FEDERAL UNIVERSITY OF UBERLÂNDIA

Guilherme de Sousa Pereira

A Hybrid CNN-LSTM Architecture for Malware Behavioral Analysis via API Call Sequences

Uberlândia, Brazil

2026

FEDERAL UNIVERSITY OF UBERLÂNDIA

Guilherme de Sousa Pereira

A Hybrid CNN-LSTM Architecture for Malware Behavioral Analysis via API Call Sequences

Undergraduate thesis submitted to the Faculty of Computing of the Federal University of Uberlandia in partial fulfillment of the requirements for the degree of Bachelor of Information Systems.

Supervisor: Daniel Duarte Abdala

Federal University of Uberlândia – UFU

Faculty of Computing

Bachelor of Information Systems

Uberlândia, Brazil

2026

Guilherme de Sousa Pereira

A Hybrid CNN-LSTM Architecture for Malware Behavioral Analysis via API Call Sequences

Undergraduate thesis submitted to the Faculty of Computing of the Federal University of Uberlandia in partial fulfillment of the requirements for the degree of Bachelor of Information Systems.

Thesis approved. Uberlândia, Brazil, May 6th, 2026:

Daniel Duarte Abdala
Supervisor

Uberlândia, Brazil
2026

Acknowledgments

This work is dedicated to my grandmother Manoelina (*in memoriam*), who is in heaven with God. She was, still is, and will forever be my primary source of strength and resilience, and I hope she is very proud of me from where she is.

First, I would like to thank my mother, Andreia, and my father, Wilton. They are responsible for giving me the greatest gift parents can provide to their child: an excellent education, both inside and outside of school. I will be forever grateful for the sacrifices you made for my well-being throughout my life.

I also want to thank the best group of friends anyone could ever ask for, who have been by my side for basically my entire life, standing with me through both great and extremely hard times. To the Brotherhood, my cousins Jhonatan and Ayrton, my neighbor Lucas, my childhood friends Milena, Velasco, and Patrick, and all the other friends who shared their time, smiles, fun, and presence—and mainly, who endured my annoyance for all these years—thank you all for accepting me into your lives.

I must also thank my family, who have always supported me. Even with our flaws (as every family has them), you took care of me and were always there to teach, correct, and guide me. To my aunt Ana Paula, my uncle Valterlei, my cousin Ana Carla, my cousin Carlos, my uncle Jean, and my aunt Simone: my most sincere thanks to all of you, because without you, this path would not have been possible.

Thank you to my advisor Daniel Abdala, for believing in me and my crazy idea to write this thesis entirely in English (because apparently, doing a graduation thesis alone in my native language was not difficult enough). Despite the bumpy roads, you did a phenomenal job guiding me to deliver this work. When you said, "I know I am annoying, but I will help you write a work that you will be very proud of,"you were completely right. Even though this took an enormous amount of dedication to complete, I am indeed incredibly proud of the result.

Lastly, I would like to thank and praise God for giving me an immeasurable amount of strength and countless signs not to give up. He knows I had thoughts of quitting many times, but somehow, I kept moving forward. What once seemed like a very distant dream—that I would finally graduate—now feels like it is just a matter of time.

Abstract

The detection and classification of malicious software through the behavioral analysis of Application Programming Interface (API) calls remains a prominent cybersecurity challenge. However, the structural complexity of modern threats and the immense semantic noise embedded within long execution sequences often hinder traditional predictive models. Purely recurrent neural networks, such as baseline Long Short-Term Memory (LSTM) architectures, frequently suffer from contextual gradient saturation and severe cross-misclassification when processing these unfiltered sequences, particularly in datasets exhibiting inherent structural asymmetry. To address this representational deficit, this research proposes and empirically evaluates a hybrid deep learning architecture (CNN-LSTM) for multiclass malware categorization. The proposed model integrates 1DConvolutional Neural Networks (CNN) as a preliminary spatial filtering phase to extract localized behavioral signatures (*n*-grams), followed by a Bidirectional LSTM (Bi-LSTM) to map the temporal narrative of the execution. Furthermore, algorithmic class weighting is applied to optimize learning across highly complex,

