




# Lightweight Malware Classification with FORTUNATE: Precision Meets Computational Efficiency


César Augusto Borges de Andrade   [ University of Brasília | [cesar.andrade@aluno.unb.br](mailto:cesar.andrade@aluno.unb.br) ]

Geraldo Pereira Rocha Filho  [ State University of Southwest Bahia | [geraldo.rocha@uesb.edu.br](mailto:geraldo.rocha@uesb.edu.br) ]

Rodolfo I. Meneguette  [ University of São Paulo | [meneguette@icmc.usp.br](mailto:meneguette@icmc.usp.br) ]

João Paulo Abreu Maranhão  [ Systems Development Center | [joaopaulo.maranhao@eb.mil.br](mailto:joaopaulo.maranhao@eb.mil.br) ]

Ricardo Sant'Ana  [ Military Institute of Engineering | [santana.ricardo@eb.mil.br](mailto:santana.ricardo@eb.mil.br) ]

Julio Cesar Duarte  [ Military Institute of Engineering | [duarte@ime.eb.br](mailto:duarte@ime.eb.br) ]

André Luiz Marques Serrano  [ University of Brasília | [andrelms@unb.br](mailto:andrelms@unb.br) ]

Vinícius P. Gonçalves  [ University of Brasília | [vpvgvinicius@unb.br](mailto:vpvgvinicius@unb.br) ]

 Electrical Engineering Dept., University of Brasília, Campus Universitário Darcy Ribeiro, Brasília, DF, 70910-900, Brazil.

Received: 05 September 2024 • Accepted: 20 February 2025 • Published: 14 April 2025

**Abstract** After detecting a malicious artifact, classifying malware into specific families becomes an essential step to understand the threat's behavior, implement mitigation strategies, and develop proactive defenses. This task is particularly challenging due to the diversity of malware formats, the rapid evolution of obfuscation and packing techniques, as well as the scarcity of labeled data for training robust models. Additionally, the high volume of samples generated daily demands solutions that combine high accuracy and computational efficiency. Although transformer-based models are widely recognized as the state-of-the-art for sequence processing tasks, their high computational demands limit their practical application in resource-constrained environments. In this work, we present FORTUNATE, a lightweight framework that leverages LSTM networks with one-hot encoding to classify malware based on variable-length opcode sequences. The framework adopts an optimized opcode extraction process focused on reducing redundancies and representing data in compact vectors, minimizing computational costs. Experimental results indicate that FORTUNATE achieves accuracies of 99.82% for active malware and 99.81% for inactive malware, with an average classification time of only 56 ms per sample, significantly outperforming related works. The obtained results demonstrate that lightweight artificial intelligence approaches can deliver competitive performance in malware classification, especially in scenarios with computational constraints. FORTUNATE not only fills an important gap in malware classification but also establishes a foundation for future research aimed at optimizing the balance between accuracy, efficiency, and scalability.

**Keywords:** Malware Classification, Opcode, Recurrent Neural Networks (RNNs), Long Short Term Memory (LSTM), Natural Language Processing (NLP), Cybersecurity.

## 1 Introduction

The classification of *malware* is a fundamental task in the field of cybersecurity, aiming to identify and categorize different types of cyber threats, such as viruses, *worms*, Trojan horses, and other forms of malicious software (Taher *et al.*, 2023) and (Vanzan and Duarte, 2023). This process is crucial for the protection of information systems and for mitigating the risks associated with information security (Djenna *et al.*, 2023). In this scenario, where the malware has already been previously detected, a variety of methods and approaches have been proposed in academic literature with the aim of classifying *malware* effectively (Abusitta *et al.*, 2021) and (Li *et al.*, 2023).

Analyzing the static and dynamic characteristics of malicious code represents one of the most common approaches in the classification of *malware*. The static characteristics are obtained from information extracted from the *malware* executable file, including signatures, *strings*, specific sequence *bytes*, and code structure. The dynamic features con-

cern the behavior of *malware* during its execution, covering system calls, network interactions, and file system activities (Sikorski and Honig, 2012; Alraizza and Algarni, 2023; Molina *et al.*, 2022; de Oliveira *et al.*, 2023).

Unlike the dynamic analysis, the static analysis technique employed in the present research does not require the program to run, thus eliminating the risk of activating or rotating *malware* during the analysis, providing a safer environment for analysts. In addition to the issue of security, static analysis has to its advantage the ability to analyze large volumes of code through its ease of automation (Gaber *et al.*, 2024).

Exploring the presented concepts of static and dynamic characteristics analysis of *malware* offers a promising way to improve the detection and classification of cyber threats (Moawad *et al.*, 2024). By incorporating artificial intelligence approaches, such as natural language processing (PLN) and recurring neural networks (RNN), one can develop a more robust and adaptable system. These approaches allow not only a deeper and more comprehensive analysis of *malware* signatures but also the ability to learn and evolve

with new threat variants, thereby ensuring more effective and dynamic cybersecurity (Abid *et al.*, 2023).

Several studies have been conducted to address the growing challenge of malware classification, driven by the continuous increase in cyber threats. Executable files in the PE format allow malicious artifacts to be represented in various ways. The most common approaches in academic works that employ deep learning include:

- Representation as Raw Data Images (Jannat Mim *et al.*, 2024), (El ghabri *et al.*, 2024), (Omar, 2022), (Hebish and Awni, 2024) and (Aslan and Yilmaz, 2021) ;
- API Call Sequence Vectors (Syeda and Asghar, 2024), (Aggarwal and Di Troia, 2024), (Owoh *et al.*, 2024), (Li and Zheng, 2021) and (Catak and Yazi, 2019),; and
- Mnemonic or Executable Code Opcode Vectors (Mehta *et al.*, 2024), (Kale *et al.*, 2023), (Zhao *et al.*, 2021), (Zhang *et al.*, 2019) and (Lu, 2019).

In this work, the use of opcode-based representation was chosen due to the numerous advantages this approach offers. Firstly, opcodes capture the executable code of the malicious artifact, enabling the effective identification of malicious behavior patterns regardless of the programming language used or syntactic details. Furthermore, this representation provides a balanced level of abstraction between binary code and source code, making it less susceptible to syntactic obfuscation techniques that preserve the code's logic. Another advantage is the computational efficiency provided by processing short, standardized opcode sequences, which facilitates the training of deep learning models. Finally, opcode analysis allows the identification of structural similarities between different malware samples and enhances the feature engineering learned by the algorithm, which can be leveraged by malware analysts to obtain more precise insights. Considering these advantages, the architectures proposed in this study employ Recurrent Neural Networks (RNNs), specifically the Long Short-Term Memory (LSTM) variant, due to its ability to efficiently process long sequences and capture complex temporal dependencies present in opcode behavior patterns.

Although many of these studies have successfully achieved their objectives, they present significant limitations, such as difficulties in extracting and encoding *opcodes* Albuquerque *et al.* (2021) and the lack of consideration for their ordering Awad *et al.* (2018). These limitations directly impact the effectiveness of the proposed solutions, as challenges in extraction may lead to incomplete or inaccurate representations of malware characteristics, while disregarding opcode ordering compromises the identification of sequential patterns essential for behavioral analysis. Furthermore, a notable limitation is the reliance on datasets that do not incorporate real and active artifacts Kong *et al.* (2023). This issue reduces the ability of the solutions to generalize to real-world scenarios, thereby hindering their practical applicability. Additionally, another evident limitation is that current research tends to analyze malware in its entirety rather than focusing on specific parts of these artifacts. This approach not only overlooks the potential for a deeper understanding of critical components but also tends to require significantly more computational resources, thereby hindering scalability

and efficiency. This limitation will be addressed in this research by considering strategies to optimize the analysis of specific parts of malware.

This work goes further and proposes FORTUNATE, a framework that employs variable-length instruction sequences to classify malware more efficiently and accurately. For this purpose, FORTUNATE applies PLN and LSTM techniques, specializing in predictive analysis of opcodes in instruction sequences. This approach not only improves the efficiency in classifying malware, but also provides a deeper understanding of the behavior of malware, avoiding data redundancy.

This article presents an extension of the work (Andrade *et al.*, 2024) initially published at the *13th International Conference on Cloud Networking*, where we proposed an innovative framework for malware classification. In this revised and extended version, we significantly improved the classification mechanism and conducted new experiments using an additional dataset, resulting in a more comprehensive and accurate analysis. The content has been enriched with detailed subsections on LightGBM, Active Malware, Inactive Malware, and the Advantages and Limitations of Static Malware Analysis in opcode extraction. Furthermore, we included a comprehensive description of the malware families used in the research, providing a more complete view of the context and contributions of the framework. Furthermore, new comparisons with state-of-the-art works were introduced, covering both active and inactive malware, further highlighting the relevance and effectiveness of the proposed approach.

The main contributions of this research include:

1. the representation of opcode in vectors of smaller dimensions possible to improve computational efficiency;
2. the developing of a method that enables the rapid and accurate classification of *malware* through the analysis of specific fragments of the code; and
3. the creation and availability of a new dataset for future research in the area.

This article is organized as follows. Section 2 presents a brief background on RNNs, LSTMs, Active Malware, and Inactive Malware. Section 3 shows the related jobs, highlighting the search gap that this search investigates. Section 4 presents how FORTUNATE was modeled, while the performance and validation of FORTUNATE in relation literature works are introduced by Section 5 and Section 6. Finally, Section 7 presents the findings and future work.

## 2 Background

In what follows, we provide some background on RNNs, LSTM, LightGBM, Active Malware, Inactive Malware, and Advantages and Limitations of Static Malware Analysis.

### 2.1 Recurrent Neural Networks (RNNs)

Recurrent Neural Networks (RNNs) are a class of artificial neural networks designed to process sequential data, making them particularly effective for tasks that involve ordered or temporal dependencies (Rumelhart *et al.*, 1986). Unlike

traditional feedforward neural networks, where inputs are treated as independent, RNNs have cyclic connections that allow information from previous states to influence the interpretation of subsequent states. This property makes RNNs ideal for a variety of applications, including natural language processing, speech recognition, machine translation, and time series analysis.

The functioning of RNNs is based on maintaining a hidden state that is updated at each time step. The hidden state  $h_t$  at time  $t$  is a function of the current input  $x_t$  and the previous hidden state  $h_{t-1}$ . This mechanism allows the network to have a form of "memory" of past data, accumulating useful information for the task at hand. However, one of the main challenges of traditional RNNs is the problem of vanishing gradients, which hinders the effective training of networks over long sequences. To mitigate these issues, variants such as Long Short-Term Memory (LSTM) networks and Gated Recurrent Units (GRU) were developed. These variants introduce gating mechanisms that regulate the flow of information, allowing networks to capture long-term dependencies more effectively.

