A Framework for Enabling Static Detection and Classification of PE Malware

Zain Nawaz
01-247212-017
Dr. Faisal Bashir

A thesis submitted in fulfilment of the requirements for the award of degree of Masters of Science (Information Security)

Department of Computer Science

BAHRIA UNIVERSITY ISLAMABAD

SEPTEMBER 2023

Approval of Examination

Scholar Name: **Zain Nawaz**
Registration Number: **75933**
Enrollment: **01-247212-017**
Program of Study: **MS (IS)**
Thesis Title: **A Framework for Enabling Static Detection and Classification of PE Malware**

It is to certify that the above scholar's thesis has been completed to my satisfaction and, to my belief, its standard is appropriate for submission for examination. I have also conducted plagiarism test of this thesis using HEC prescribed software and found similarity index **10%** that is within the permissible limit set by the HEC for the MS degree thesis. I have also found the thesis in a format recognized by the BU for the MS thesis.

Principal Supervisor Name: **Dr. Faisal Bashir**
Principal Supervisor Signature:

Date: **27-Oct-2023**

Author's Declaration

I, **Zain Nawaz** solemnly affirm that my MS/M.Phil thesis titled "A Framework for Enabling Static Detection and Classification of PE Malware" is the product of my own work and has not been previously submitted by me to obtain any degree from Bahria University or any other academic institution, whether within the country or internationally. I understand that if my statement is determined to be false or incorrect at any point, even after my graduation, Bahria University reserves the right to revoke or cancel my MS/M.Phil degree.

Name of Scholar: **Zain Nawaz**

Date: **27-Oct-2023**

Plagiarism Undertaking

I, solemnly declare that research work presented in the thesis titled **A Framework for Enabling Static Detection and Classification of PE Malware** is solely my research work with no significant contribution from any other person. Small contribution / help wherever taken has been duly acknowledged and that complete thesis has been written by me. I understand the zero tolerance policy of the HEC and Bahria University towards plagiarism. Therefore I as an Author of the above titled thesis declare that no portion of my thesis has been plagiarized and any material used as reference is properly referred / cited.

I undertake that if I am found guilty of any formal plagiarism in the above titled thesis even after award of MS degree, the university reserves the right to withdraw / revoke my MS degree and that HEC and the University has the right to publish my name on the HEC / University website on which names of scholars are placed who submitted plagiarized thesis.

Name of Scholar: **Zain Nawaz**

Date: **27-Oct-2023**

Dedication

I dedicate this diligent effort to my beloved parents, teachers, friends, and family members who have consistently been a source of inspiration, unwavering support, and innovative ideas throughout my journey. My gratitude extends to all the advertisers and service providers involved in the field of malware detection and classification, as they play a vital role in this realm. With deep respect, I offer this work to all the researchers and knowledge seekers, hoping that it contributes to future explorations in this field. The mechanism described in this document has been developed and implemented internally at CRC-BU and is poised to function as an active service provider upon the completion of this project.

Zain Nawaz

Acknowledgements

In the process of preparing this thesis, I had the privilege of engaging with numerous individuals, including researchers, academicians, and practitioners, whose insights and contributions greatly enriched my understanding and ideas. I would like to extend my heartfelt gratitude, in particular, to my primary thesis supervisor, Professor Dr. Faisal Bashir, for their unwavering support, guidance, constructive criticism, and friendship. Without their continued mentorship and genuine interest in my work, this thesis would not have reached its current form.

I also wish to express my sincere thanks to the dedicated librarians at Bahria University, whose assistance in providing relevant literature was invaluable to my research. I am indebted to my fellow postgraduate students for their camaraderie and support throughout this journey. I extend my appreciation to all my colleagues and others who offered their help and insights on various occasions. Their perspectives and advice have been truly beneficial, even if I can't name them all within the constraints of this space.

Lastly, but certainly not least, I am deeply grateful to my family members for their unwavering encouragement and support.

Abstract

Over the preceding years, the proliferation of malware on PC platforms, particularly on Windows OS, has become notably more severe. To counter the propagation of numerous malware variations, the implementation of ML classifiers for identifying malicious PE files has been suggested, aiming for autonomous categorization. Recent advancements in computer systems have transitioned human experiences from the physical to virtual realms, a shift that has been accelerated by the Covid-19 pandemic. Similarly, the interest of cybercriminals has pivoted from real-world to virtual environments. This transition is driven by the greater ease of committing cybercrimes in the digital realm compared to the physical world. Cyber attackers often utilize malicious software (malware) to execute cyber assaults. The evolution of malware variants continues through the utilization of sophisticated obfuscation and packing techniques. Conventional artificial intelligence (AI), particularly traditional ML algorithms, struggle to effectively identify novel and intricate malware variants. Embracing a distinct paradigm from traditional ML algorithms, the deep learning (DL) approach offers a promising avenue to address the challenge of detecting diverse malware variants.

This study introduces an innovative deep learning architecture (LSTM) designed to categorize malware variations based on features extracted from function call graphs (FCGs). A particularly demanding task involves the selection of pertinent features from extensive datasets, ensuring that the classification model can be constructed with enhanced efficiency and accuracy. This research serves a dual purpose: first, to conduct a comprehensive overview of prevailing classification and detection methodologies, secondly, to devise an automated system for the detection and categorization of malicious Portable Executable files. This classification relies on function call graphs, emphasizing efficiency without sacrificing accuracy. Additionally, an aspiration of this study is to extend its scope to encompass the classification of malware families through the utilization of a deep learning model.

TABLE OF CONTENTS

LIST OF TABLE

LIST OF FIGURE

Acronyms & Abbreviations

LSTM	Long Short Term Memory
FCG	Function Call Graph
CFG	Control Flow Graph
GEMAL	Graph Embedding Network
IDA	Interactive Disassembler
GBDT	Gradient Boosting Decision Tree
GIN	Graph Isomorphism Network
RETDEC	Retargetable Decompiler

CHAPTER 1

INTRODUCTION

Every day, data security companies receive tens of thousands of different malicious executables for analysis. Automated identification, verification, and classification systems are required to handle these massive amounts of samples in a timely manner. In practice, however, code obfuscation techniques such as packed or encrypted executable code make automated identification of malware difficult. Furthermore, cybercriminals are continually developing new versions of their harmful software in order to avoid pattern-based detection by antivirus programs. Due to the recent spike in the number of cybercrime and security breaches throughout the world. According to an AV-TEST statistical study [1], approximately 30 million malicious Portable Executable files were registered in the first quater of 2019 [2]. Malware detection and categorization are typically performed separately on each malicious PE file. To begin, if an executable application has dangerous content, it must be detected using malware analysis tools.

Malware is a programme that is designed to interfere with normal system processes, acquire crucial data, or authorize access to confidential systems. Malware can take the shape of coding, scripts, active material, and so on. The term 'malware' refers to a wide range of hostile or intrusive software. Malware are problematic because they impose too many restrictions on disordered PCs, such as disabling malware detectors or AV scanners that have been installed for security purposes. As we all know, malware is classified into three generations: payload, enabling vulnerability, and transmission method. The features of viruses that are replicated or propagated by some human actions, such as emails and file sharing, are shared by first generation malware. In the Second Generation malware, the traits of worms are hybrid in nature, involving some features of virus and Trojans that are not duplicated by human acts. As a result, Third Generation malware's are geographically restricted or organization-specific. Malware of this type typically targets security technologies and goods.Following malware detection, A malware-classification tech-

nique places the executable programme in the most closely related family for further analysis.
Focuses on both malware detection and categorization [3]. Two techniques are used to detect malware's [4]. Behavior-based techniques: Primary goal to recognize the behavior of known or benign malware. The parameters of the behavior-based technique include a number of criteria such as source/destination address, types of attachments and other measurable statistical characteristics of the malware. Signature-based techniques: This category includes most antivirus detection methods. These signatures are generated by inspecting the disassembled code. There are several disassemblers and debuggers to help you disassemble portable executable. The result is parsed code and function loading. As a result, these properties serve a crucial role in creation of a malware family's signature. These approaches been utilized to analyze malicious PE files to extract characteristics useful for malware detection and categorization. Static analysis often extracts content-driven traits such as API sequences, opcodes instruction[5], and FCG [6, 7] from disassembled PE files as the inceptive features for analysis. Static analysis easily collects structural and semantic knowledge for detailed study, but it is prone to code concealment strategies like as reduction and polymorphic/metamorphic transformation [8].
In most cases, dynamic analysis places malware samples in a simulated setting that is then analysed using a debugging tool to determine their behaviour such as network activity, system calls[9], file actions, and registry change records. Code obfuscation strategies have less of an impact on dynamic analysis, although virus execution takes far more time and resources than static analysis. The number of cybercrime and security breaches has recently increased worldwide. This usually happens after naive people are tricked into installing malware on their devices. To avoid this, a static analysis of Windows executables is used to categorize them as malware or goodware. In static malware analysis, the executable to be extracted is provided with static parameters, such as function imports and API calls from the executable's PE header file, and these inputs are fed to a trained mathematical model that ultimately categorizes the file as good-ware/malware.

1.1 Malware Analysis

Malicious software that infiltrates computers, servers, hosts or networks is commonly referred to as malware. It encompasses a range of software designed to exploit programmable systems, networks or services. There are risks associated with malware including infections, worms, adware, spyware, Trojan infections and ransomware.

Figure 1.1: Types of Malware

Malware analysis involves comprehending the actions and intent of a questionable file or URL. The insights gained from this examination contribute to identifying and lessening potential risks. The primary advantage of engaging in malware analysis lies in its support for incident responders and security analysts. Malware analysis have been performed using these techniques that are shown in Fig 1.2

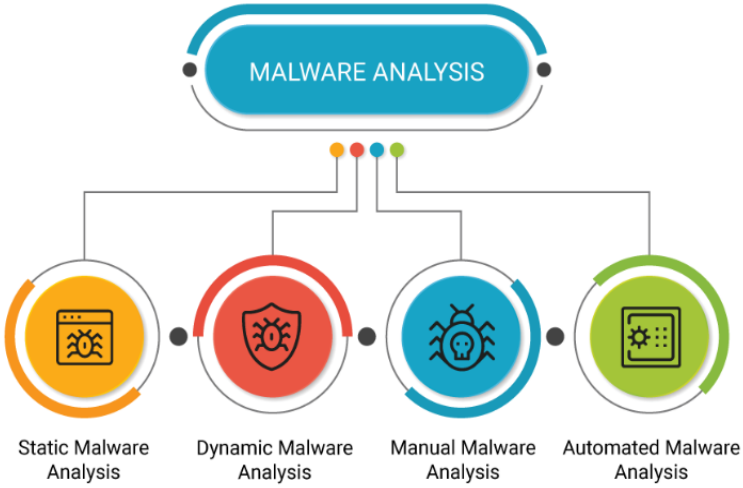

Figure 1.2: Malware Analysis Approaches

1.2 Types

Two types of malware analysis will be discussed. The first method is Static Analysis 1.3.1, second is the Dynamic analysis 1.3.2 [10].