underrepresented families. Experimental evaluation against a strictly replicated single-layer LSTM baseline demonstrates the architectural superiority of the hybrid approach. The CNN-LSTM achieved a global accuracy of 67.1%—surpassing the 61.0% baseline ceiling—while drastically reducing the computational training time from 407 minutes to merely 12.4 minutes. By effectively isolating semantic noise, the spatial feature extraction yielded substantial relative F1-Score improvements in complex families, such as Dropper (+31.9%) and Virus (+21.9%). These empirical findings definitively validate that spatial dimensionality reduction prior to temporal inference prevents memory saturation, establishing the CNN-LSTM hybridization as a highly viable, computationally efficient, and robust paradigm for real-world threat classification.

Keywords: Deep Learning, Malware Classification, CNN-LSTM, API Calls, Cybersecurity, Behavioral Analysis.

Resumo

A detecção e classificação de *softwares* maliciosos através da análise comportamental de chamadas de Interface de Programação de Aplicações (API) permanece um desafio proeminente na cibersegurança. No entanto, a complexidade estrutural das ameaças modernas e o imenso ruído semântico embutido em longas sequências de execução frequentemente dificultam a precisão dos modelos preditivos tradicionais. Redes neurais puramente recorrentes, como as arquiteturas base *Long Short-Term Memory* (LSTM), frequentemente sofrem com a saturação do gradiente contextual e severos erros de classificação cruzada ao processar essas sequências não filtradas, particularmente em conjuntos de dados que exibem assimetria estrutural inerente. Para solucionar esse déficit representacional, esta pesquisa propõe e avalia empiricamente uma arquitetura híbrida de aprendizado profundo (CNN-LSTM) para a categorização multiclasse de *malwares*. O modelo proposto integra Redes Neurais Convolucionais 1D (CNN) como uma fase preliminar de filtragem espacial para extrair assinaturas comportamentais localizadas (*n*-gramas), seguidas por um bloco LSTM Bidirecional (Bi-LSTM) para mapear a narrativa temporal da execução. Além disso, uma penalização algorítmica de pesos de classe é aplicada para otimizar o aprendizado em famílias sub-representadas e altamente complexas. A avaliação experimental, comparada a um modelo de referência LSTM de camada única rigorosamente replicado, demonstra a superioridade arquitetural da abordagem híbrida. A CNN-LSTM alcançou uma acurácia global de 67,1% — superando o teto de 61,0% da linha de base — enquanto reduziu drasticamente o tempo de treinamento computacional de 407 minutos para apenas 12,4 minutos. Ao isolar efetivamente o ruído semântico, a extração de características espaciais

gerou melhorias relativas substanciais no F1-Score em famílias complexas, como Dropper (+31,9%) e Vírus (+21,9%). Estas descobertas empíricas validam de forma definitiva que a redução da dimensionalidade espacial prévia à inferência temporal previne a saturação da memória, estabelecendo a hibridização CNN-LSTM como um paradigma altamente viável, computacionalmente eficiente e robusto para a classificação de ameaças em cenários reais.

Palavras-chave: Aprendizado Profundo, Classificação de *Malware*, CNN-LSTM, Chamadas de API, Cibersegurança, Análise Comportamental.

List of Figures

List of Tables

List of Abbreviations and Acronyms

API	Application Programming Interface
UFU	Federal University of Uberlândia
LSTM	Long-Short Term Memory
CNN	Convolutional Neural Network
RNN	Recurrent Neural Network
GPU	Graphics Processing Units

Table of Contents

1 Introduction

In the contemporary digital landscape, society is increasingly interconnected. From social media platforms and banking applications to corporate infrastructures, virtually every daily activity involves some degree of data sharing. This data can range from seemingly innocuous details, such as recent entertainment choices or culinary preferences, to more sensitive information. However, data that might appear insignificant—such as a pet's name, a grandmother's maiden name, or a favorite travel destination—can be weaponized by threat actors through social engineering.