## 2.2 Long Short-Term Memory Networks (LSTMs)

Long Short-Term Memory (LSTM) networks are an advanced class of recurrent neural networks (RNNs) developed to address the limitations of traditional RNNs, such as the vanishing gradient problem. Proposed by Hochreiter and Schmidhuber (1997), LSTMs are designed to model and capture long-term dependencies in data sequences. This makes them particularly suitable for tasks involving complex sequential data, such as machine translation, language modeling, speech recognition, and time series forecasting.

The functioning of LSTMs is based on a memory cell and three key gates: the input gate, the forget gate, and the output gate. The input gate controls which new information is added to the memory cell, while the forget gate decides which old information should be discarded. The output gate determines which parts of the memory cell are used to compute the output. These gates are activated by sigmoid and tanh functions, which regulate the flow of information within the cell. This mechanism allows LSTMs to maintain and utilize relevant information over long sequences, mitigating the vanishing gradient problem.

Recently, LSTMs have been widely adopted across various applications due to their ability to handle long-term dependencies. Additionally, current research explores combining LSTMs with other neural network architectures, such as Convolutional Neural Networks (CNNs) and attention mechanisms, to further enhance their performance in complex tasks. Bidirectional LSTMs, which process information in both temporal directions, have also shown promising results. With these innovations, LSTMs continue to play a crucial role in advancing artificial intelligence and developing robust solutions for sequential modeling problems.

## 2.3 LightGBM

The LightGBM (Light Gradient Boosting Machine) is a machine learning framework widely recognized for its efficiency and performance in predictive modeling tasks. Based on the Gradient Boosting Decision Tree (GBDT) technique, it was developed by Microsoft to address classification, regression, and ranking problems in a scalable and precise manner Ke *et al.* (2017). Its distinctive architecture, which includes techniques such as leaf-wise growth and histogram-based learning, enables the construction of highly optimized models, even in scenarios with large-scale datasets and high dimensionality Shi *et al.* (2024). Furthermore, LightGBM natively supports sparse data and categorical variables, reducing the need for extensive preprocessing. These features make the framework a popular choice in both industrial applications and academic research, where computational performance and model robustness are critical requirements.

## 2.4 Active Malware

In this search, *malware* is considered active one characterized by its ability to be executed directly by the operating system, containing all the essential parts, such as headers and sections, that allow its loading and execution. This includes the presence of input points and import tables, crucial to the execution of critical functions in the system, enabling malicious activities to be carried out.

As presented in the following seven families were selected for this research:

**Bifrose:** A backdoor trojan that connects to a remote IP address using TCP port 81 or a random port. It allows an attacker to access the infected computer and perform various actions (Microsoft, 2024b).

**Vundo:** A trojan associated with adware and pop-up infections; often obfuscated to hinder detection (Microsoft, 2024m).

**Zwangi:** Modifies Internet browser settings, alters search engine results, and displays pop-up advertisements (Microsoft, 2024d).

**Koutodoor:** A malware family that changes the Internet Explorer homepage and downloads arbitrary files from specific servers. It can also open certain web pages using Internet Explorer (Microsoft, 2024k).

**Rbot:** A backdoor trojan family that allows attackers to control infected computers. It connects to specific IRC servers and joins channels to receive commands, enabling actions such as spreading to other computers, exploiting Windows vulnerabilities, launching denial-of-service (DoS) attacks, and retrieving system information (Microsoft, 2024c).

**Hupigon:** A backdoor trojan family capable of stealing personal information, such as usernames and passwords, and granting hackers access to and control over the infected PC (Microsoft, 2024i).

**Startpage:** A trojan family that changes the browser homepage without consent. These threats may also perform other malicious actions, with behaviors that can vary widely (Microsoft, 2024f).

## 2.5 Inactive Malware

In the present research, inactive malware refers to malicious software samples that, despite containing characteristics or fragments associated with malicious code, cannot be executed directly on an operating system due to the absence of essential components such as headers or import tables. These malware samples are incapable of initiating or performing autonomous actions within the system environment, being restricted to static analysis or serving as partial representations of malicious samples.

A widely known and publicly accessible repository of inactive malware is the Microsoft Malware Classification Challenge (BIG 2015) (Ronen *et al.*, 2018), a competition organized by Microsoft in collaboration with Kaggle<sup>1</sup> to advance the use of machine learning for malware detection and classification. To ensure safety, the malware samples were deactivated, and their binaries were modified, focusing exclusively on static analysis by providing assembly language (ASM) and bytecode (BYTE) representations of 10,868 samples from 9 malware families. This approach minimized risks associated with handling malware and encouraged the development of practical and secure solutions, establishing BIG 2015 as a benchmark widely used in academic research and cybersecurity practices.

The description of each malware family is presented below:

**Ramnit:** A multifunctional malware capable of behaving as both a worm and a file infector, primarily used for stealing banking credentials Microsoft (2024l).

**Lollipop:** Adware or potentially unwanted program (PUP) that displays advertisements and collects user data (Microsoft, 2024a).

**Kelihos\_ver3:** A botnet variant used for spamming, credential theft, and malware distribution (Microsoft, 2012) and (Arora *et al.*, 2017).

**Vundo:** A Trojan associated with adware and pop-up infections, often obfuscated to evade detection (Microsoft, 2024m).

**Simda:** A family of Trojans that steal passwords and can provide remote access to malicious hackers, allowing full control of the infected computer and data collection (Microsoft, 2024l).

**Tracur:** Malware that redirects users to malicious websites and steals information (Microsoft, 2024g).

**Kelihos\_ver1:** An early version of the Kelihos botnet, with characteristics similar to Kelihos\_ver3 (Microsoft, 2024j).

**Obfuscator.ACY:** A malware family employing advanced obfuscation techniques to evade detection (Microsoft, 2024h).

**Gatak:** A backdoor Trojan used for information theft and remote control of infected machines (Microsoft, 2024e).

## 2.6 Advantages of Static Analysis with Opcode Extraction from Active Malware

Static analysis of active malware focused on opcode extraction offers notable advantages in terms of accuracy and depth

of analysis. The presence of the PE (Portable Executable) header, particularly the "AddressOfEntryPoint" field, enables the precise identification of the malware's execution entry point, allowing opcode extraction to target the most relevant instructions. This facilitates understanding the malware's operational logic and mapping its critical functionalities, such as exploitation routines, persistence, and evasion mechanisms. Additionally, this approach eliminates the need to process redundant or irrelevant information, optimizing the analysis and reducing computational effort.

Another advantage is the ability to perform more robust comparative studies, as active malware retains the structural integrity of the file, enabling the identification of unique patterns associated with specific families. This characteristic is crucial for creating reliable signatures and distinguishing malware from legitimate software with similar structures. The precision provided by opcode extraction in active malware also enhances the effectiveness of machine learning model training, enabling the development of more efficient classifiers capable of better generalizing to new samples.

Finally, static analysis of active malware offers greater flexibility for exploring the impact of obfuscation and packing techniques used by attackers. Since the extracted opcodes directly reflect the instructions contained in the file, analysts can evaluate the extent to which these techniques interfere with the malware's functionality, aiding in the development of specific countermeasures. In summary, static analysis with opcode extraction in active malware strikes a balance between safety, accuracy, and efficiency, making it an indispensable tool for in-depth studies of cyber threats and the enhancement of defense systems.

## 2.7 Limitations of Static Analysis with Opcode Extraction from Malware

Static malware analysis, while powerful and widely used, faces significant limitations when dealing with polymorphic and metamorphic malware. Polymorphic malware modifies its superficial appearance each time it is executed or replicated, employing techniques such as code encryption and obfuscation to hinder detection by signature-based tools (Mauri and Damiani, 2025) and (Mohammed *et al.*, 2025). In this scenario, static analysis, which relies on code inspection without execution, can be easily deceived, as frequent changes in the malware's format or structure make it impractical to create reliable signatures. Furthermore, dependence on decompilers or disassemblers may be insufficient to handle additional layers of obfuscation or compression, limiting precise identification capabilities.

Metamorphic malware presents an even greater challenge, as it can completely rewrite its code while maintaining functionality, removing similarities even between consecutive instances (Gulmez *et al.*, 2024) and (Habib *et al.*, 2024). This continuous evolution of code renders pattern-based approaches ineffective, as no consistent static characteristics exist for identification. Moreover, advanced evasion techniques, such as the introduction of junk code, instruction re-ordering, and control flow alteration, further complicate the static analysis process. This underscores the need to complement static analysis with dynamic or hybrid approaches,

<sup>1</sup><https://www.kaggle.com/c/malware-classification>

which can observe runtime behavior and detect malicious actions regardless of the code's superficial appearance.

### 3 Related Works

The classification of *malware* is an ever-evolving area of research, exploring various approaches and techniques to identify and categorize cyber threats effectively. In this section, the main related works will be analyzed, highlighting their proposals, methodologies adopted, main contributions, best metrics achieved, *malware* sets used and the limitations presented.

In Mehta *et al.* (2024), the authors propose an innovative method for classifying malware, using a hybrid approach that combines *Hidden Markov Model (HMM)* and *Random Forest (RF)*. Initially, HMMs are trained with *opcode* sequences, and the resulting hidden state sequences are used as characteristic vectors, with RF demonstrating the best performance. The main contribution is the application of PLN techniques in the classification of *malware*, where the hidden state sequences of HMMs act as a feature engineering step. The approach achieved an accuracy of 97.58% using the Malicia data set, with 11,688 malware binaries categorized into 7 different families, running in a virtualized environment. Limitations include the need for a large set of data for training HMMs and the possibility that deep learning techniques outperform the proposed methods.

In Kong *et al.* (2023) MalFSM is presented, a feature subset selection method for malware family classification. The methodology involves extracting opcode sequences from disassembled malicious code, selecting key features, and merging them with metadata (file size and line count). This results in a set of 18 selected features. The main contribution is the construction of a lightweight feature set that reduces classification time and space, achieving an accuracy of up to 98.6% on the Microsoft Kaggle malware dataset. The used set consists of inactive malware, and the limitation includes the need to balance complexity and performance, as well as the possibility of losing important features when transforming malware samples into grayscale images for classification.