1.2.1 Static Analysis

Static malware analysis involves detecting potential malicious intent within files. This analysis focuses on the former, looking at static attributes such as headers, metadata and integrated assets, etc. Doesn't require an actively functioning malware program; a basic static evaluation suffices. This technique is

beneficial for uncovering malevolent libraries, bundled files, or infrastructure. Technical indicators like hashes, file names, strings such as IP addresses, file header data, and domains are pinpointed during the malware investigation. Through various tools like network analyzers and disassemblers, the malware can be examined without actually executing it. These tools accumulate insights into the malware's operation. However, it's worth noting that certain advanced malware might display perilous run time behaviors that static malware analysis, due to its non-execution nature, might fail to detect.

Static malware analysis is examining a malware sample without running it, obviating the requirement of an analyst at any stage of the procedure. This method scrutinizes the sample's behavior to determine its capabilities and the extent of its potential harm to the system. Static analysis encompasses the process of conducting signature analysis on a malware exe file. This exe file carries distinctive identifier and dissected in reverse using disassembler such as IDA, which converts the machine code into assembly language code. Within this approach to malware analysis, several techniques come into play, including virus scanning, detection of packers, file fingerprinting, debugging, and memory dumping. Fig 1.3 shows how static analysis have been performed for detection.

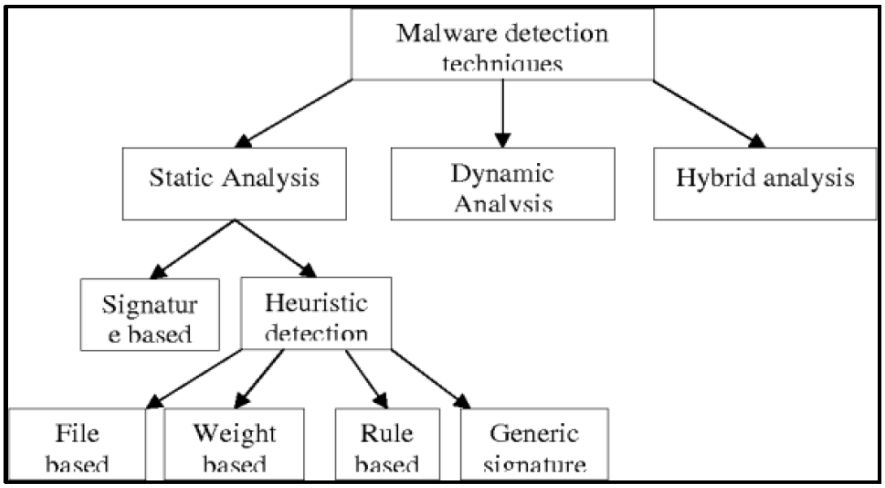

Figure 1.3: Static Analysis

Static analysis predominantly adopts a signature-driven methodology for both malware identification and analysis. The specific identifier within malware manifests as a sequence of bytes. These signatures are examined based on diverse patterns. However, antivirus programs that rely on signature-based detection prove effective only against commonly encountered malware. They exhibit limitations in tackling more intricate and sophisticated forms of mal-

ware. This is where dynamic malware analysis takes center stage.

1.2.2 Dynamic Analysis

A sandbox is an environment used in dynamic malware analysis, where harmful code is executed. It allows security professionals to closely monitor the behavior of viruses without worrying about infecting their machines or networks. Furthermore, autonomous the sandbox saves time that would otherwise be spent on reverse engineering files to uncover code. However dynamic analysis can be challenging when facing adversaries who anticipate the use of sandboxes. These adversaries hide their code. Make it inactive until specific conditions are met as a way to deceive.

Dynamic analysis enables you to observe the actions of malware as they occur within a controlled environment. The use of machines (VMs) is crucial, during analysis since there is a potential risk of the malware causing irreversible damage to the host environment. Throughout the process we need to focus on cues such, as how the malware interacts with network traffic its actions concerning the file system and any changes made to the registry. Network setup for the VM could look like this in below Fig 1.4

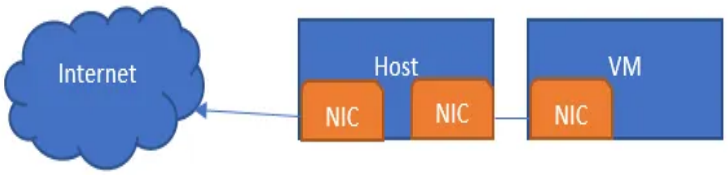

Figure 1.4: Isolated VM Network for Dynamic Analysis

A signature-based methodology is not used in the dynamic analysis. Instead, it makes this determination based on the behavior of the infection. It entails looking at what the malware is doing.

1.3 Problem Motivation

The ever-evolving landscape of digital threats presents a formidable challenge for cybersecurity experts and organizations worldwide. Among these threats, malware stands out as a persistent and ever-adapting adversary. Detecting and countering malware has become a critical imperative in the realm of cybersecurity, but it remains an intricate and multifaceted problem.

Malware, by its very nature, is designed to be elusive and complex. This complexity makes it a formidable foe, and traditional approaches to detection often fall short. As noted in a previous study [1], one of the primary difficulties

lies in the execution-based dynamic analysis of malware. While dynamic analysis is a valuable tool for understanding the behavior of malicious software, it comes at a significant cost in terms of resources and implementation. The need to execute potentially harmful code in a controlled environment introduces inherent risks and resource-intensive requirements.

On the other hand, static analysis [11], which relies on the examination of malware signatures and executable file attributes, offers an alternative approach. However, this method has faced increasing challenges in keeping up with the ever-changing landscape of malware. Malicious actors continuously modify and obfuscate their code, rendering traditional static analysis techniques less effective. This inherent adaptability of malware necessitates a reevaluation of our approaches to cybersecurity.

An integral aspect of combating malware is the classification of malware families. Surprisingly, this area has received limited attention in research, despite its pivotal role in cybersecurity. Categorizing malware into families provides essential insights for defenders and researchers alike. It serves as a foundation for understanding the evolving tactics and techniques employed by malicious actors. Recognizing patterns and similarities among malware variants aids in predicting forthcoming attacks, thus enabling proactive defense strategies.

In response to these challenges, a new initiative has emerged – a repository dedicated to the comprehensive evaluation of malware and their respective families. This repository represents a valuable resource for researchers and defense teams alike. It empowers them to conduct thorough assessments of malware samples and facilitates the identification of common traits and behaviors within specific malware families. By fostering a deeper understanding of these malicious entities, this repository equips the cybersecurity community with a powerful tool for staying ahead of evolving threats.

1.4 Problem Statement

There are several methods for extracting features from static analysis in order to use ML to identify malware. Foregoing research has retrieved characteristics from printed strings[12] in malware binaries based on the length of the disassembled file's functions, n-gram instruction s [13], or a combination of several non-graph features[1]. Finally, focuses on feature produced through FCG [10]. The primary reason for this is that, when compared to n-gram features, FCGs better maintain structural information in binaries. They contain malicious code information in form of procedures [14], as well as information about how the functions interact with one another.

1.5 Research Objectives

Objective of this research is to construct a model able to conduct static analysis on PE files to discover and categorize malware. This model will be utilized for security purposes in various computer and IT fields.
The research stated goal is to accomplish the following features and benefits.

- To detect malware using machine learning approach.
- To classify into malware types and families
- To categorize malware into three classifications.
    1. Binary Classification
    2. Category Classification
    3. Family Classification
- Windows user security at the organizational and individual levels.

1.6 Research Methodology

The following is the primary methodology of this study:

- An executable file will be given to the application.
- Application will extract features from the file.
    1. Function Call Graph
    2. APIs and Class Imports
- Application will generate the CSV file of features.
- From the CSV file, DL model will be trained and will be tested.
- DL model will detect and classify the malware.

1.7 Thesis Organization

The residue of thesis is organized as : Following chapter discusses existing methodologies; Chapter 3 includes thorough research methodology; Chapter 4 includes analysis and outcomes of the research methodology; and the final portion concludes with a conclusion and future research goals.

CHAPTER 2

RELATED WORK

Within the current body literature, the task to identify and classify malware is addressed through a diverse range of methodologies, which can be mainly categorized into two streams: static and dynamic analyses.

With each passing year, the proliferation of malware continues to rise. Consequently, online activities are increasingly viewed with a sense of caution and concern due to these ongoing malware threats. Researchers have been actively involved in this domain since as early as 1995. However, the evolution of DL and ML techniques for the detection and classification remains an ongoing endeavor. The evaluation of malware encompasses both dynamic and static approaches. Realm of static analysis, code and script malware samples undergo meticulous scrutiny, focusing on their functionality, behavior, and potential consequences. Conversely, dynamic analysis involves delving deeper into the code and executing systems within a controlled environment to assess their real-time effects.

Figure 2.1 presents a comprehensive block diagram illustrating the spectrum of static analysis techniques employed for the detection of malware. In the forthcoming sections, we will delve into each of these techniques, examining their historical usage and contributions to the malware identification and categorization.

