_5 (Conv1D)                     (None, 13492, 64)
max_pooling1d_3 (MaxPooling1D)        (None, 6746, 64)
bidirectional_3 (Bidirectional)       (None, 128)
dense_2 (Dense)                       (None, 32)
dropout_1 (Dropout)                   (None, 32)
dense_3 (Dense)                       (None, 1)
```

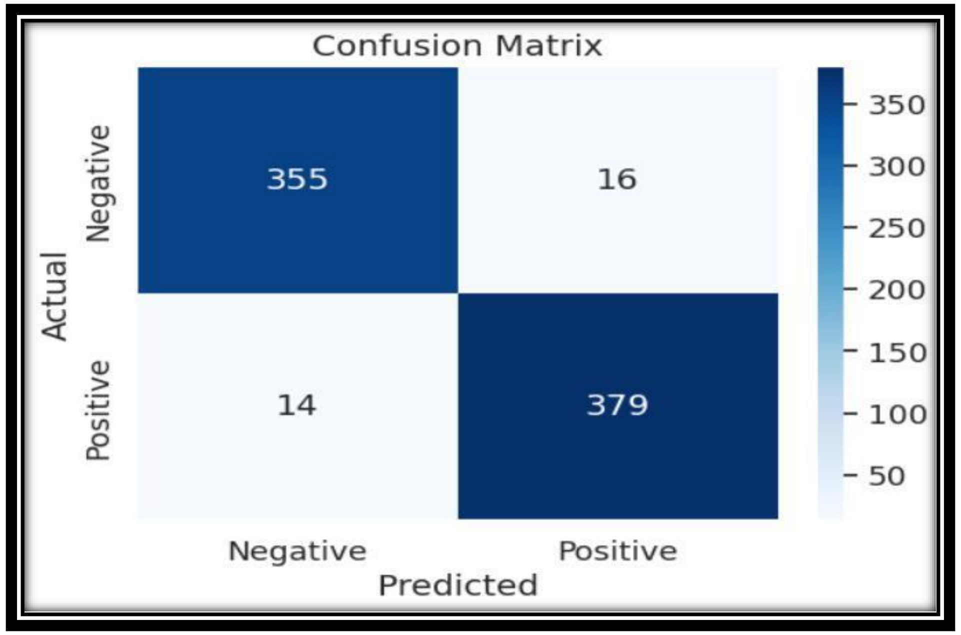
Model.Summary()

Result



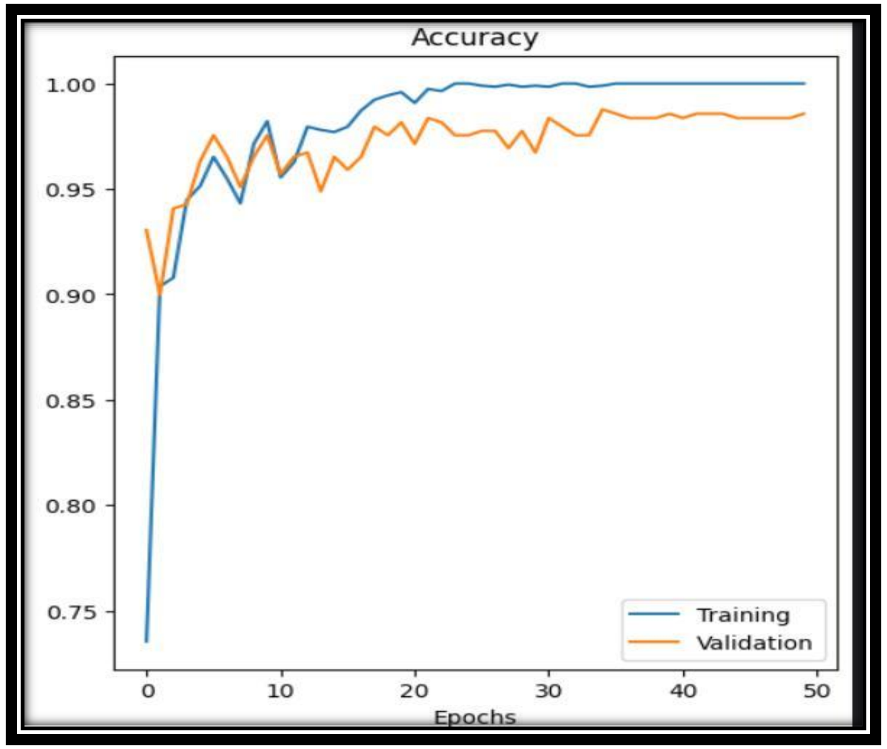
```
8
Epoch 48/50
16/16 ━━━━━━━━━━━━━━━━━━━━ 24s 1s/step - accuracy: 1.0000 - loss: 3.7572e-04
5
Epoch 49/50
16/16 ━━━━━━━━━━━━━━━━━━━━ 24s 1s/step - accuracy: 1.0000 - loss: 4.4097e-04
0
Epoch 50/50
16/16 ━━━━━━━━━━━━━━━━━━━━ 24s 1s/step - accuracy: 1.0000 - loss: 6.3693e-04
7
24/24 ━━━━━━━━━━━━━━━━━━━━ 11s 407ms/step - accuracy: 0.9797 - loss: 0.1820
Test Loss: 0.1841904953122139
Test Accuracy: 0.9606109988212585
```

~96% testing accuracy



Confusion matrix

Results & Comparison



Training and validation accuracy vs number of epochs

Model Type	Accuracy
1d-cnn	95.8%
LSTM	96.95%
1dcnn-lstm(wd2vec,dex)	94.7%
1dcnn-lstm(inst2vec,asm)	96.06%

Reference:[1],[2],[3]

CONCLUSION & DISCUSSION

- In conclusion, the growing demand for internet connectivity has heightened the risk of cyber- attacks, emphasizing the urgent need for effective malware detection. Traditional methods fall short, leading to the exploration of advanced solutions.
- This report outlines a machine learning- based approach focusing on XGBoost, 1-CNN & LSTM algorithms to detect malware in Windows executable files, which are new approaches.
- The chosen algorithms were strategically selected based on their individual merits, ranging from robustness and feature importance to instance-based learning and efficiency.