The study of Albuquerque *et al.* (2021) proposes a method of analyzing *malware* using recurring neural networks, specifically LSTM, to predict *opcodes* of *malwares* and classify them into families. The methodology involves extraction and encoding of *opcodes*, training of LSTM models for prediction, and the use of an artificial neural network for classification. The main contribution is the innovation in using *opcodes* prediction as an entry to the classification of *malware* families, achieving an average accuracy of 92%. However, the work faces difficulties in the extraction and correct encoding of the *opcodes*, which can impact the quality of the input data. Additional adjustments to the hyperparameters are needed to improve accuracy, and the model can benefit from a larger and more varied database for training. Furthermore, the current approach may not be effective in detecting new *malware* families that are not represented in the database used. The authors use the *Microsoft BIG 2015* data set, which does not contain active *malware*.

In the work of Zhao *et al.* (2021), the authors propose

the use of *Gaussian Mixture templates Model-HMMs* (GMM-HMM) for the classification of *malware*, comparing the results with *Hidden Markov Models* (HMM) discrete. The methodology involves the analysis of *opcodes* sequences and entropy as characteristics. The study's main contribution is the demonstration that GMM-HMMs, by modeling continuous data such as *opcodes*, can improve the classification of *malware*, raising the average accuracy, measured by the area under the ROC (AUC) curve, from 73.83% obtained by discrete HMMs to 74.16% achieved by GMMs. The data set used included active samples from three families of *malware*: Winwebsec, Zbot, and Zeroaccess, divided into 80% for training and 20% for testing, with 5-fold cross-validation. The main limitation of the work lies in the complexity and computational cost of training GMM-HMMs, which are more challenging compared to discrete HMMs and the effectiveness depends heavily on the choice of characteristics and modeling parameters, requiring considerable experimentation.

The article of Dang *et al.* (2021) addresses the classification of malware into 20 distinct families using opcode sequences extracted from executables. The study evaluates the performance of various LSTM-based models, including more complex variants with embeddings and bidirectional LSTMs (BiLSTMs). The authors compare five architectures, culminating in a combination of BiLSTM, embeddings, and CNN, which demonstrated the best performance. The most advanced model achieved an average accuracy of 81% when classifying 20 families, while the LSTM without embeddings showed limited results, with an average accuracy of only 55.73% in simpler scenarios (5 families with 9,952 malware samples). This highlights that embeddings are essential for capturing semantic relationships between opcodes and improving the model's generalization. The proposed solution does not handle variable-length opcode sequences, as all sequences are normalized to a fixed length through truncation or padding. Consequently, classification is performed by analyzing only part of the malware. The article emphasizes how integrating NLP techniques, such as embeddings and BiLSTMs, significantly enhances accuracy, while the inclusion of CNN provides a more detailed analysis of the sequences. However, limitations include the reliance on manually tuned hyperparameters and challenges in handling under-represented malware families. The authors suggest exploring additional techniques, such as dimensionality reduction and adaptive segment selection, to address these limitations and enhance the proposed approach.

The study of Sung *et al.* (2020) proposes an innovative method for classifying *malware* in drone control stations using FastText and Bi-LSTM. The methodology involves creating low-dimensional vectors from *opcodes* and API function names, analyzed by a bi-LSTM to increase classification accuracy. The main contribution is the demonstration that FastText and Bi-LSTM can outperform traditional methods of *one-hot encoding*. Using dataset *Microsoft Malware Classification Challenge*, which contains inactive malware from several families, the method achieved an accuracy of 96.76%. Limitations include the need to optimize the size of the input vector and the Bi-LSTM model to balance learning time and cost.

The article of Sun *et al.* (2020) presents an efficient scheme for malware categorization by combining a pre-trained Word2Vec model with Temporal Convolutional Networks (TCN) to enhance malware detection on IoT devices, addressing security challenges in edge computing. Using the Microsoft Malware Classification Challenge (BIG 2015) dataset, the authors leverage Word2Vec to generate compact, low-dimensional representations of malicious file names, overcoming the limitations of one-hot encoding. TCN is employed to model sequences more efficiently than LSTM networks, demonstrating superior speed and reduced memory usage. The proposed approach achieved a final accuracy of 97.5%, surpassing the LSTM-based model, which reached 96.2%. Additionally, the total training time was reduced to 732 seconds in the TCN model, compared to 4712 seconds for LSTM, and the testing time was 16.4 seconds for TCN versus 11.9 seconds for LSTM. This significant reduction in computational cost highlights TCN as an effective and scalable alternative for sequence modeling in security tasks. However, the reliance on the BIG 2015 dataset limits the model's generalization to other types of malware and more diverse IoT scenarios. The study represents a significant advancement in malware categorization, offering a practical and efficient solution for securing smart devices.

In Zhang *et al.* (2019), a hybrid method for detecting malware variants is proposed that integrates different types of features, specifically opcodes and API calls. Opcodes are represented through a bi-gram model, while API calls are represented by a frequency vector. PCA (Principal Component Analysis) is used to optimize these representations and improve convergence speed. Convolutional Neural Networks (CNN) are employed for opcode-based embedding and back-propagation neural networks (BPNN) for API-based embedding. These features are combined, and a detection model is trained using Softmax. The main contribution is the integration of multiple features to enhance the accuracy in malware detection and family classification, achieving over 95% accuracy in malware detection and nearly 90% in family classification. Limitations include the high training time, which exceeds a day, and difficulties if API calls cannot be extracted from certain binaries, whether benign or malicious.

In Lu (2019), an innovative method is proposed for malware classification using LSTM and NLP techniques to analyze opcode sequences. The methodology includes disassembling executable files with IDA Pro, extracting opcodes, converting them into vectors using word embedding, and training a two-layer LSTM model with mean-pooling. The main contribution is the automation of malware pattern learning, reducing the need for manual engineering. The model was evaluated on a dataset with 969 malware samples and 123 benign files, achieving up to 97.87% accuracy for binary classification and 94.51% for multiclass classification. Limitations include the exclusion of operands and a lack of details about execution time.

The study by Awad *et al.* (2018) presents an interesting approach to static malware analysis, treating malware as a natural language. The methodology involves transforming malware executables into malware language documents, using the word2vec model to semantically represent these documents, and the word mover's distance (WMD) algorithm to

measure their proximity. The main contribution is the classification of malware using this semantic approach, achieving an accuracy of up to 98% with cross-validation. The dataset includes 10,868 instances of inactive malware divided into 9 classes, provided by Microsoft. Among the limitations of the work are the high computational cost, especially with the increase in the vocabulary of the malware language, and the disregard for word order by the word2vec model, which can be a disadvantage in certain contexts.

For a comparative analysis between this research and related works, Table 1 was created based on criteria such as whether the used malware set is active or not, whether it works with variable-length instruction sequences, whether it is possible to classify malware from a small part or only from the complete malware, and the number of classes and samples. This research, FORTUNATE, stands out for using a large exclusive dataset composed of a wide variety of active malware. Unlike related works, this research focuses on a restricted set of opcodes, demonstrating its high efficiency and performance. Additionally, methods for extracting opcodes from real and active malware, processing data to minimize redundancies, and representing opcodes by smaller vectors were developed, contributing to accuracy and effectiveness in classifying malware into specific families. A new dataset was also created and made available.

## 4 FORTUNATE: Framework for malware classification Using variable instruction sequence

FORTUNATE can be presented as a new framework that uses variable-length instruction sequences to classify malware efficiently and accurately, with a particular focus on active malware that represents real threats. Using NLP techniques, proper data handling, and LSTM, FORTUNATE specializes in the predictive analysis of opcode sequences. This strategy not only improves the malware classification capability, but also provides valuable insight into the behavior of these digital threats, minimizing the occurrence of unnecessary duplications in the analyzed datasets.

FORTUNATE, illustrated in Figure 1, is organized into three modules: (i) Data Collection, where the acquisition, disassembly, and extraction of malware opcodes occur; (ii) Data Processing, dedicated to data cleaning and transformation; and (iii) Classification Mechanism, which encompasses the training and performance evaluation of the model.

### 4.1 Problem Definition

In this subsection, the problem of malware classification in a multi-class context will be addressed, using an LSTM-based model for the prediction of subsequent opcodes. As presented in Figure 2, the modeling process includes converting opcode sequences (i.e., “push”, “call”, “jnz”, “mov”) into vectors, where identical opcodes are mapped to equivalent vectors. The LSTM model is trained with temporal sequences of  $k = 5$  consecutive opcodes (lookback) to predict the  $k + 1 = 6^{\text{th}}$  opcode, which constitutes the desired out-

**Table 1.** Comparative analysis of related works.

Work	Active	VLIS	CPM	CAD	Classes	# Samples
Mehta <i>et al.</i> (2024)	✓	✗	✓	✗	7	11.688
Kong <i>et al.</i> (2023)	✗	✗	✗	✗	9	10.868
Albuquerque <i>et al.</i> (2021)	✗	✗	✗	✗	9	10.868
Zhao <i>et al.</i> (2021)	✓	✗	✗	✗	3	7.801
Dang <i>et al.</i> (2021)	✓	✗	✓	✗	5	9.952
Sung <i>et al.</i> (2020)	✗	✗	✗	✗	9	10.868
Sun <i>et al.</i> (2020)	✗	✗	✗	✗	9	10.868
Zhang <i>et al.</i> (2019)	✓	✗	✗	✗	5	3.250
Lu (2019)	✓	✓	✗	✗	6	1.092
Awad <i>et al.</i> (2018)	✗	✗	✗	✗	9	10.868
<b>This</b>	✓	✓	✓	✓	7	13.719

<sup>1</sup> VLIS (Variable-Length Instruction Sequences). <sup>2</sup> CPM (Classification from a Small Part of the Malware). <sup>3</sup> CAD (Creation and Availability of a New Dataset).

put. For example, given a malware that presents an opcode sequence from op1 to op6, the input for the model consists of the first five opcodes (op1 to op5), and the sixth opcode (op6) represents the desired output. The value of  $k$  can be adjusted to represent a larger or smaller opcode window, depending on the experiment conducted.

The step of encoding opcode sequences for the intended inputs and outputs is crucial for the effective training of the LSTM model, as illustrated in Figure 3.

This process is replicated for different categories of malware, resulting in the creation of seven distinct LSTM models, each dedicated to a specific malware family. This approach aims to improve the accuracy of opcode predictions, allowing a deeper exploration of the behavior of various malware families.

In this way, it significantly contributes to the improvement of classification and understanding of the threats that these malware represent.

## 4.2 Data Collection

This section presents how the collection, disassembly, and extraction of malware opcodes were performed.