2.1 Static Analysis Techniques

Are used to analyze and assess software (e.g., executable files, scripts) without executing them. These techniques help identify characteristics, patterns, or code structures that are indicative of malware behavior or potential security threats. Here are some static analysis techniques commonly employed in malware detection:

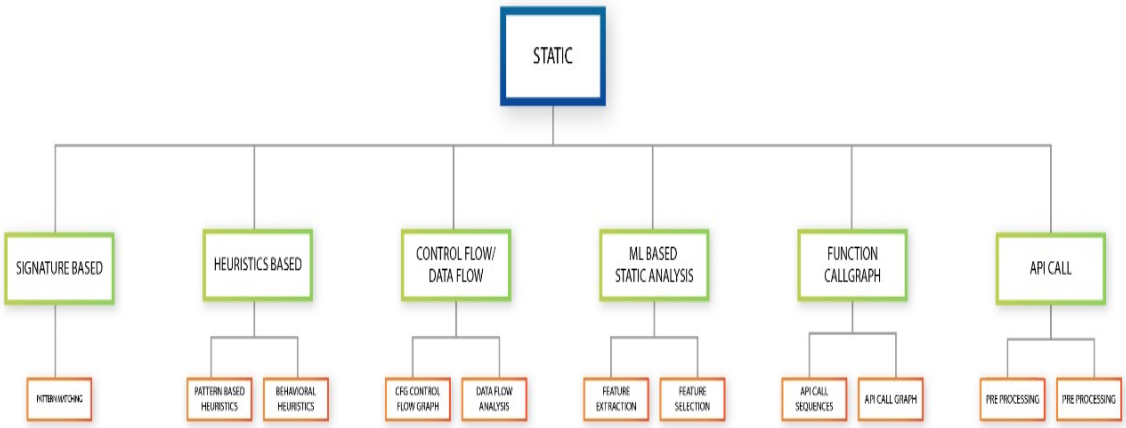

Figure 2.1: Static Analysis Techniques

a) **Signature Based Detection**

Signature-based detection entails the search for particular specimen or signatures within a program's binary code. These signatures are predefined and represent known malware or malicious behaviors. Effective for detecting well-known malware; relatively fast. Ineffective against zero-day threats and polymorphic malware, as it relies on predefined signatures.

In the realm of malware analysis, a function call graph emerges as a remarkably resilient and enduring representation of a program, especially when compared to traditional byte or hash signature techniques. This research [8] venture boldly positions the FCG as central trademark of a program. It harnesses the power of two distinct graph isomorphism methods to discern between known malware and its myriad variations.

The outcomes of the experiments conducted firmly validate the effectiveness and efficiency of this pioneering approach. It excels in the identification of well-established strains of malware and select variants, demonstrating its potential to be a cornerstone in the task of indexing and identifying a vast repository of malware specimens. Furthermore, it exhibits promise in the classification of malware into discrete families, promising a holistic solution in the field of malware analysis.

b) **Heuristic-Based Detection**

Heuristic analysis looks for patterns or behaviors that are indicative of malware but may not be explicitly defined in signatures. It often involves identifying suspicious characteristics like code obfuscation or unusual API calls. Can detect variants of known malware and previously unseen threats. May produce false positives due to its reliance on heuristics.

The significance of this contribution [6] lies in its adaptability and versatility. Instead of being limited to a single domain, this innovative representation proves its potential across diverse applications where Function Call Graphs (FCGs) are crucial. Whether in cybersecurity, software analysis, or any field dealing with intricate program structures, this paper promotes more effective and precise data representation, thereby advancing the state of the art in this domain.

c) **Control Flow Graphs**

Control flow analysis is a fundamental static analysis technique used in the context of malware detection. It focuses on examining how the program's instructions and statements are structured, connected, and executed. This analysis can help identify anomalies, suspicious behavior, or patterns indicative of malware. Example of control flow graph in figure 2.2

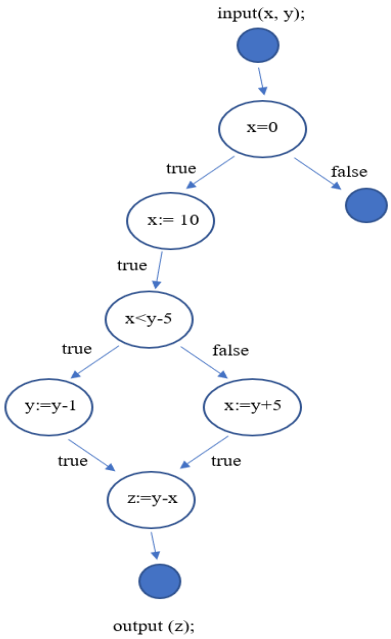

Figure 2.2: control flow graph

As society increasingly relies on computer systems, the threat of malicious software becomes more severe. While machine learning methods like GBDT and deep neural networks can handle cyber threats, most rely on statistical information from PE files. To address this, study [15] introduces a malware classification system on Control-Flow Graphs (CFG) and Graph Isomorphism Networks (GIN).
CFG basic block feature vectors are generated using MiniLM, benefiting GIN for compression and classification via multi-layer perceptron. Evaluation on the Malware Geometric Binary Dataset (MGD-BINARY) demonstrates high accuracy (0.99160) and AUC (0.99148) results.

d) **Function Call Graph**

FCG analysis, a technique used in software engineering and malware analysis to create and analyze a graph that shows the hierarchy between functions or methods within program. This analysis provides insights into how different functions or methods call one another, their dependencies, and the flow of control and data through the program. In the context of malware analysis, function call graph analysis helps identify suspicious or malicious behavior by examining the function calls within a program or binary.

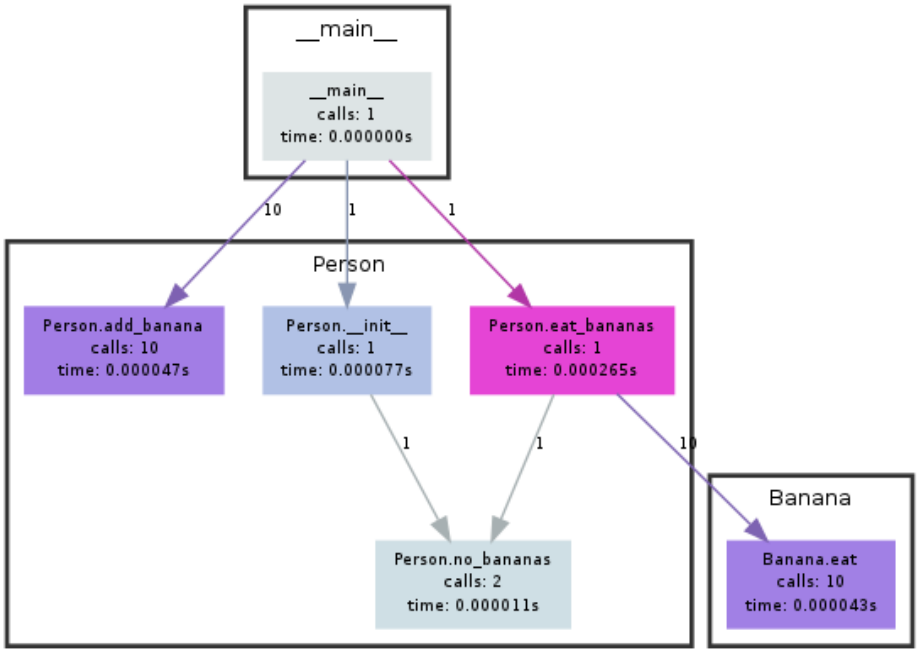

Figure 2.3: Function Call Graph

Detail literature review have been discussed in this section regarding

FCG's.

The authors [6] present an innovative methodology that creates vector presentation of the linear FCG through feature cluster. This approach is remarkable for its capacity to significantly improve performance, resulting in substantial gains in classification accuracy. It's not confined to theory; the paper provides empirical evidence demonstrating the seamless integration of this representation with non-graph elements.

This study [3] seeks to boost the classification accuracy of a ML model utilizing function call graph Vectors (FCGV) by combining non-graph and graph features. It proposes Random Forest classification model using FCGV and Statistical Features (SF) from Portable Executable (PE) files. It acknowledges that hashing in FCGV may lose vital PE file properties. To address this, six non-graph features like metadata, register, data definition, symbol, section, and operation code are integrated into a unified vector. This comprehensive approach aims to enhance model accuracy by considering both graph and non-graph PE file characteristics.

GEMAL [16] introduces an innovative approach to malware analysis using function call graphs (FCGs) and a graph embedding network. FCGs store critical structural details about binary files, which have proven valuable in prior malware research. In this method, instructions are likened to words, while functions are likened to sentences, facilitating the automatic extraction of semantic features.
To generate embedding vectors for malware, streamlining the analysis process, we utilize a graph embedding network featuring an attention mechanism. This network effectively merges both structural and semantic characteristics, thereby enhancing our overall analytical capabilities.

e) **API**

Examining sequence and frequency of API calls made by program can reveal patterns associated with malware behavior. It often involves creating API call graphs. Useful for detecting malicious behavior that relies on specific API calls. Limited to the analysis of API calls and may miss other aspects of malware behavior.

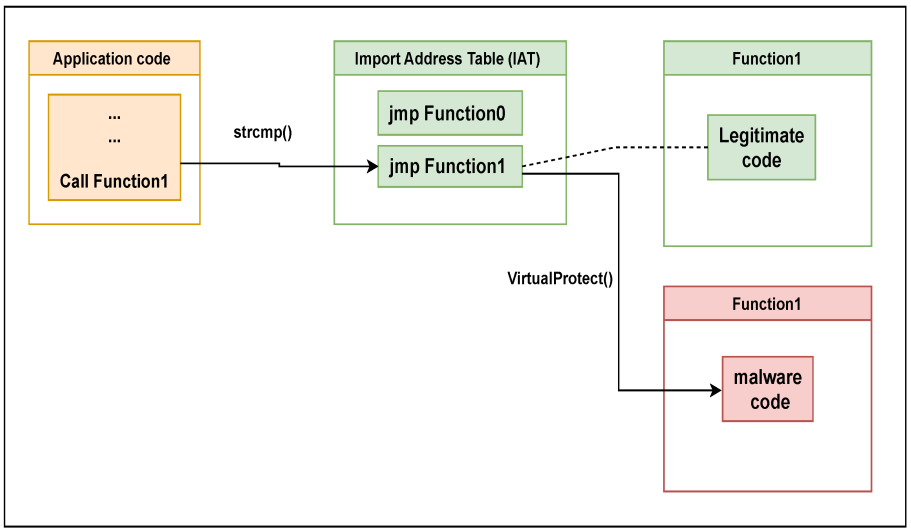

Figure 2.4: API Call Analysis

Traditional approaches in malware analysis heavily relied on gathering Application Programming Interface (API) data through dynamic analysis. However, the effectiveness of this approach was limited due to the diverse evasion techniques employed by malware creators. Consequently, API invocation data often failed to provide an accurate depiction of malicious software actions.

In response to this challenge, the paper's author [17] introduces an innovative malware classification method that places a central focus on byte-level information. This cutting-edge approach utilizes a deep learning algorithm to convert malware byte data into images, effectively encapsulating the malware's behavioral context. These resultant images undergo further analysis using a convolutional neural network-based phrase analysis technique.