- Future scope of project can be to add new virus data tested on the site in the database so it can be used for future detections.

References

1) Malware Detection with LSTM using Opcode Language: Renjie Lu: arXiv:1906.04593v1

2) SP.W., Hwang, YS. (2021). Malware Detection by Merging 1D CNN and Bi-directional LSTM Utilizing Sequential Data .Springer 10.1007/978-981-33-6385-4_16

3) Malware Detection Using 1-Dimensional Convolutional Neural Networks:10.1109/EuroSPW.2019.00034

4) Lee, & Kwon, Hyun & Choi, Sang-Hoon & Lim, Seung-Ho & Baek, Sung Hoon & Park(2019). Instruction2vec: Efficient Preprocessor of Assembly Code 9. 4086. 10.3390/app9194086. Applied Sciences. 9. 4086. 10.3390

5) Malware Prediction using XGBoost and CATBoost, ISSN NO:0377-9254, Journal of Engineering Sciences, 05, May/2022

6) Static Malware Detection using Deep Neural Networks on Portable Executables, August 2019

7) Introduction to Malware and Malware Analysis: A brief overview, ISSN: 2321-7782, Research Gate, October 2016

8) A Study on Malware and Malware Detection Techniques, DOI: 10.5815, MECS, 08 March 2018

9) Malware and Malware Detection Techniques: A Survey, ISSN: 2278-0181, International Journal of Engineering Research and Technology, 12-December – 2013

10) A CNN and Image-Based Approach for Malware Analysis::10.1109/ETCEA57049.2022.10009748, 12 January 2023 *International Conference on Emerging Trends in Computing and Engineering Applications (ETCEA),*

11) Wind Forecasting using CNN and Bidirectional LSTM, ISSN No. 17445302.223.2218323, 2023, Offshore Structures. 1-9. 10.1080/17445302.2023.2218323.