For the selection of samples from malware families, seven VirusShare<sup>2</sup> packages were selected: 00015, 00021, 00023, 00024, 00026, 00047 and 00094. With the exception of the final package, which encompasses 65,536 samples, each of the other packages consists of 131,072 samples, in accordance with the standard prevailing at the time of their release.

Subsequently, a set of acceptance criteria for the selection and labeling of samples was established:

- The artifact was detected by Microsoft's antivirus;
- The artifact was detected by at least 10 other antivirus solutions;
- At least two additional antivirus solutions use terminology similar to the family nomenclature employed by Microsoft's Windows antivirus.

These samples were categorized into twelve distinct malware families to cover a wide spectrum of malicious behaviors.

As seen in Table 2, column # *malware* (a), the criterion used to select malware families was to consider only those with more than 900 and less than 7000 samples, resulting in only seven families.

The compatibility of the artifacts with the executable standard of the Microsoft Windows platform was verified using the PEFile library<sup>3</sup>, a technical choice that ensures the relevance of the samples within the scope of this research.

For the disassembly of the malware, a Python program was developed to automate this process for the malware base. The program calculated the exact location of the first opcode of each malware by using the "AddressOfEntryPoint"<sup>4</sup> header field of the PE (Portable Executable) file. This step is crucial for the effective extraction of data from active malware, differentiating it from approaches that use inactive or inoperative samples.

The effectiveness of the program was corroborated through comparative analyses with the outputs produced by IDA Pro<sup>5</sup>, a leading disassembler in the market, ensuring the reliability of the extracted data.

After disassembly, the focus turns to extracting opcode sequences from the malicious artifacts. Unlike other approaches, this research concentrates exclusively on the opcode sequence, such as "mov", "push", "call", "or" which is extracted and stored for subsequent analyses. This methodological selection allows for a detailed analysis of the malicious code's behavior, grounding the research in concrete data derived directly from the malware samples.

For the validation of our proposal, we used the BIG 2015 dataset. We exclusively utilized ASM files as the data source. To achieve this, we developed a Python script aimed at extracting the opcode sequences present in all code sections of the ASM files. However, the extraction process was constrained by the absence of the file headers in the PE (Portable

<sup>2</sup><https://virusshare.com/>

<sup>3</sup><https://github.com/erocarrera/pefile>

<sup>4</sup>Address of the entry point, where the program execution begins.

<sup>5</sup><https://hex-rays.com/ida-pro/>



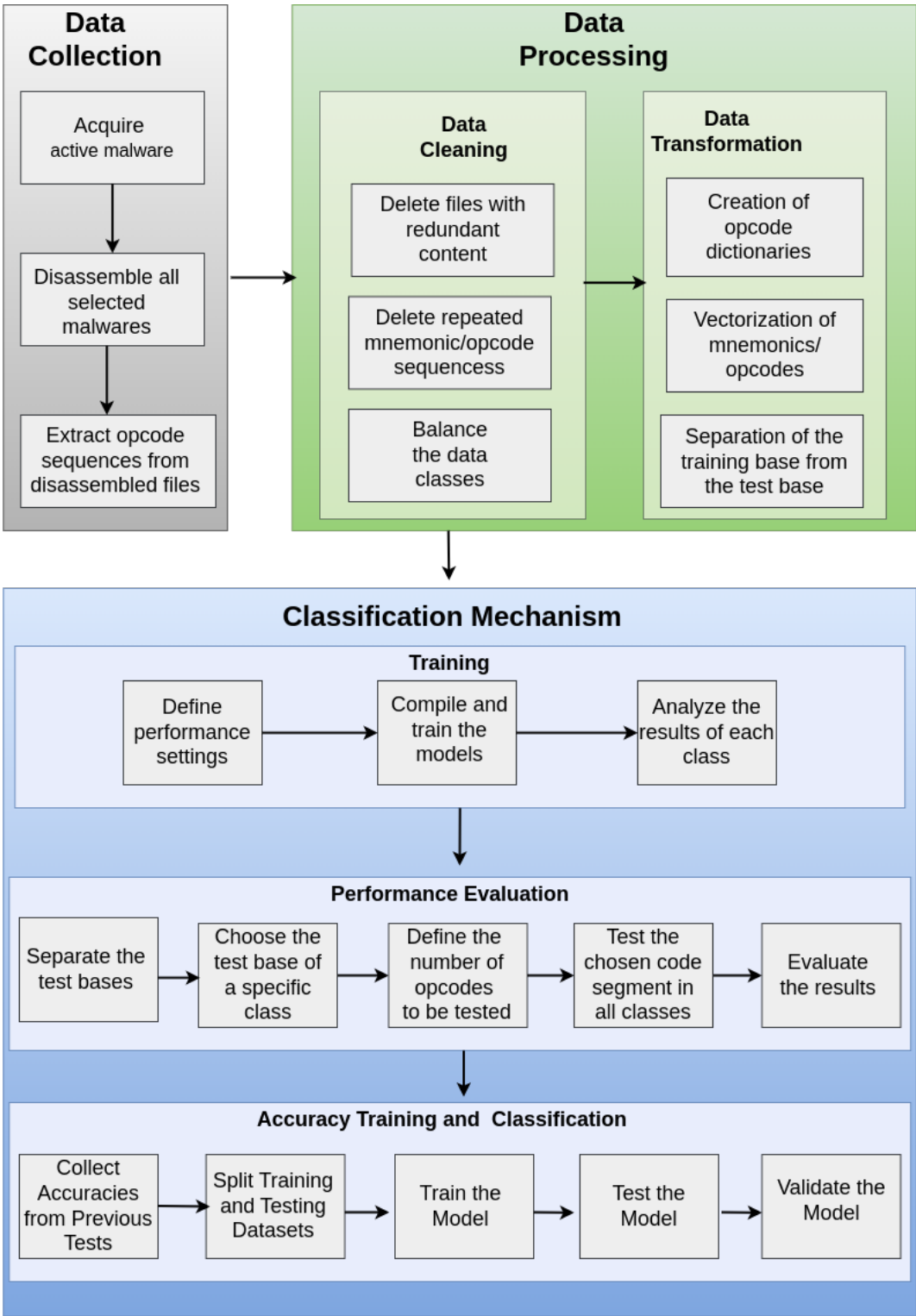


Figure 1. Operational Scenario of FORTUNATE.

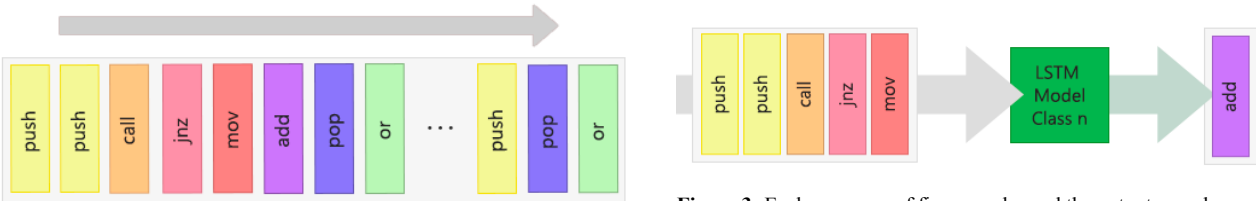


Figure 2. Example of an opcode sequence from a specific artifact.

Figure 3. Each sequence of five opcodes and the output opcode are used in the LSTM training.

Executable) format, which had been previously removed by Microsoft for security reasons. As a result, it was not possi-



ble to determine the exact location of the first opcode, "AddressOfEntryPoint", for each malware. This information is crucial for a more precise and efficient extraction of relevant data associated with the malware's behavior. The number of malware samples per family can be found in Table 3, column # *malware* (a).

### 4.3 Data Processing

The first step in the cleaning process involved identifying and removing duplicate files. The files were organized by MD5 hash, and identified duplicates were automatically deleted. This procedure resulted in a significant reduction in data volume, facilitating subsequent handling and analysis. The effectiveness of this step is demonstrated by the quantitative reduction in files, as presented in Table 2, column # *malware* (b) and in Table 3, column # *malware* (b).

Next, the focus was on eliminating repeated opcode sequences among the samples, aiming to only preserve unique instances for analysis. It is worth noting that automatic filtering was performed, removing redundant mnemonic sequences. This action contributed to a substantial decrease in the number of opcodes per class, as presented in Table 2, column # *opcodes* (b) and in Table 3, column # *opcodes* (b).

To avoid analytical distortions caused by class imbalance in the dataset, all classes were balanced. The number of *opcodes* in each class was adjusted to match the total in the class with the smallest volume. In the first case, involving the analyzed classes, class 3 was used as the reference, as it contained 3,621,441 *opcodes*. This adjustment ensured a fair comparison between classes, as shown in Table 2, column # *opcodes* (c).

For the BIG 2015 dataset, the same criterion was applied, using class 5 as the reference since it had 84,735 *opcodes*. This adjustment also provided an equitable basis for comparison between classes, as detailed in Table 3, column # *opcodes* (c).

It should be noted that, to enable processing of *opcodes* by deep learning algorithms, it was essential to perform a data transformation step, converting them into numerical vectors. To this end, the prevalent opcodes in each class analyzed were identified and cataloged. This activity is fundamental to understanding the distribution and relevance of terms within the data corpus. Using the extracted information, bar charts (Figure 4) and word clouds (Figure 5) were generated for an intuitive analysis of the frequency and importance of the opcodes in each class. In addition, a specific dictionary for each class was developed, where the terms are defined and organized according to their relevance and frequency. This dictionary is of utmost importance as it serves as the basis for subsequent data processing steps.

Following the creation of the dictionaries, the next step involved transforming the *opcodes* into vectors to serve as input for the classification mechanism. For our real malware dataset, the *One Hot Encoder* vectorization technique with dimension 64 was employed, as it offered greater computational efficiency and reduced processing time. During this stage, 80,000 *opcodes* from each malware class were allocated for model training, while 17,000 *opcodes* from each class were reserved for validation. The size of the test set

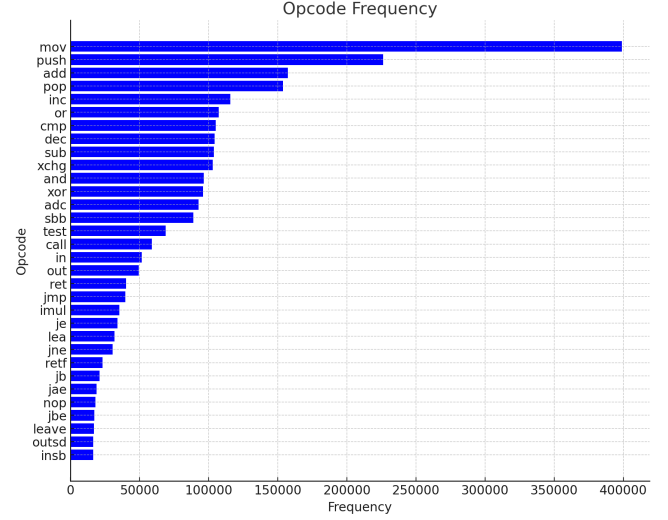


Figure 4. the 32 most frequent opcodes from class 1.

was adjusted according to the specific requirements of each experiment.