APIs form the basis of common malware detection techniques, primarily relying on statistical API features. Yet, malware creators evade these methods by changing API call sequences or parameters. Existing detectors either overlook arguments or demand extensive resources for their analysis. This study [18] introduces a lightweight API-based dynamic feature extraction method using machine learning, enabling malware detection and classification without the need for expertise in API arguments. Evaluation on dataset spanning ten malware types yields impressive results, with the malware detection module outperforming state-of-the-art API-based detectors in accuracy.

We have reviewed other existing analyses that have aided our understanding of areas that have not been fully researched or thoroughly experimented.

The article [11] delves into the importance of static malware analysis and detection in the context of addressing security issues. It specifically underscores the challenge of dealing with imbalanced datasets within static analysis. The paper puts forth a model designed to proficiently construct a attribute set from dataset to identify PE files. However, rather than concentrating solely on model development, the research places a strong emphasis on the significance of feature extraction. The study illustrates that well-extracted features, when fed into neural networks with a limited number of layers, significantly enhance overall results. In essence, the paper offers valuable insights for advancing static malware analysis by emphasizing the crucial role of feature extraction.

A commonly used technique in surface analysis involves extracting printable characters from portable executable (PE) files. Recent advancements in NLP has enabled the swift detection of malicious PE files. In this study [12], the authors evaluated this approach by applying it to the latest FFRI dataset. This research is noteworthy as it is the first to encompass both malicious and benign time series data in its investigation.

Table 1 contain some commonly employed techniques utilized to perform detection and classification using static analysis.

Table 2.1: Deep Learning Models

	YearRef	Objectives	Ext Source	Dataset	Acc	Limitation
DL Model	2022 [11]	Detection and Classification Using Multi-layer Deep Learning	Binaries	EndGame Inc	97.52	Time complexity
	2021 [19]	An innovative hybrid method that combines deep learning is applied for malware classification	Image Files	Malimg, BIG 2015, Male Vis	97.7	Didn't test the adversary's attacks with crafted inputs
	2021 [20]	Malware Classification Using LSTM	OP codes	Malicia, Virus Share	81	Combined family classification degrades accuracy
	2020 [21]	LSTM based Malware Classification	Wins API Calls	Self created	95	Dataset not correctly labelled
	2020 [22]	Multimodal deep learning framework for malware classification	API's, Bytes, OP codes	MS BIG 2015	0.99	Pretrain each component to avoid overfitting

Table 2.2: Machine learning Models

	Year	Ref	Objectives	Ext Source	Dataset	Acc	Limitation
ML	2022	[16]	Detection and Classification Using NLP	FCG	WUFCG	99.16	False Positive Rates are higher
	2022	[12]	Malware detection using ML and NLP model	Printable Characters	FFRI	0.981	Samples are not distributed
	2020	[3]	Classifying malware using call graph vectorization and ML	FCG, System call	MS BIG 2015	0.99	Only malware sample used
	2020	[18]	Malware detection and classification on advanced Windows methods	API Calls	Malshare	98	Dataset is small due to system resources
	2019	[23]	Using the ML approach, achieve high-fidelity detection in reduced time frame	PE Headers	Virus Share	99.68	Training time is not feasible

Table 2.3: Function Call Graphs Models

	Year	Ref	Objectives	Ext Source	Dataset	Acc	Limitation
FCG	2021	[24]	Coarsening technique to construct multi-level static call graph representations	FCG's	—	—	Graph presented in static format, which is hard to understand
	2019	[8]	Isomorphism Algorithm used to identify malware and variants	FC Graph	VX Heaven	99.2	Cannot identify Unknown Malware
	2017	[6]	Classifying malware using call graph vectorization	FCG's, OP codes	MS BIG 2015	0.98	Only malware sample used
	2013	[10]	Obfuscated malware detection based on FCG similarity metric	FCG's	VX Heaven	90	FCG's aren't complete
	2016	[9]	Detection and classification using ScD Graphs and Similarity graphs	System call graph	Virus Total	94.7	Loss of Information

CHAPTER 3

METHODOLOGY

The proliferation of malware instances has rendered manual analysis by human experts increasingly challenging. Consequently, machine learning approaches have emerged as a powerful strategy to combat this surge in malware. Machine learning algorithms, such as Naive Bayes [17], Support Vector Machine, Random Forest Tree [3], and others, facilitate the classification, regression, and clustering of malware by subjecting input data to statistical analysis or established algorithms. DL, a subset of ML, builds upon artificial neural networks and learns from specimen. This innovative method finds broad application in fields like image processing, autonomous vehicles, and voice recognition. Unlike traditional machine learning algorithms, deep learning [1]derives intricate features from vast datasets, thus reducing the reliance on domain expertise for feature extraction. Consequently, recent years have witnessed the widespread adoption of DL in the field of malware analysis.

Graphs serve as versatile data structures capable of effectively capturing relationships among diverse entities. They have gained significant traction in recent times, leading to the emergence of graph-centric approaches. For example, Li et al [25] introduced a DL framework for android detection, leveraging embedding of API graphs. Similarly, Zhang et al [26] developed a multi-attribute heterogeneous graph convolutional network for robust bot activity identification. Furthermore, the innovation of HGDom introduced a diverse graph convolutional network adept at detecting hostile domains. Beyond the realm of malware detection, graph-based techniques have proven effective in various domains [27]. For instance, industrial control systems have employed graph-based methodologies to estimate suspicious communication patterns.

Figure 3.1 illustrate the overall flow of our model. Decompiling exe files to generating FCG's then convert them onto adjacent list. Creating a database

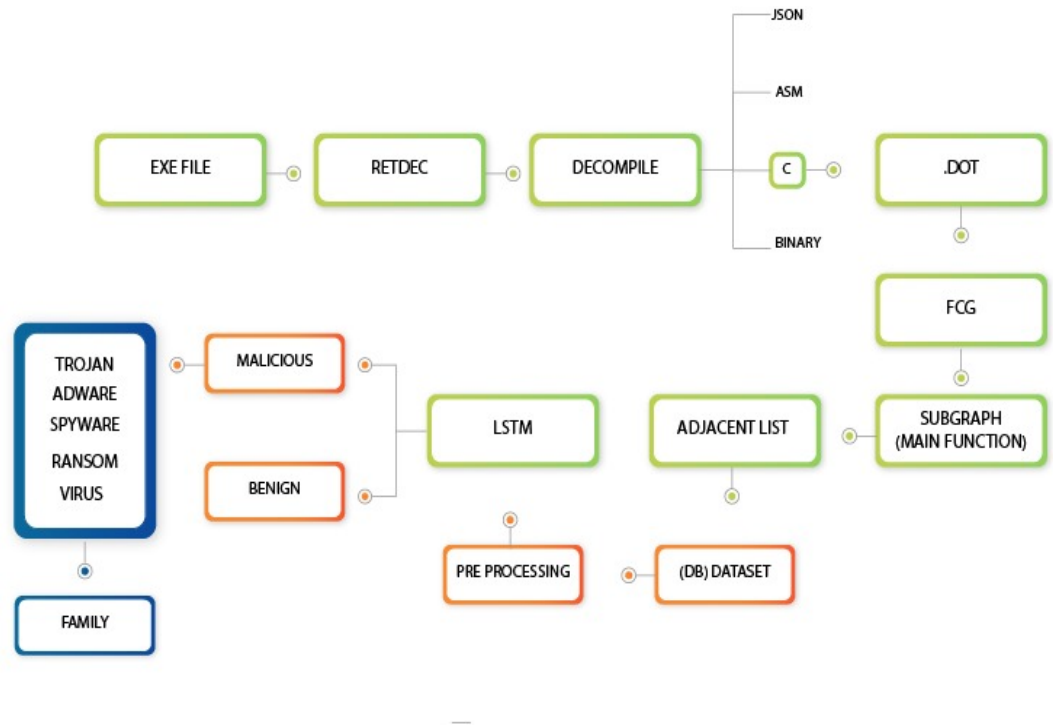

Figure 3.1: Overview of Model Architecture

and generate labelled dataset. Apply pre-processing techniques on dataset to remove unnecessary features and atlast implement the LSTM model on dataset. All these steps have been described below with detail.

3.1 RETDEC Tool

RETDEC is retargetable machine-code decompiler based on LLVM. It provides call graph of PE file in the form of .dot (Graphviz) file, which can be visualize through different tools.

Features:

1. Examination of executable files through static analysis with comprehensive details.
2. Recreation of functions, data types, and higher-level structure
3. Output provided in C and python languages
4. Generation of FCG's and CFG's

3.2 Graph Analysis and Database

a) Develop a script to convert the .dot file generated by retdec into a graph structure.

b) Implement algorithms to extract the main or entry graph from a collection of multiple graphs.

c) Test the graph extraction process on various PE files to ensure accuracy and reliability.

d) Optimize the graph extraction algorithms for efficiency and performance.

The objective of this milestone was to enhance the analysis and visualization of disassembled code by transforming the generated .dot files into a structured graph representation shown in Fig 3.2

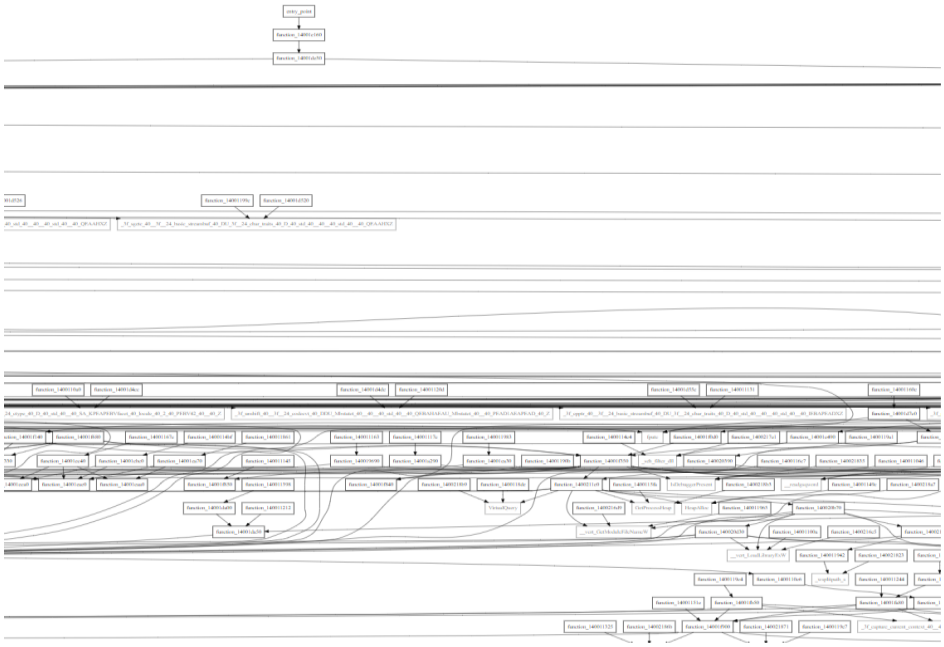

Figure 3.2: Generated Function Call Graph

In the initial phase, we developed a script capable of efficiently handling the .dot files provided by RETDEC, leveraging the Networkx and PyGraphviz libraries. These libraries enabled us to interpret the intricate relationships and dependencies present in the disassembled code, presenting them in a graph format that aids in comprehension. The subsequent step entailed implementing

algorithms specifically designed to extract the primary or entry graph from a multitude of charts. This step was crucial in isolating the fundamental logic within the analyzed executable files, providing a clear depiction of their functional flow. By concentrating on the entry graph, our objective was to distill the most relevant information and comprehend the core program execution paths.