For the BIG 2015 dataset, the *One Hot Encoder* vectorization technique with dimension 40 was used, as class/family 5 had only 40 distinct *opcodes*. In this phase, 70,000 *opcodes* from each class were selected for training the model, while 14,000 *opcodes* from each class were used for validation.

### 4.4 Classification Mechanism

This section presents how the malware classification mechanism was modeled. A sequential neural network was used, equipped with four CuDNNLSTM layers, a variation of LSTM layers optimized for GPU processing. This choice was made due to the effectiveness of these layers in analyzing temporal sequences or sequential data, which are essential characteristics for malware classification.

Figure 6 presents the classification process structured in five stages, showing how the data is processed from input to the final malware classification. The first stage of the process (Label 1, Figure 6) involves the reception of malware opcodes, which are specific assembly instructions used by malware during its execution. These opcodes, including commands like "push", "call", "jnz", "mov", "add" and "or" represent the various operations that can be performed by the malware, serving as an initial signature for its identification.

After reception, the assembly instructions are converted into a one-hot encoding format (Label 2, Figure 6). In this format, each opcode is transformed into a binary vector, where only the index corresponding to the specific instruction is marked with "1", while all other indices are filled with "0". This vector representation simplifies the analysis and processing of data by machine learning models, maintaining a uniform and easily interpretable structure.

The encoded vectors are then fed into a series of LSTM models, each designed to recognize a specific malware class (Label 3, Figure 6). The LSTM architecture is particularly suitable for this task, given its ability to process data sequences and identify complex patterns over time. Each LSTM model outputs a probability or confidence level associated with each malware class, identified as "acc 1" to "acc n", indicating the existence of n distinct classes.

**Table 2.** Distribution of the number of malware samples and opcodes by family.

Nr	Family	# <i>malware</i>		# <i>opcodes</i>		
		(a)	(b)	(a)	(b)	(c)
1	Bifrose	2291	1079	42.935.835	28.186.045	3.621.441
2	Vundo	6794	5644	123.161.189	101.995.711	3.621.441
3	Zwangi	920	468	4.910.368	3.621.441	3.621.441
4	Koutodoor	5605	3937	45.849.096	27.896.441	3.621.441
5	Rbot	1170	771	35.864.389	26.273.978	3.621.441
6	Hupigon	1943	1174	115.136.258	79.103.051	3.621.441
7	Startpage	1648	646	49.431.472	30.048.605	3.621.441
Total		20.371	13.719	417.288.607	297.125.272	25.350.087

**Table 3.** Distribution of the number of malware samples and opcodes by family of dataset BIG 2015.

Nr	Family	# <i>malware</i>		# <i>opcodes</i>		
		(a)	(b)	(a)	(b)	(c)
1	Ramnit	1541	1458	86.537.983	54.720.902	84.735
2	Lollipop	2478	2471	690.360.592	44.889.117	84.735
3	Kelihos_ver3	2942	2934	333.158.665	3.371.778	84.735
4	Vundo	475	447	43.505.289	1.109.759	84.735
5	Simda	42	42	3.117.280	84.735	84.735
6	Tracur	751	625	85.342.069	15.243.812	84.735
7	Kelihos_ver1	398	386	7.226.591	1.139.816	84.735
8	Obfuscator.ACY	1228	1195	23.515.457	5.848.263	84.735
9	Gatak	1013	904	290.823.473	5.369.644	84.735
Total		10.868	10.462	1.563.587.399	132.777.826	762.615

Stages four and five of the process represent decision-making phases. In the fourth stage, the *argmax* function is applied to identify the *malware* class with the highest probability among the outputs generated by the LSTM models (Label 4, Figure 6). This procedure provides an initial classification of the *malware*, based on the class with the highest confidence or likelihood of association.

In the fifth stage, a LightGBM classifier is employed to enhance the classification process. At this stage, the model is trained using 80% of the accuracies from the samples of the entire malware dataset and tested on the remaining 20% (Label 5, Figure 6). The primary objective of this phase is to assess the classifier's ability to accurately identify the classes of previously unseen malwares.

## 5 Performance Evaluation

To evaluate FORTUNATE, the experiments were conducted on hardware that includes an Intel® Core™ i7-10700 processor, an NVIDIA® GeForce® RTX™ 3060 graphics card, and 16GB of DDR4 memory. In terms of software, Linux Mint 20.1, Python 3.6.5, tensorflow-gpu 1.8.0, and Keras 2.2.0 were used. The neural network architecture was se-

quential and comprised four CuDNNLSTM layers. Seven LSTM models were trained, each dedicated to a specific class of malicious artifacts, aiming to predict sequences of five opcodes. The Pandas and Numpy libraries were used for data handling, and the Scikit-Learn and TensorFlow libraries were used to implement ML and DL techniques.

To determine the minimum number of opcodes required to correctly classify malware, 182 experiments were conducted, distributed equally among seven malware classes. The number of mnemonics/opcodes tested ranged from 7 to 600 per segment, with a total of 1000 segments analyzed. These segments were evaluated across all classes to determine suitability, thereby providing a detailed understanding of classification effectiveness. Additionally, to define the best neural network hyperparameters, experiments were conducted with various values, including batch sizes of 2, 5, 10, 20, 50, and 100; epochs of 20, 25, 50, and 100; a lookback of 5; units of 64 and 128; loss function defined as mean squared error; and the use of the Adam optimizer. Table 4 presents the description of the parameters used in these experiments.

To evaluate FORTUNATE, the following metrics were used: (i) Precision ( $\text{Precision} = \frac{TP}{TP+FP}$ ) which assesses the accuracy of positive identifications; (ii) Accuracy



**Table 4.** Set of parameters adopted for conducting the experiments in FORTUNATE

Parameter	Description
Total experiments	182
Experiments per malware class/family	18
# of malware classes/families	7
# of segments analyzed	1000
# of mnemonics/opcodes per segment	Varies from 7 to 600
Batch size	2, 5, 10, 20, 50, <b>100</b>
Epochs	20, <b>25</b> , 50, 100
Lookback	5
Units	<b>64 e 128</b>
Loss function	<i>mean squared error</i>
Optimizer	Adam

call; and (v) MCC ( $MCC = \frac{c \times s - \sum_k (p_k \times t_k)}{\sqrt{(s^2 - \sum_k (p_k^2)) \times (s^2 - \sum_k (t_k^2))}}$ ), which considers all categories of predictions. The impact of the achieved results will be presented below.

Initially, the effectiveness of FORTUNATE in detecting different malware families based on opcode sequence analysis was evaluated, as presented in Figure 7. The results demonstrate high effectiveness in detecting seven distinct malware families through opcode sequence analysis. Particularly, the models for Backdoor:Win32/Bifrose and Trojan:Win32/Vundo achieve high precision with few opcodes, while the models for BrowserModifier:Win32/Zwangi and Trojan:Win32/Koutodoor maintain high accuracy regardless of the number of opcodes. It is important to note that there are variations in the precision of identifying other malware, such as Backdoor:Win32/Rbot, Backdoor:Win32/Hupigo, and Trojan:Win32/Startpage. These variations are attributed to the complexity of opcode patterns and their similarity. It is highlighted that approximately 390 opcodes are required to achieve accurate classification, resulting in an average accuracy of 99.7%. This result illustrates FORTUNATE's ability to effectively recognize complex code patterns.

Next, considering only the first 390 opcodes of each malware and from the perspective of the *argmax* function, the correct classification of all samples in our malware dataset into their respective categories was evaluated, as presented in Figure 8. The evaluation of FORTUNATE revealed satisfactory performance across all metrics analyzed, achieving, at best, 99.44% F1-Score and, at worst, 99.23% MCC. These results indicate FORTUNATE's capability to correctly classify instances, effectively balancing precision and recall, and demonstrating its robustness and reliability in detections. Precision varied between 97.60% and 99.91% for all classes, reflecting the model's high capacity to minimize false positives. The overall accuracy of 99.44%, with recall values ranging between 97.01% and 100%, and F1-Scores above 98% for all classes, presents a robust overall performance, highlighting a balance between precision and recall and confirming the model's effectiveness even in the presence of potential class imbalances and a large amount of packaged malware. Table 5 presents the percentage of packed malware samples in each family of our dataset.

For the **LightGBM** model, the parameters used configure various aspects of the model's learning process and structure. The *boosting\_type* parameter defines the type of

**Table 5.** Percentage of packed malware by family.

Family	Packed	Family	Packed
Family 1	39,30%	Family 5	6,36%
Family 2	33,03%	Family 6	35,95%
Family 3	2,14%	Family 7	51,39%
Family 4	66,78%	<b>Total</b>	<b>41,77%</b>

boosting used, with the default being *Gradient Boosting Decision Tree (gbdt)*, which is suitable for classification and regression problems. The *objective* parameter specifies the loss function to be optimized, which is automatically set to *multiclass* for problems with more than two classes. The *learning\_rate* controls the learning rate, balancing the contribution of each tree in the model, while *n\_estimators* defines the total number of trees to be trained. The *num\_leaves* parameter specifies the maximum number of leaves per tree, controlling its complexity, and *max\_depth* sets the maximum depth of the trees, allowing for greater flexibility or limiting *overfitting*. Regularization parameters, such as *reg\_alpha* and *reg\_lambda*, apply L1 and L2 penalties, respectively, to reduce model complexity and avoid *overfitting*. The *subsample* and *colsample\_bytree* parameters determine the proportions of samples and *features* used for each tree, promoting generalization. Additionally, *min\_child\_samples* and *min\_child\_weight* specify the minimum conditions for splits to occur, preventing splits with low representativeness in the data. Finally, *random\_state* ensures result reproducibility when set, while *importance\_type* defines how feature importance is calculated, defaulting to being based on the number of splits. These parameters were tuned to provide a balance between performance and efficiency in the model. Table 6 presents the description of the parameters used in these experiments.

**Table 6.** Set of parameters adopted for conducting the experiments with LightGBM

Parameter	Description
<i>boosting_type</i>	Gradient Boosting Decision Tree ( <i>gbdt</i> )
<i>objective</i>	Automatically inferred (e.g., <i>multiclass</i> )
<i>learning_rate</i>	0.1
<i>n_estimators</i>	100
<i>num_leaves</i>	31
<i>max_depth</i>	-1 (no limit)
<i>min_child_samples</i>	20
<i>min_child_weight</i>	0.001
<i>min_split_gain</i>	0.0
<i>subsample</i>	1.0 (100% of the data)
<i>subsample_for_bin</i>	200000
<i>subsample_freq</i>	0
<i>colsample_bytree</i>	1.0 (100% of the features)
<i>reg_alpha</i>	0.0 (no L1 regularization)
<i>reg_lambda</i>	0.0 (no L2 regularization)
<i>importance_type</i>	Split-based feature importance
<i>random_state</i>	None (not set)

Subsequently, using 80% of the *malware* dataset samples for training and the remaining 20% for testing, the correct classification of *malware* into their respective categories was

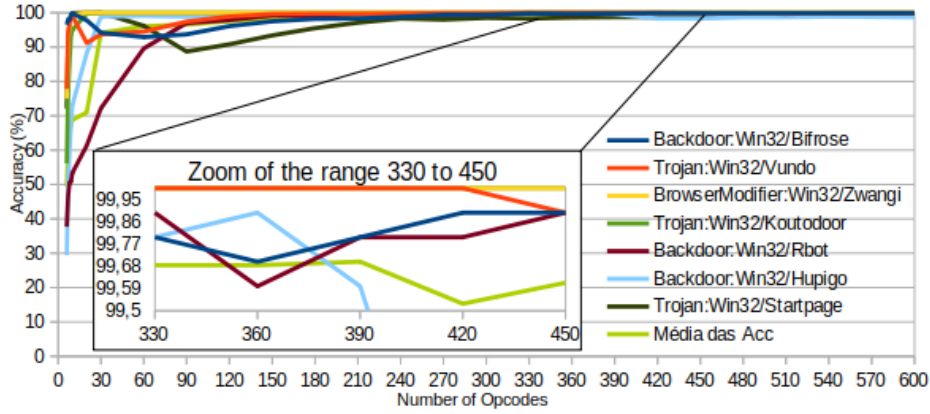


Figure 7. Impact of accuracy in identifying malware segments.

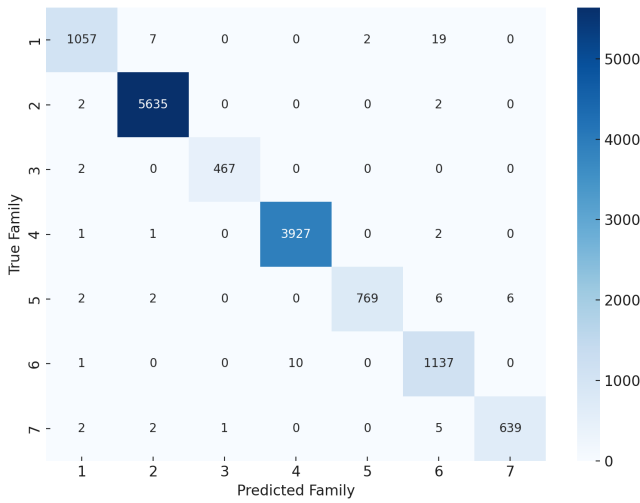


Figure 8. Confusion matrix for malware family classification.

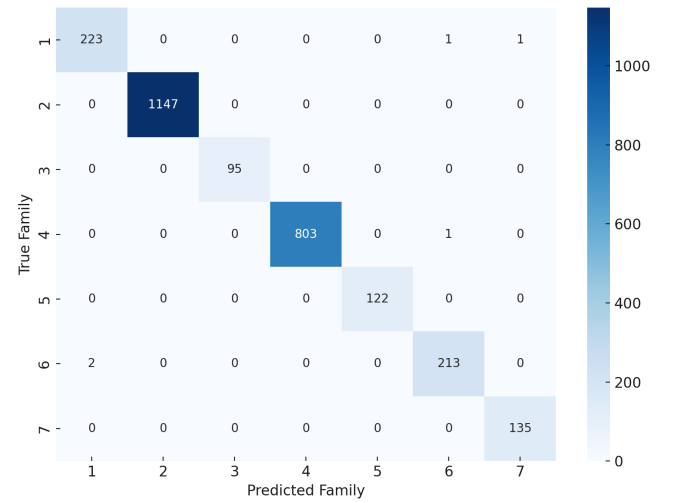


Figure 9. Confusion Matrix LightGBM.

analyzed, as illustrated in Figure 9. The evaluation of FORTUNATE revealed highly satisfactory performance, achieving an accuracy of 99.82%. Other metrics also showed exceptional results, with a precision (macro) of 99.64%, recall (macro) of 99.72%, F1-Score (macro) of 99.68%, and MCC of 99.75%. These results highlight the remarkable capability of FORTUNATE to correctly classify instances, effectively balancing precision and sensitivity, while demonstrating its robustness and reliability. The overall performance confirms the model's effectiveness even in scenarios with potential class imbalances.

Finally, a comparison was conducted between FORTUNATE and five related works that also utilize active malwares, as illustrated in Table 7. The results demonstrate the superior performance of FORTUNATE, achieving accuracies of 99.44% with the argmax function and 99.82% with the LightGBM model, regardless of the solution compared. Among the analyzed studies, Dang *et al.* (2021) stands out as the most comparable approach, given the similar number of samples, classes, and dataset partitioning. However, the LSTM model without embeddings presented by Dang *et al.* achieved only 55.73% accuracy. In comparison, FORTUNATE offers a substantial improvement of 43.71% with the argmax function and 44.09% with the LightGBM model, highlighting its superior efficiency in terms of accuracy.

Table 7. Impact of FORTUNATE compared to the state of the art using active malware.

Work	Year	Accuracy
Mehta <i>et al.</i> (2024)	2024	97,58%
Zhao <i>et al.</i> (2021)	2021	74,16%
Dang <i>et al.</i> (2021)	2021	55,73%
Zhang <i>et al.</i> (2019)	2019	90%
Lu (2019)	2019	94.51%
<b>Argmax function</b>	<b>2024</b>	<b>99,44%</b>
<b>LGBMClassifier</b>	<b>2024</b>	<b>99,82%</b>

## 6 Validation of FORTUNATE with the BIG 2015 dataset

To validate FORTUNATE, the BIG 2015 dataset was used. Considering only the first 390 *opcodes* of each *malware* and from the perspective of the *argmax* function, the correct classification of the *malware* into their respective categories was analyzed, as illustrated in Figure 10. The evaluation of FORTUNATE revealed highly satisfactory performance, achieving an accuracy of 97.80%. Other metrics also showed remarkable results, with a precision (macro) of 97.09%, recall (macro) of 96.49%, F1-Score (macro) of 96.58%, and MCC of 97.34%. These results demonstrate FORTUNATE's ability to correctly classify instances, effectively balancing preci-



sion and sensitivity, while confirming its robustness and reliability in multi-class classification. The overall performance highlights the model's effectiveness even in the presence of potential class imbalances.

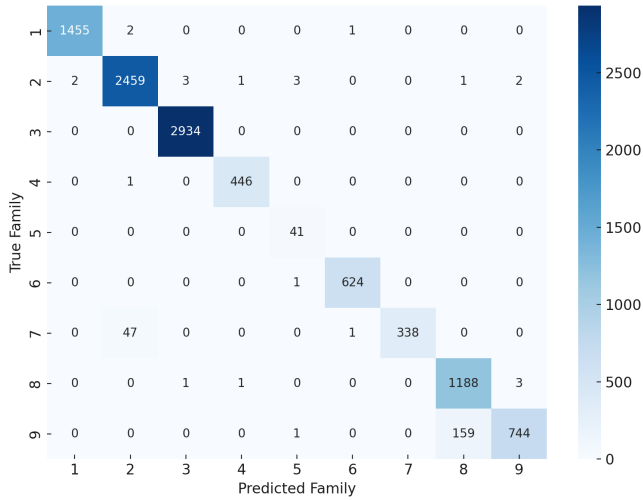


Figure 10. Confusion Matrix BIG.

Considering 80% of the accuracy of the *malware* samples from the BIG 2015 dataset for training and the remaining 20% for testing, and from the perspective of the *LightGBM* classifier, the correct classification of *malware* into their respective categories was analyzed, as illustrated in Figure 9. The evaluation of FORTUNATE demonstrated exceptional performance, achieving an accuracy of 99.81%. Other metrics also showed equally impressive results, with a precision (macro) of 99.69%, recall (macro) of 99.78%, F1-Score (macro) of 99.74%, and an MCC of 99.77%. These results highlight FORTUNATE's ability to correctly classify instances, efficiently balancing precision and sensitivity, while also demonstrating its robustness and reliability. The overall performance reaffirms the model's effectiveness, even in scenarios with potential class imbalances.

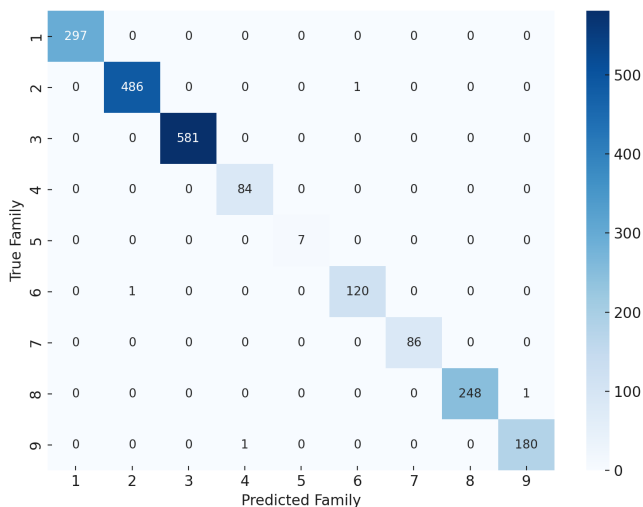


Figure 11. Confusion Matrix BIG LightGBM.

Finally, a comparison of FORTUNATE was conducted with five related works that utilize inactive *malware*, as illustrated in Table 8. The results highlight the superior per-

formance of FORTUNATE, achieving accuracies of 97.80% with the *argmax* function and 99.81% with the *LightGBM* model. Considering the lowest recorded value of 92.00%, attributed to the work in Albuquerque *et al.* (2021), FORTUNATE demonstrates a significant improvement of 5.80% with the *argmax* function and 7.81% with the *LightGBM* model, reinforcing its efficiency and robustness in terms of accuracy for the classification of inactive *malware*.