To validate the precision and reliability of the graph extraction process, we subjected a diverse range of Portable Executable (PE) files to our analysis pipeline. This systematic evaluation allowed us to scrutinize how the graphs were generated from the call structures within the disassembled code. This critical assessment helped us ascertain the accuracy of the graph extraction process, ensuring that the visual representation accurately reflected the intricate calling relationships within the code.

Furthermore, we optimized the graph extraction algorithms to enhance efficiency and overall performance. One of the key optimization strategies employed involved integrating a thread pool mechanism, facilitating concurrent processing of multiple files. This approach effectively reduced the time required for graph extraction, contributing to a more streamlined and expedited analysis process.

3.3 Dataset Generation

For the generation of dataset, we follow those steps.

a) Set up a SQLite database to store the extracted graphs' adjacency lists.

b) Design a database schema to efficiently store and retrieve graph data.

c) Develop functionality to convert the main or entry point graph into an adjacency list representation.

d) Implement the storage of the adjacency list as a string in the SQLite database.

e) Perform extensive testing to ensure the correctness of data storage and retrieval operations.

Initially, we targeted the main function or entry point function within the disassembled code as discussed in section 3.2. By parsing the code and

identifying these essential components, we constructed an adjacent list that outlined the code's structural connections and dependencies. This adjacent list provided a concise representation of how functions within the codebase interacted, capturing the underlying logic of the program's execution flow. Fig 3.2 shows adjacency list that represent graph where nodes are connected to each other based on relationships specified in the lists.


```
{
  "Node_function_40aa10":
      ["Node_cygwin_internal", "Node_vsnprintf", "Node_abort", "Node_cygwin_conv_to_posix_path", "Node_WriteFile", "Node_GetStdHandle"],
  "Node_function_40a790":
      ["Node_cygwin_internal", "Node_function_40b1c0", "Node_GetModuleHandleA"],
  "Node_function_40a730":
      ["Node_dll_crt0__FP11per_process", "Node_function_40a790"],
  "Node_entry_point":
      ["Node___asm_int3", "Node_function_40a730"],
  "Node_function_40ac70":
      ["Node_memcpy", "Node_function_40aa10", "Node_VirtualQuery", "Node_VirtualProtect"],
  "Node_function_40b1c0":
      ["Node_function_40ac70"]
}
```


Figure 3.3: Adjacent List

Subsequently, we transformed the generated adjacent list into a coherent string format. This string encapsulated the intricate relationships between different functions, encapsulating the essence of the code's functional structure. These string representations were then stored within a dedicated database, accompanied by labels that denoted the nature of the code as either "Benign" (label 0) or "Malicious" (label 1). This labeling facilitated the classification of code instances for machine learning or deep learning purposes, contributing to subsequent analysis and modeling efforts.

To facilitate further processing and model application, we exported the amassed dataset from the database in CSV format. This exportation transformed the structured data into a more accessible form, enabling seamless integration with a wide range of machine learning or deep learning frameworks. The CSV files served as a crucial resource for training, validating, and test-

ing predictive models, thus amplifying the potential for our project's broader impact.

3.4 Dataset Pre-Processing

1. Remove data columns, metadata, and features that are unnecessary or redundant to the malware detection activity.
2. Clean the dataset to retain only relevant information.
3. Each sample in the dataset is correctly labeled with its corresponding malware class or benign label.
4. Data labeling significantly improves and impact model performance.
5. Split our dataset into training, validation, and test sets.
6. Convert the textual data, such as function call sequences, into numerical vectors.

3.5 Long Short Term Memory

We initiated a comprehensive examination of numerous ML and DL models in order to enhance our grasp of function call graphs and their inherent characteristics. Throughout our analysis journey, we applied multiple models, each meticulously tailored to address specific facets of the function call graph's representation and behavior.

A prominent model in our arsenal was the LSTM model. In order to make the most of LSTM's expertise in handling sequences, initially transformed the adjacent list obtained from the function call graph into a structured input format. This transformation was achieved through the utilization of a tokenizer, which translated the intricate relationships between functions into a format conducive to effective analysis by LSTM. LSTM uses three layers: one Bidirectional layer, 2 dense layers. This adaptation allowed us to capture temporal dependencies within the code's execution flow, thereby enhancing our predictive capabilities.

Algorithm

Data: Windows Executables (.exe) Files
Result: File is Malicious or Benign

files - Exe files from the selected directory;
foreach file in files **do**
  Dot File $\leftarrow retdec(file)$;
  Graph $\leftarrow nx(DotFile)$;
  SubGraph $\leftarrow Graph(Node - Main)$;
  adjList $\leftarrow GraphToAdjList(SubGraph)$;
  Store AdjList (adjList):
  token $\leftarrow Tokenize(adjList)$;
  token $\leftarrow sequence, adding(token)$;
  token $\leftarrow wordindering(token) result - LSTM(token)$;
Show(result)end

CHAPTER 4

ANALYSIS & RESULTS

This chapter represents implementation detail along with result analysis.

4.1 Implementation of Proposed Framework

The architecture of system our system consist of two section. The first section contain brief detail of feature extraction process; How FCG's have been extracted.

The second section defines how dataset have been created then pre-processing of the dataset and after that the implementation of LSTM model.

4.1.1 Feature Extraction

In feature extraction section, retdec tool is used to decompile the exe file. During decompilation process, various files have been opted. Retdec tool provide us function call graph of exe file in the form of .dot (Graphviz) file which later converted into graph structure.

4.1.2 LSTM Model

The model architecture in this part includes an embedding layer for representing words as dense vectors, a bidirectional LSTM layer for capturing sequence information, and two dense layers for classification. For multi-class text classification, the model is trained using categorical cross-entropy loss and the Adam optimizer.

 # Build and train

```
model = tf.keras.Sequential
([

    tf.keras.layers.Embedding(input_dim=len(tokenizer.word_index) + 1, out
    put_dim=100, input_length=max_seq_length),
    tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(64)),
    tf.keras.layers.Dense(64, activation='relu'),
    tf.keras.layers.Dense(labels.shape[1], activation='softmax')
])
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
```

 # Training

```
history = model.fit(X_train, y_train, epochs=20, node_size=32,
validation_data=(X_test, y_test))
```

The provided code defines and trains a neural network model for text classification using TensorFlow and Keras. Let's break down the architecture step by step:

a) **Sequential Model**

The 'tf.keras.Sequential' function is employed to construct a layers of linear stack. Each layer is added sequentially, one after the other.

b) **Embedding Layer**

- **Input Dimension ('input_dim'):** This is set to the length of 'tokenizer.word_index' + 1. It represents the terminology size, and the "+1" accounts for an index reserved for out-of-vocabulary (OOV) words.
- **Output Dimension ('output_dim'):** The output of the embedding layer for each word is a vector of length 100. This means that each word in the vocabulary will be represented as a 100-dimensional vector.

- **Input Length (‘input_length‘):** This is set to ‘max_seq_length‘, which indicate greatest feasible length of input sequence. It ensures that input sequences of different lengths are padded or truncated to this length.

c) **Bidirectional Layer**

- With 64 nodes, this layer involves Long Short-Term Memory (LSTM) layer. The LSTM is a sort of recurrent neural network, also known as an RNN, that works well with sequence data.
- The "Bidirectional" wrapper generates two distinct LSTM layers, with one processing the input sequence from left to right, and the other from right to left. This bidirectional approach enables the model to capture dependencies in both forward and reverse directions, enhancing its ability to understand sequential patterns.

d) **Dense Layers**

- There are two dense layers in the model.
- The first layer has 64 nodes and uses the ReLU (Rectified Linear Unit) activation function.
- The second dense layer has the same number of units as the number of unique labels in your classification task (represented by ‘labels.shape[1]‘). It uses the softmax function, which commonly used for multi-class classification problems.

e) **Model Compilation**

- The ‘model.compile‘ method used to configure the training process.
- **Loss Function:** The loss function employed in this context is categorical cross-entropy (categorical_crossentropy), which is well-suited for multi-class classification.
- **Optimizer:** The Adam optimizer is chosen as the optimization algorithm.
- **Metrics:** The model will track accuracy as a metric during training.

f) **Training**

- The ‘model.fit‘ method is used to train the model.
- ’X_train’ and ’y_train’ are the simulated data and tags.
- The training process will run for 20 epochs (iterations over the entire training dataset).
- With a group size of 32, the variables of the model are altered after processing.
- Validation data denoted as ‘(X_test, y_test)‘ is provided to monitor model’s performance on a separate dataset during training.

4.2 Result & Analysis

The detection and classification framework discussed in this thesis has been used to analyse FCG and extract vital information. This section examines these results to demonstrate that the method suggested in this study is capable of accurately detecting and classifying malware. Our system extract FCG’s from exe files and using those call graphs we implement our LSTM model.In prior work, detection [18]of malware and classification is also used.

4.2.1 Experiment Evaluation

In this section, we delve into our malware detection, which are conducted using dataset. However, before delving into the assessment of experimental outcomes, we provide an overview of the datasets and implementation particulars. Notably, all experiments within this section are assessed using exe files.

4.2.2 Dataset

We have curated a proprietary dataset comprising a total of 1194 samples, drawn from diverse sources such as Virus Share and Malware Bazzar. This dataset encompasses a spectrum of malicious software specimens, spanning six distinct categories: Trojan, Worms, spyware, virus, fileless malware, and ransomware. To ensure the dataset’s integrity and balance, each of these categories consists of precisely 200 samples. This meticulous curation process results in a well-balanced dataset that serves as a robust foundation for clas-

sification tasks and research endeavors.
Figure 4.1 shows the count of total and labelled samples in dataset.

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1194 entries, 0 to 1193
Data columns (total 2 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   list    1194 non-null   object
 1   label   1194 non-null   object
dtypes: object(2)
memory usage: 18.8+ KB
```

	list	label
count	1194	1194
unique	1092	6
top	"{\"Node_entry_point\": [\"Node__CorExeMain\"]}"	bot
freq	32	200

Figure 4.1: Dataset

4.2.3 Training & Validation

The study encompasses two distinct datasets, partitioned into two core segments: training and testing. Notably, 20% of the datasets are earmarked for testing purposes. During the training phase, the models are supplied with Function call sequences to predict the corresponding malware family class. Given the inherent challenge of handling highly imbalanced datasets, a pivotal step involves preserving the distribution of classes. To address this, the process of data splitting adheres to a stratified methodology. Furthermore, for each dataset, a Stratified 5-Fold approach is employed on the training data.

Figure 4.2 and 4.3 shows accuracy and loss while training and validation. Increasing in the epoche shows that accuracy increases while loss decrease on training and validation data.

Accuracy: During the initial training phases, the accuracy curve remains relatively stable. Both the training and validation curves achieve nearly 99 percent accuracy by the 5th epoch and then maintain a steady, oscillatory pattern until the 18th epoch. Throughout this period, there is a consistent

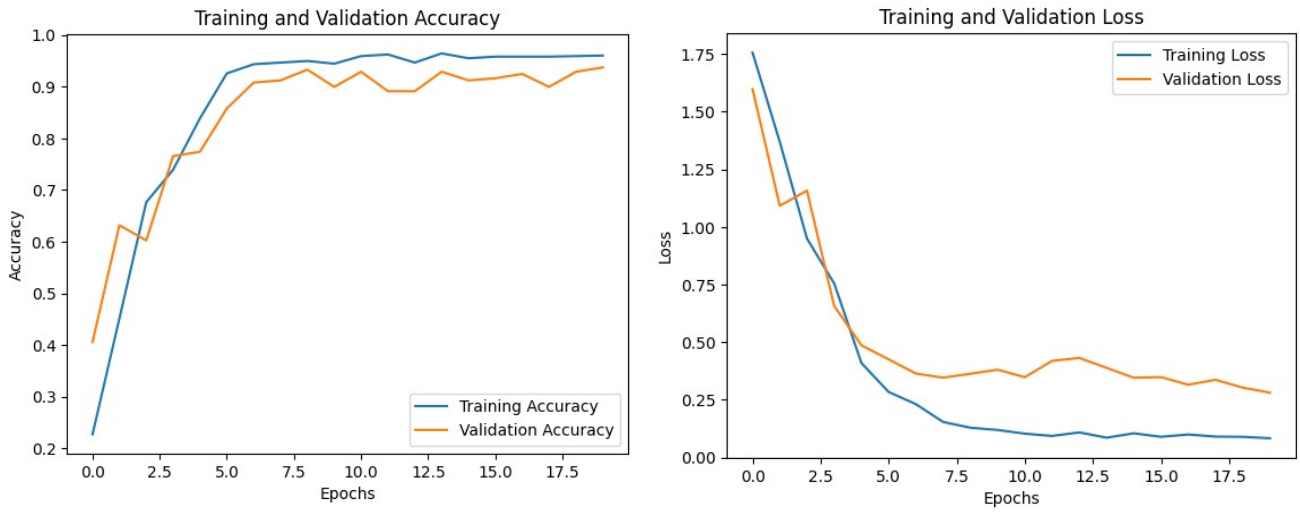

Figure 4.2: LSTM Model Accuracy/Loss

difference of 0.005 in the oscillation between the two curves. This persistent accuracy across consecutive epochs underscores the model's high classification efficiency.

Loss: Loss is computed to assess the model's performance after each optimization iteration. Initially, the loss is quite high, but it experiences a sharp decline after the 4th epoch. Beyond the 7th epoch, the loss for the validation set continues to decrease at a gradual pace and eventually stabilizes at a value less than 0.02. This remarkably low loss value indicates that the model's weights are being optimized very effectively after each epoch.

4.2.4 ROC-AC

The goal is to categorize data points into one of two classes, often referred to as "positive" and "negative."

1. **True Positive (TP) and True Negative (TN):**
   - **True Positive (TP):** Instances that the model accurately defines as positive.
   - **True Negative (TN):** Instances that the model effectively detects as negative.

2. **False Positive (FP) and False Negative (FN):**

   - **False Positive (FP):** Cases that are labelled as positive when they are actually negative.
   - **False Negative (FN):** Events that are categorised as negative when they are, in fact, positive.

3. **Sensitivity (True Positive Rate):** Examines the model's ability to effectively identify positive instances.

$$\text{TPR=TP / (TP + FN)}$$

4. **Specificity (True Negative Rate):** The model's ability to accurately identify negative cases is evaluated by the following calculation:

$$\text{TN / (TN + FP)}$$

5. **ROC Curve:** The ROC curve is an illustration of how well a model performed at various decision points. For various threshold values, it compares the TPR vs FPR. Formula to calculate FPR

$$\text{FPR=FP / (FP + TN)}$$

An AUC-ROC curve has been generated, with sensitivity represented on y-axis and specificity on x-axis. Figure 4.3, it is evident that the LSTM achieves a notably high AUC value of around 0.98. This indicates a 98 percent probability of accurately classifying calls as either malware or goodware. Consequently, it can be inferred that the LSTM model demonstrates enhanced classification efficiency, surpassing the performance of the other deep learning algorithms that were employed.

4.2.5 Confusion Matrix-F1 Score

A confusion matrix with the six classes. The confusion matrix will have six rows and six columns, representing the actual and predicted class labels. The inclining elements of the matrix represent the number of correct classifications for each class.

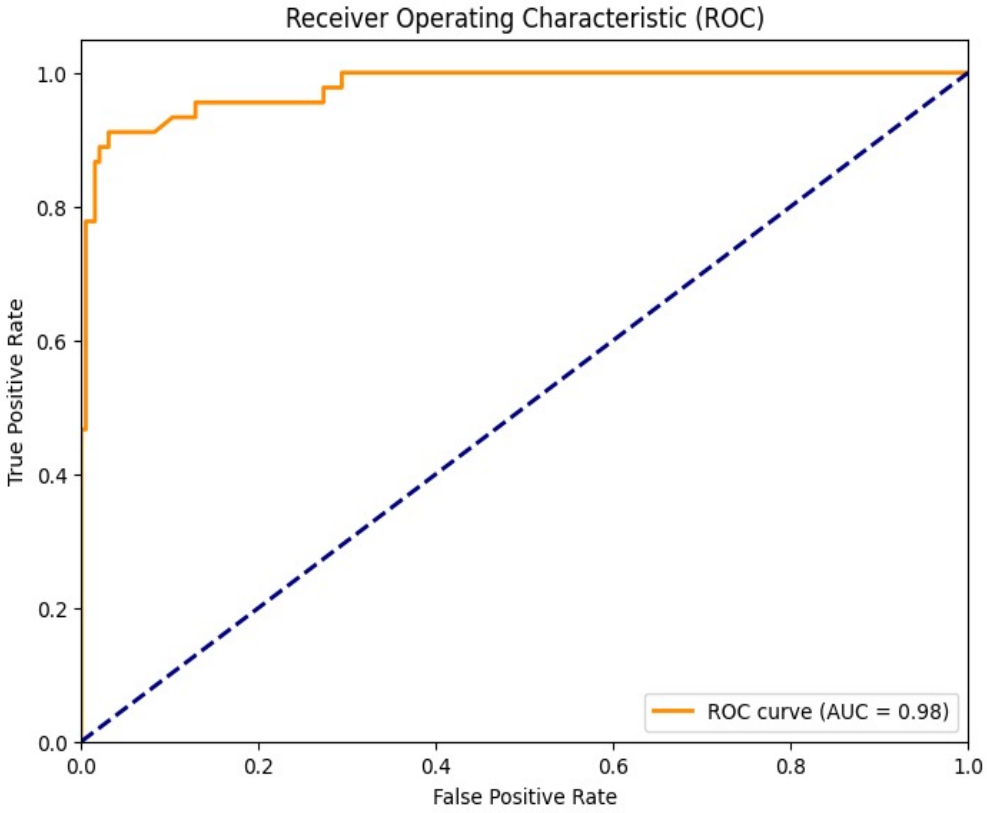

Figure 4.3: LSTM ROC-AUC

Figure 4.4 displays a comprehensive confusion matrix. Utilizing this matrix, we can calculate various metrics for each class, including accuracy, precision, recall, and F-1 score.

Using above values in the confusion matrix, we can calculate various performance metrics to assess the model's classification accuracy, such as:

Accuracy:

$$(TP + TN) / (TP + TN + FP + FN)$$

Precision:

$$TP / (TP + FP)$$

Recall (Sensitivity):

$$TP / (TP + FN)$$

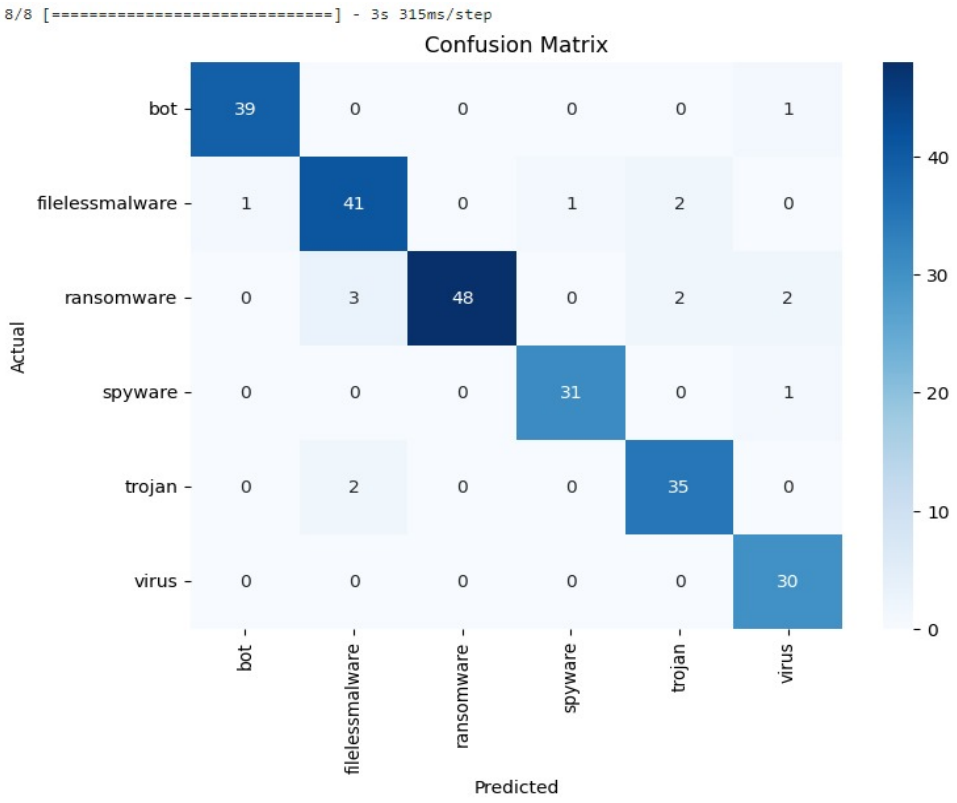

Figure 4.4: LSTM Confusion Matrix

F1-Score:

$$2 * (\text{Precision} * \text{Recall}) / (\text{Precision} + \text{Recall})$$

4.2.6 BenchMarking of Models

Bench-marking employs two distinct models, encompassing a diverse range. First one is Graph neural Network. Additionally, the ensemble category is represented by two models XGBoost and HGBoost. Introducing the realm of neural networks, we have Long Short-Term Memory (LSTM), which leverages specialized units in recurrent architecture.

Ensemble Model

Ensemble learning offers a structured methodology for harnessing the predictive power of multiple learning models. It leads to the creation of a unified model that synthesizes results from a multitude of individual models.