Table 8. Impact of FORTUNATE compared to the state of the art using inactive *malware*.

Work	Year	Accuracy
Kong <i>et al.</i> (2023)	2023	98,60%
Albuquerque <i>et al.</i> (2021)	2021	92%
Sung <i>et al.</i> (2020)	2020	96,76%
Sun <i>et al.</i> (2020)	2020	96,2%
Awad <i>et al.</i> (2018)	2018	98%
<b>Argumax function</b>	<b>2024</b>	<b>97,80%</b>
<b>LGBMClassifier</b>	<b>2024</b>	<b>99,81%</b>

In the context of the conducted experiment, the use of the BIG 2015 dataset resulted in accuracy metrics, based on the *Argmax* function, slightly lower than those observed in experiments with active *malware* from our own dataset. We believe this outcome is directly related to the absence of headers and critical fields, such as the 'AddressOfEntryPoint', in the inactive *malware* samples from BIG 2015. The lack of these elements hinders the precise identification of entry points and the optimized extraction of opcodes, negatively impacting the analysis and classification.

Regarding execution time, FORTUNATE exhibited a training time of approximately 130 seconds per model, while the classification time per sample was about 56 milliseconds. This result represents a significant improvement in classification time compared to the study by Kong *et al.* (2023), the only related work that reports classification time per sample, which is 7.6 seconds.

The ability of FORTUNATE to train a *malware* family model in just 130 seconds stands out as a competitive advantage, directly addressing the challenges posed by polymorphic and metamorphic *malware*. This speed enables not only frequent updates and rapid responses to emerging threats but also facilitates the deployment of scalable, specialized, and cost-effective solutions. When integrated with a continuous update pipeline, this efficiency establishes FORTUNATE as a robust and adaptable tool, essential for defense in an ever-evolving cybersecurity landscape.

This result can be attributed to the combination of three factors: the use of NLP, proper data handling, and the application of LSTM, which together make FORTUNATE effective in *malware* classification.

## 7 Conclusions and Future Work

Despite transformers being widely recognized as the state-of-the-art in sequence processing tasks, this work opted for a model based on LSTM with one-hot encoding. Our goal was to investigate a lighter and more accessible approach, suit-

able for scenarios with computational constraints. Models such as BERT, GPT, RoBERTa, XLNet, and T5 demonstrate excellent performance in complex natural language processing and classification tasks. However, these architectures often require significant computational resources for training and inference, which may limit their practical application in systems with hardware restrictions.

In this context, our results show that simpler methods, such as LSTM with one-hot encoding, can still achieve competitive performance in malware classification through opcodes, particularly in specific scenarios. This observation paves the way for future research to compare these approaches with transformer-based models, investigating differences in performance, efficiency, and generalization to identify the most suitable solution for each context.

In this research, we presented **FORTUNATE**, a framework that utilizes variable-length instruction sequences to classify malware efficiently and accurately. FORTUNATE excelled in predictive analysis of opcode sequences, demonstrating high effectiveness in identifying cyber threats. In addition to optimizing classification, FORTUNATE also provided relevant insights into the behavior of these threats.

The results obtained demonstrate that FORTUNATE achieves satisfactory performance in classifying malware distributed across  $n$  distinct families. Metrics such as precision, accuracy, recall, F1-Score, and MCC confirm its ability to precisely identify different malware classes. The framework also demonstrated an effective balance by reducing false positives while maintaining high efficiency in detecting true positives—an essential characteristic in the field of cybersecurity.

Additionally, FORTUNATE's performance in classifying active and inactive malware was highlighted, achieving accuracies of 99.82% and 99.81%, respectively. Furthermore, the average classification time per sample using FORTUNATE was approximately 56 ms, demonstrating its high efficiency in terms of speed.

For future directions, we propose expanding FORTUNATE through additional tests using different recurrent neural networks, exploring new opcode encoding methodologies, and applying the techniques to various malware datasets. These initiatives aim not only to enhance FORTUNATE's effectiveness but also to adapt it to a wide variety of neural network architectures and encoding techniques, further promoting its versatility and efficiency across different scenarios.

## Acknowledgements

The authors thank the University of Brasília (UnB), the LATITUDE Laboratory, and the research groups Projectum and Lincex for their institutional and technical support. This work was also supported by the Research Support Foundation of the Federal District (FAPDF), through the Grant Agreement No. 550/2024 — Call 06/2024 — FAPDF Learning Program, and by the Coordination for the Improvement of Higher Education Personnel (CAPES).

## Authors' Contributions

The authors equally contributed to the elaboration of this paper. All

the authors read and approved the final manuscript.

## Competing interests

The authors declare that they have no competing interests.

## Availability of data and materials

The datasets generated and analyzed during the current study are available in SMARTNESS (*dataSet Malware fRom The wiNdows opErating SyStem*) repository, <https://github.com/CABorges72/SMARTNESS>.

## References

- Abid, Y. A., Wu, J., Farhan, M., and Ahmad, T. (2023). ECMT Framework for Internet of Things: An Integrative Approach Employing In-Memory Attribute Examination and Sophisticated Neural Network Architectures in Conjunction With Hybridized Machine Learning Methodologies. *IEEE Internet of Things Journal*, pages 1–1. Conference Name: IEEE Internet of Things Journal. DOI: 10.1109/JIOT.2023.3312152.
- Abusitta, A., Li, M. Q., and Fung, B. C. (2021). Malware classification and composition analysis: A survey of recent developments. *Journal of Information Security and Applications*, 59:102828. DOI: <https://doi.org/10.1016/j.jisa.2021.102828>.
- Aggarwal, S. and Di Troia, F. (2024). Malware classification using dynamically extracted api call embeddings. *Applied Sciences*, 14(13). DOI: 10.3390/app14135731.
- Albuquerque, D. G. d., Vieira, L. d. Q., Sant'Ana, R., and Duarte, J. C. (2021). Análise de comportamento de malware utilizando redes neurais recorrentes - uma abordagem por intermédio da previsão de opcodes. *Revista Militar de Ciência e Tecnologia*, 37(3). Available at: <http://ebrevistas.eb.mil.br/CT/article/view/6914>.
- Alraizza, A. and Algarni, A. (2023). Ransomware Detection Using Machine Learning: A Survey. *Big Data and Cognitive Computing*, 7(3):143. Number: 3 Publisher: Multidisciplinary Digital Publishing Institute. DOI: 10.3390/bdcc7030143.
- Andrade, C. A. B., Rocha Filho, G. P., Meneguetto, R. I., Maranhão, J. P. A., Sant'Ana, R., Duarte, J. C., Serrano, A. L. M., and Gonçalves, V. P. (2024). Fortunate: Decrypting and classifying malware by variable length instruction sequences. In *2024 IEEE 13th International Conference on Cloud Networking (CloudNet)*, pages 1–9. DOI: 10.1109/CloudNet62863.2024.10815801.
- Arora, A., Gannon, M., and Warner, G. (2017). Kelihos botnet: A never-ending saga. Available at: <https://commons.erau.edu/cgi/viewcontent.cgi?article=1271&context=adfs1>.
- Aslan, Ö. and Yilmaz, A. A. (2021). A new malware classification framework based on deep learning algorithms. *IEEE Access*, 9:87936–87951. DOI: 10.1109/ACCESS.2021.3089586.
- Awad, Y., Nassar, M., and Safa, H. (2018). Modeling Malware as a Language. In *2018 IEEE International Confer-*