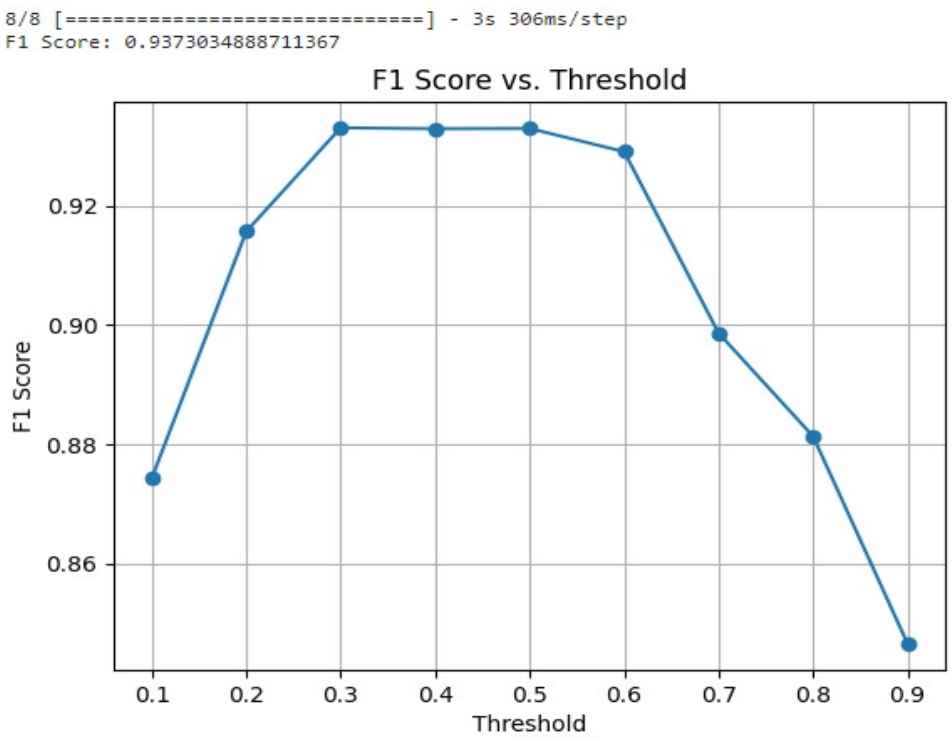

Figure 4.5: F1-Score

Two notable ensemble machine learning algorithms, XGBoost and HGBoost, are based on decision trees. These algorithms operate within a gradient boosting framework, which has gained popularity due to its remarkable effectiveness in diverse application domains and its efficient model management practices.

Accuracy: Initially, during the early stages of training, the accuracy curve remains relatively constant. However, the model demonstrates an impressive initial accuracy of approximately 93% at the outset. As training progresses, there is a gradual improvement in accuracy. The graphical representation implies that the model achieves an accuracy of around 96%, signifying that it accurately classifies approximately 96% of the calls as either malware or goodware. Moreover, the validation curve closely mirrors the training curve, indicating that the data points selected in the training dataset provide a representative sample of the overall dataset.

Loss: In the initial 20 epochs, it shows that the loss values have nearly stabilized, with a minimal gap between the two curves. This indicates a good fit for the model. However, it's important to note that training a well-fitted

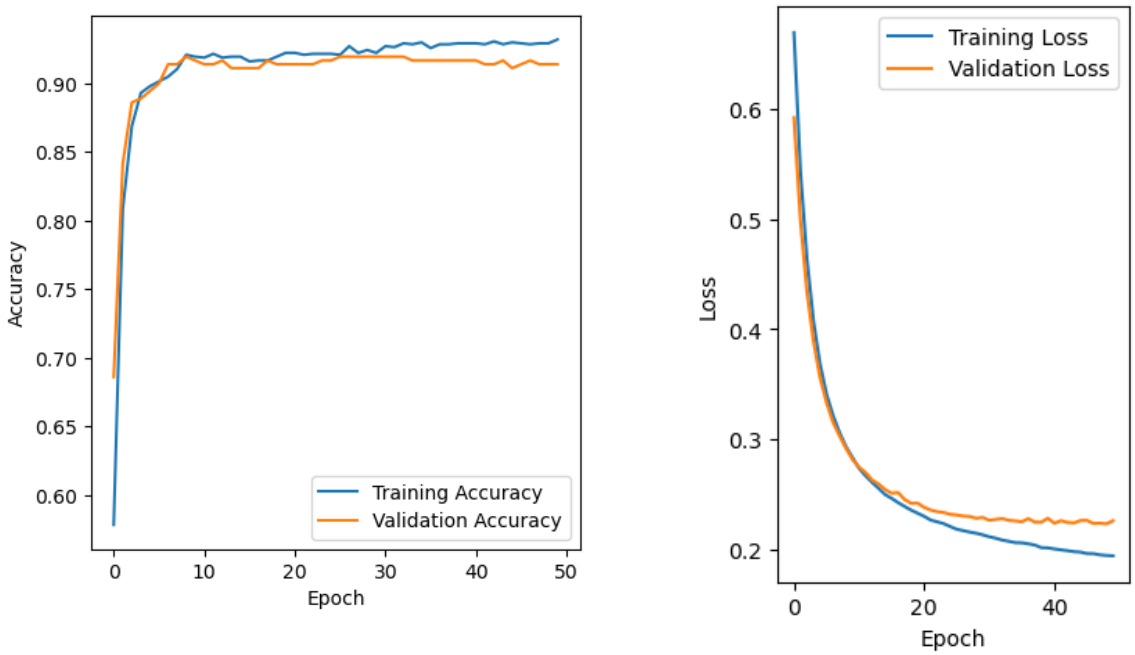

Figure 4.6: Ensemble Model Accuracy/Loss

model for an extended number of epochs can sometimes lead to over-fitting. In this case, the model is trained for a substantial number of epochs, and as a result, the loss value continues to decrease, indicating ongoing improvements in the classification model.

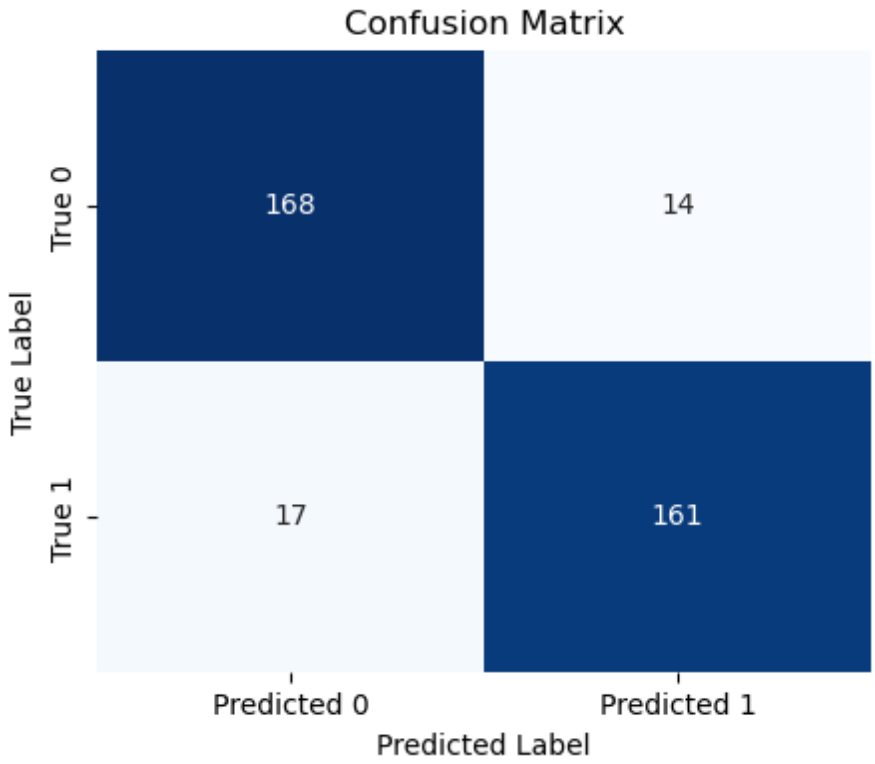

Figure 4.7: Ensemble Confussion Matrix

Graph Neural Network

Graph Neural Networks (GNNs) are a potent approach for training models in malware analysis. They excel in representing malware as a graph, with nodes denoting components (e.g., functions, APIs, system calls) and edges representing their relationships. GNNs are tailored for processing graph data, enabling them to capture the intricate dependencies in malware behavior. Their capacity to model complex interactions and hierarchies, generalize to new samples, makes them invaluable in malware analysis.

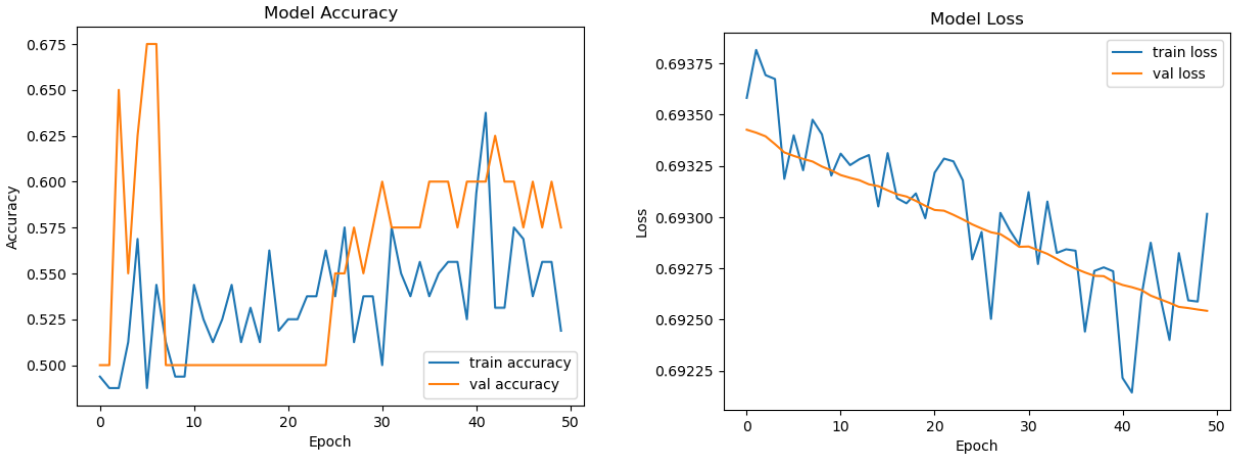

Figure 4.8: GNN Model Accuracy/Loss

The outcomes of the analysis shown in Table 4.1 reveals that, LSTM model outperformed all the other models in terms of F1-score, ROC across both versions of the VirusShare dataset. Shifting focus to the VirusSample dataset, distinct patterns emerged. In the case of the imbalanced version, LSTM emerged as the frontrunner with the highest F1-score.

Table 4.1: Dataset BenchMark

Model	**F1-Score (Original)**	**ROC Score (Original)**	**F1-Score (Balanced)**	**ROC Score (Balanced)**
XGBoost	0.71	0.96	0.75	0.95
HGBoost	0.69	0.95	0.74	0.94
GNN	0.69	0.92	0.72	0.90
LSTM	**0.90**	**0.96**	**0.93**	**0.98**

CHAPTER 5

CONCLUSION & FUTURE WORK

5.1 Conclusion

In conclusion, our project's analysis has illuminated the intricate nature of the complex data we've been handling. Notably, it has become evident that distinguishing between graphs belonging to different classes presents a formidable challenge, often with only minor variations of two or three nodes among hundreds. In response to this challenge, we propose a comprehensive approach to bolster our outcomes.

Our primary recommendation involves the adoption of a multi-model strategy, capitalizing on the unique strengths of diverse machine learning techniques. Firstly, we advocate for the implementation of separate models tailored to address specific aspects of the task at hand. To facilitate effective feature extraction, we endorse the use of GraphSAGE and Node2Vec models. These methods have consistently demonstrated their prowess in capturing intricate graph structures while preserving vital information during feature generation.

Furthermore, we propose the deployment of either GCN (Graph Convolutional Network) or LSTM (Long Short-Term Memory) models for the prediction and classification phase. These models have been proven to excel in tasks concerning graph data and sequential patterns, respectively. GCN leverages graph connectivity to enhance classification accuracy, while LSTM's proficiency in recognizing temporal dependencies offers significant advantages in certain contexts.

By seamlessly integrating these components, our approach aims to tackle the data complexity head-on and amplify the distinguishability between different graph classes. Through the combination of specialized feature extraction and precise prediction models, we anticipate achieving superior results

and gaining deeper insights into the underlying patterns embedded within the data. This multifaceted strategy promises to enhance the overall efficacy of our project, addressing the intricacies of our dataset and advancing our understanding of its intricate patterns.

5.2 Furure Work

Due to limited computational resources, we use a small dataset and and features related to the file. In future work, we can add resources and explore different techniques to enhance the scalability and work on large dataset. Due to this, we can efficiently test our model how it can bare the complex scenarios.

REFERENCES

[1] M. Naseer, J. Rusdi, N. Shanono, S. Salam, M. Zulkiflee, N. Abu, I. Abadi, Malware detection: Issues and challenges, Journal of Physics: Conference Series 1807 (2021) 012011. doi:10.1088/1742-6596/1807/1/012011.