- ence on Communications (ICC), pages 1–6. ISSN: 1938-1883. DOI: 10.1109/ICC.2018.8422083.
- Catak, F. O. and Yazici, A. F. (2019). A benchmark api call dataset for windows pe malware classification. *ArXiv*, abs/1905.01999. Available at: <https://api.semanticscholar.org/CorpusID:146120927>.
- Dang, D., Troia, F. D., and Stamp, M. (2021). Malware classification using long short-term memory models. DOI: 10.5220/0010378007430752.
- de Oliveira, J. A., Gonçalves, V. P., Meneguette, R. I., de Sousa Jr, R. T., Guidoni, D. L., Oliveira, J. C., and Rocha Filho, G. P. (2023). F-nids—a network intrusion detection system based on federated learning. *Computer Networks*, 236:110010. DOI: 10.1016/j.comnet.2023.110010.
- Djenna, A., Bouridane, A., Rubab, S., and Marou, I. M. (2023). Artificial Intelligence-Based Malware Detection, Analysis, and Mitigation. *Symmetry*, 15(3):677. Number: 3 Publisher: Multidisciplinary Digital Publishing Institute. DOI: 10.3390/sym15030677.
- El ghabri, N., Belmekki, E., and Bellafkih, M. (2024). Pre-trained deep learning models for malware image based classification and detection. In *2024 Sixth International Conference on Intelligent Computing in Data Sciences (ICDS)*, pages 1–7. DOI: 10.1109/ICDS62089.2024.10756501.
- Gaber, M. G., Ahmed, M., and Janicke, H. (2024). Malware Detection with Artificial Intelligence: A Systematic Literature Review. *ACM Comput. Surv.*, 56(6):148:1–148:33. DOI: 10.1145/3638552.
- Gulmez, S., Kakisim, A. G., and Sogukpinar, I. (2024). Analysis of the zero-day detection of metamorphic malware. In *2024 9th International Conference on Computer Science and Engineering (UBMK)*, pages 1–6. DOI: 10.1109/UBMK63289.2024.10773421.
- Habib, F., Shirazi, S. H., Aurangzeb, K., Khan, A., Bhushan, B., and Alhussein, M. (2024). Deep neural networks for enhanced security: Detecting metamorphic malware in iot devices. *IEEE Access*, 12:48570–48582. DOI: 10.1109/ACCESS.2024.3383831.
- Hebish, M. W. and Awni, M. (2024). Cnn-based malware family classification and evaluation. In *2024 14th International Conference on Electrical Engineering (ICEENG)*, pages 219–224. DOI: 10.1109/ICEENG58856.2024.10566448.
- Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural Comput.*, 9(8):1735–1780. DOI: 10.1162/neco.1997.9.8.1735.
- Jannat Mim, M. M., Nela, N. A., Das, T. R., Rahman, M. S., and Ahmed Shibly, M. M. (2024). Enhancing malware detection through convolutional neural networks and explainable ai. In *2024 IEEE Region 10 Symposium (TENSYMP)*, pages 1–6. DOI: 10.1109/TENSYMP61132.2024.10752108.
- Kale, A. S., Pandya, V., Di Troia, F., and Stamp, M. (2023). Malware classification with Word2Vec, HMM2Vec, BERT, and ELMo. *J Comput Virol Hack Tech*, 19(1):1–16. DOI: 10.1007/s11416-022-00424-3.
- Ke, G., Meng, Q., Finley, T., Wang, T., Chen, W., Ma, W., Ye, Q., and Liu, T.-Y. (2017). Lightgbm: a highly efficient gradient boosting decision tree. In *Proceedings of the 31st International Conference on Neural Information Processing Systems, NIPS’17*, page 3149–3157, Red Hook, NY, USA. Curran Associates Inc. Available at: [https://proceedings.neurips.cc/paper\\_files/paper/2017/file/6449f44a102fde848669bdd9eb6b76fa-Paper.pdf](https://proceedings.neurips.cc/paper_files/paper/2017/file/6449f44a102fde848669bdd9eb6b76fa-Paper.pdf).
- Kong, Z., Xue, J., Wang, Y., Zhang, Q., Han, W., and Zhu, Y. (2023). Malfsm: Feature subset selection method for malware family classification. *Chinese Journal of Electronics*, 32(1):26–38. DOI: 10.23919/cje.2022.00.038.
- Li, C. and Zheng, J. (2021). Api call-based malware classification using recurrent neural networks. *Journal of Cyber Security and Mobility*. DOI: 10.13052/jcsm2245-1439.1036.
- Li, Z., Liu, H., Shan, R., Sun, Y., Jiang, Y., and Hu, N. (2023). Binary code similarity detection: State and future. In *12th IEEE International Conference on Cloud Networking, CloudNet 2023, Hoboken, NJ, USA, November 1-3, 2023*, pages 408–412. IEEE. DOI: 10.1109/CLOUD-NET59005.2023.10490019.
- Lu, R. (2019). Malware detection with lstm using opcode language. Available at: <https://arxiv.org/abs/1906.04593>.
- Mauri, L. and Damiani, E. (2025). Hardening behavioral classifiers against polymorphic malware: An ensemble approach based on minority report. *Information Sciences*, 689:121499. DOI: <https://doi.org/10.1016/j.ins.2024.121499>.
- Mehta, R., Jurečková, O., and Stamp, M. (2024). A natural language processing approach to Malware classification. *J Comput Virol Hack Tech*, 20(1):173–184. DOI: 10.1007/s11416-023-00506-w.
- Microsoft (2012). Update on kelihos botnet and new related malware. Available at: <https://blogs.microsoft.com/blog/2012/02/03/update-on-kelihos-botnet-and-new-related-malware/> Accessed: 2024-12-09.
- Microsoft (2024a). Adware:win32/lollipop - malware encyclopedia. Available at: <https://www.microsoft.com/en-us/wdsi/threats/malware-encyclopedia-description?Name=Adware:Win32/Lollipop> Accessed: 2024-12-09.
- Microsoft (2024b). Backdoor:win32/bifrose - malware encyclopedia. Available at: <https://www.microsoft.com/en-us/wdsi/threats/malware-encyclopedia-description?Name=Backdoor%3AWin32%2FBifrose> Accessed: 2024-12-09.
- Microsoft (2024c). Backdoor:win32/rbot - malware encyclopedia. Available at: <https://www.microsoft.com/en-us/wdsi/threats/malware-encyclopedia-description?Name=Backdoor:Win32/Rbot> Accessed: 2024-12-09.
- Microsoft (2024d). Browsermodifier:win32/zwangi - malware encyclopedia. Available at: <https://www.microsoft.com/en-us/wdsi/threats/malware-encyclopedia-description?Name=BrowserModifier%3AWin32%2FZwangi> Accessed:

- 2024-12-09.
- Microsoft (2024e). Trojan:win32/gatak - malware encyclopedia. Available at: <https://www.microsoft.com/en-us/wdsi/threats/malware-encyclopedia-description?Name=Trojan%3AWin32%2FGatak> Accessed: 2024-12-09.
- Microsoft (2024f). Trojan:win32/startpage - malware encyclopedia. Available at : <https://www.microsoft.com/en-us/wdsi/threats/malware-encyclopedia-description?Name=Trojan:Win32/Startpage&threatId=15435> Accessed: 2024-12-09.
- Microsoft (2024g). Trojan:win32/tracur.b - malware encyclopedia. Available at: <https://www.microsoft.com/en-us/wdsi/threats/malware-encyclopedia-description?Name=Trojan:Win32/Tracur.B&threatId=-2147311368> Accessed: 2024-12-09.
- Microsoft (2024h). Virtool:win32/obfuscator.acy - malware encyclopedia. Available at: <https://www.microsoft.com/en-us/wdsi/threats/malware-encyclopedia-description?Name=VirTool:Win32/Obfuscator.ACY> Accessed: 2024-12-09.
- Microsoft (2024i). Win32/hupigon - malware encyclopedia. Available at: <https://www.microsoft.com/en-us/wdsi/threats/malware-encyclopedia-description?Name=Win32/Hupigon> Accessed: 2024-12-09.
- Microsoft (2024j). Win32/kelihos - malware encyclopedia. Available at: <https://www.microsoft.com/en-us/wdsi/threats/malware-encyclopedia-description?Name=Win32/Kelihos> Accessed: 2024-12-09.
- Microsoft (2024k). Win32/koutodoor - malware encyclopedia. Available at <https://www.microsoft.com/en-us/wdsi/threats/malware-encyclopedia-description?Name=Win32/Koutodoor> Accessed: 2024-12-09.
- Microsoft (2024l). Win32/ramnit - malware encyclopedia. Available at: <https://www.microsoft.com/en-us/wdsi/threats/malware-encyclopedia-description?Name=Win32/Ramnit> Accessed: 2024-12-09.
- Microsoft (2024m). Win32/vundo - malware encyclopedia. Available at: <https://www.microsoft.com/en-us/wdsi/threats/malware-encyclopedia-description?Name=Win32%2FVundo> Accessed: 2024-12-09.
- Moawad, A., Ebada, A. I., El-Harby, A., and Al-Zoghby, A. M. (2024). *An Automatic Artificial Intelligence System for Malware Detection*, chapter 6, pages 115–138. John Wiley & Sons, Ltd. DOI: <https://doi.org/10.1002/9781394213948.ch6>.
- Mohammed, M., Abdalla, M., and Elhoseny, M. (2025). Detecting zero-day polymorphic worms using honeywall. *Journal of Cybersecurity and Information Management*, pages 34–49. DOI: 10.54216/JCIM.150104.
- Molina, A. L., Gonçalves, V. P., De Sousa, R. T., Pividal, M., Meneguette, R. I., and Rocha Filho, G. P. (2022). A lightweight unsupervised learning architecture to enhance user behavior anomaly detection. In *2022 IEEE Latin-American Conference on Communications (LATINCOM)*, pages 1–6. IEEE. DOI: 10.1109/latincom56090.2022.10000477.
- Omar, M. (2022). *New Approach to Malware Detection Using Optimized Convolutional Neural Network*, pages 13–35. Springer International Publishing, Cham. DOI: 10.1007/978-3-031-15893-3\_2.
- Owoh, N., Adejoh, J., Hosseinzadeh, S., Ashawa, M., Osamor, J., and Qureshi, A. (2024). Malware detection based on api call sequence analysis: A gated recurrent unit-generative adversarial network model approach. *Future Internet*, 16(10). DOI: 10.3390/fi16100369.
- Ronen, R., Radu, M., Feuerstein, C., Yom-Tov, E., and Ahmadi, M. (2018). Microsoft malware classification challenge. Available at: <https://arxiv.org/abs/1802.10135>.
- Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986). Learning representations by back-propagating errors. *Nature*, 323:533–536. Available at: <https://api.semanticscholar.org/CorpusID:205001834>.
- Shi, Y., Ke, G., Chen, Z., Zheng, S., and Liu, T.-Y. (2024). Quantized training of gradient boosting decision trees. In *Proceedings of the 36th International Conference on Neural Information Processing Systems, NIPS '22*, Red Hook, NY, USA. Curran Associates Inc.. DOI: <https://doi.org/10.48550/arXiv.2207.09682> 10.48550/arXiv.2207.09682.
- Sikorski, M. and Honig, A. (2012). *Practical Malware Analysis: The Hands-On Guide to Dissecting Malicious Software*. No Starch Press, USA, 1st edition. Book.
- Sun, J., Luo, X., Gao, H., Wang, W., Gao, Y., and Yang, X. (2020). Categorizing malware via a word2vec-based temporal convolutional network scheme. *Journal of Cloud Computing*, 9. DOI: 10.1186/s13677-020-00200-y.
- Sung, Y., Jang, S., Jeong, Y.-S., and Park, J. H. J. J. (2020). Malware classification algorithm using advanced Word2vec-based Bi-LSTM for ground control stations. *Computer Communications*, 153:342–348. DOI: <https://doi.org/10.1016/j.comcom.2020.02.005>.
- Syeda, D. Z. and Asghar, M. N. (2024). Dynamic malware classification and api categorisation of windows portable executable files using machine learning. *Applied Sciences*, 14(3). DOI: 10.3390/app14031015.
- Taher, F., AlFandi, O., Al-kfairy, M., Al Hamadi, H., and Alrabae, S. (2023). DroidDetectMW: A Hybrid Intelligent Model for Android Malware Detection. *Applied Sciences*, 13(13):7720. Number: 13 Publisher: Multidisciplinary Digital Publishing Institute. DOI: 10.3390/app13137720.
- Vanzan, M. and Duarte, J. (2023). Malware classification using transfer learning through the gpt-2 model. In *Anais do XXIII Simpósio Brasileiro de Segurança da Informação e de Sistemas Computacionais*, pages 167–180, Porto Alegre, RS, Brasil. SBC. DOI: 10.5753/sbseg.2023.233086.
- Zhang, J., Qin, Z., Yin, H., Ou, L., and Zhang, K. (2019). A feature-hybrid malware variants detection using cnn based opcode embedding and bpnn based api

embedding. *Computers & Security*, 84:376–392. DOI: <https://doi.org/10.1016/j.cose.2019.04.005>.

Zhao, J., Basole, S., and Stamp, M. (2021). Malware Classification with GMM-HMM Models. arXiv:2103.02753 [cs, stat]. DOI: 10.48550/arXiv.2103.02753.