[2] X. Hu, T.-c. Chiueh, K. G. Shin, Large-scale malware indexing using function-call graphs, in: Proceedings of the 16th ACM Conference on Computer and Communications Security, CCS '09, Association for Computing Machinery, New York, NY, USA, 2009, p. 611–620. doi:10.1145/1653662.1653736.

[3] Y. Zhang, X. Chang, Y. Lin, J. Mišić, V. B. Mišić, Exploring function call graph vectorization and file statistical features in malicious pe file classification, IEEE Access 8 (2020) 44652–44660. doi:10.1109/ACCESS.2020.2978335.

[4] A. Aslan, R. Samet, A comprehensive review on malware detection approaches, IEEE Access 8 (2020) 6249–6271. doi:10.1109/ACCESS.2019.2963724.

[5] A. Pektaş, T. Acarman, Classification of malware families based on runtime behaviors, Journal of Information Security and Applications 37 (2017) 91–100. doi:https://doi.org/10.1016/j.jisa.2017.10.005.

[6] M. Hassen, P. K. Chan, Scalable function call graph-based malware classification, in: Proceedings of the Seventh ACM on Conference on Data and Application Security and Privacy, CODASPY '17, Association for Computing Machinery, New York, NY, USA, 2017, p. 239–248. doi:10.1145/3029806.3029824.

[7] B. Ryder, Constructing the call graph of a program, IEEE Transactions on Software Engineering SE-5 (3) (1979) 216–226. doi:10.1109/TSE.1979.234183.

[8] J. Bai, Q. Shi, S. Mu, V. Conti, A malware and variant detection method using function call graph isomorphism, Sec. and Commun. Netw. 2019 (jan 2019). doi:10.1155/2019/1043794.

[9] S. Nikolopoulos, I. Polenakis, A graph-based model for malware detection and classification using system-call groups, Journal of Computer Virology and Hacking Techniques 13 (02 2017). doi:10.1007/s11416-016-0267-1.

[10] M. Xu, L. Wu, S. Qi, J. Xu, H. Zhang, Y. Ren, N. Zheng, A similarity metric method of obfuscated malware using function-call graph, J. Comput. Virol. 9 (1) (2013) 35–47. doi:10.1007/s11416-012-0175-y.

[11] A. Adamuthe, Improved deep learning model for static pe files malware detection and classification, International Journal of Computer Network and Information Security 14 (2022) 14–26. doi:10.5815/ijcnis.2022.02.02.

[12] M. Mimura, Evaluation of printable character-based malicious pe file-detection method, Internet of Things 19 (2022) 100521. doi:https://doi.org/10.1016/j.iot.2022.100521.

[13] T. Lee, B. Choi, Y. Shin, J. Kwak, Automatic malware mutant detection and group classification based on the n-gram and clustering coefficient, J. Supercomput. 74 (8) (2018) 3489–3503. doi:10.1007/s11227-015-1594-6.

[14] J. Kinable, O. Kostakis, Malware classification based on call graph clustering, Journal in Computer Virology 7 (09 2010). doi:10.1007/s11416-011-0151-y.

[15] Y. Gao, H. Hasegawa, Y. Yamaguchi, H. Shimada, Malware detection by control-flow graph level representation learning with graph isomorphism network, IEEE Access 10 (2022) 111830–111841. doi:10.1109/ACCESS.2022.3215267.

[16] X.-W. Wu, Y. Wang, Y. Fang, P. Jia, Embedding vector generation based on function call graph for effective malware detection and classification,

Neural Comput. Appl. 34 (11) (2022) 8643–8656. doi:10.1007/s00521-021-06808-8.

[17] B. Jung, T. Kim, E. G. Im, Malware classification using byte sequence information, in: Proceedings of the 2018 Conference on Research in Adaptive and Convergent Systems, RACS '18, Association for Computing Machinery, New York, NY, USA, 2018, p. 143–148. doi:10.1145/3264746.3264775.

[18] D. Rabadi, S. G. Teo, Advanced windows methods on malware detection and classification, in: Annual Computer Security Applications Conference, ACSAC '20, Association for Computing Machinery, New York, NY, USA, 2020, p. 54–68. doi:10.1145/3427228.3427242.

[19] Aslan, A. A. Yilmaz, A new malware classification framework based on deep learning algorithms, IEEE Access 9 (2021) 87936–87951. doi:10.1109/ACCESS.2021.3089586.

[20] D. Dang, F. Di Troia, M. Stamp, Malware classification using long short-term memory models, arXiv preprint arXiv:2103.02746 (2021).

[21] F. O. Catak, A. F. Yazı, O. Elezaj, J. Ahmed, Deep learning based sequential model for malware analysis using windows exe api calls, PeerJ Computer Science 6 (2020) e285. doi:10.7717/peerj-cs.285.

[22] D. Gibert, C. Mateu, J. Planes, Hydra: A multimodal deep learning framework for malware classification, Computers Security 95 (2020) 101873. doi:https://doi.org/10.1016/j.cose.2020.101873.

[23] I. Abdessadki, S. Lazaar, A new classification based model for malicious pe files detection, International Journal of Computer Network and Information Security (2019).

[24] R. Alanazi, G. Gharibi, Y. Lee, Facilitating program comprehension with call graph multilevel hierarchical abstractions, Journal of Systems and Software 176 (2021) 110945. doi:https://doi.org/10.1016/j.jss.2021.110945.

[25] S. Li, Q. Zhou, R. Zhou, Q. Lv, Intelligent malware detection based on graph convolutional network, The Journal of Supercomputing 78 (02 2022). doi:10.1007/s11227-021-04020-y.

[26] F. Xiao, Z. Lin, Y. Sun, Y. Ma, Malware detection based on deep learning of behavior graphs, Mathematical Problems in Engineering 2019 (2019) 1–10. doi:10.1155/2019/8195395.

[27] A graph-based feature generation approach in android malware detection with machine learning techniques, Mathematical Problems in Engineering, 2020, 1–15 (2020) 1–15doi:10.1155/2020/3842094.