It is widely acknowledged that the human element is often the weakest link in the majority of security systems. By default, individuals rarely anticipate how routinely shared information might be leveraged against them; caution is typically learned over time, often prompted by a negative personal experience or targeted educational content.

Furthermore, even when users are security-conscious, they remain vulnerable to direct attacks from malicious software (malware) engineered to inflict social, financial, or operational crises upon individuals and organizations. Security breaches frequently occur without the victim's immediate knowledge, compromising even undisclosed sensitive information. Such incidents lead to the exposure of critical data, including Social Security Numbers (SSNs), credit card details, and residential addresses, allowing cybercriminals to maximize their illicit profits.

Historically, a perpetual arms race has existed between malware authors and security software developers. While one side strives to construct barriers and fortify defenses, the other continuously innovates to bypass these obstacles and exfiltrate targeted information. Most traditional antivirus software relies on mapping the "digital fingerprints"(signatures) of known malware variants, which confers an advantage in identifying and neutralizing recognized threats. However, sophisticated malware complicates this process. For instance, polymorphic malware rewrites its own code during execution, evading signature-based detection and forcing defense systems to rely on continuous updates to prevent "zero-day"attacks (previously unseen threats).

To address these evasive techniques, the study by Catak et al. (2020) (CATAK et al., 2020), titled "Deep learning based Sequential model for malware analysis using Windows exe API Calls,"demonstrates a behavioral analysis methodology that classifies malware based on its utilization of Windows API calls. Consequently, even if a malware variant is polymorphic, its execution patterns allow for identification by analyzing which operating system features are invoked and the sequential order of those invocations. Their research reported classification

accuracy rates ranging from 83.5% to 95.8%, confirming that sequential behavioral analysis is a highly valid approach for malware categorization.

This work proposes to analyze and compare deep learning architectures for classifying malware families based on their Windows API call sequences. The research initiates with a strict replication of the single-layer LSTM methodology utilized by Catak et al. (CATAK et al., 2020) to establish a quantitative baseline. Subsequently, it proposes a hybrid architecture combining a Convolutional Neural Network (CNN) with a Bidirectional LSTM. This approach was designed to investigate whether spatial feature extraction via a CNN, applied prior to temporal processing, can overcome the baseline model's limitations in handling long API sequences.

In this proposed architecture, the Convolutional Neural Network functions strictly as an automated feature extractor. This approach is frequently employed in contemporary academic literature (QI et al., 2023; TAHA; DIAS; WERGHI, 2017). Numerous studies demonstrate that convolutional layers can successfully replace manual feature engineering techniques, acting as robust spatial pre-processors not only for recurrent networks like LSTMs (forming hybrid architectures) but also for classical machine learning classifiers, such as Support Vector Machines (SVM) and Random Forests.

1.1 Objectives

This project was established with two primary objectives. First, to validate the reproducibility of the methodology proposed by Catak et al. by rigorously evaluating their single-layer LSTM model. Second, and more importantly, to propose an architectural optimization utilizing a hybrid network (CNN-LSTM). This proposed model aims to address the baseline architecture's deficiencies in accurately classifying complex malware families, such as spyware and trojans, thereby demonstrating that a hybrid approach offers superior robustness and feature extraction capabilities for this type of sequential data.

1.1.1 Specific Objectives

The specific objectives of this work were to:

1. Pre-process the dataset of sequential Windows API calls by applying data cleaning and tokenization techniques to enable neural network training;

2. Implement and evaluate the reference single-layer LSTM architecture proposed by Catak et al., utilizing a one-vs-rest binary classification strategy to establish a quantitative performance baseline;

3. Develop a hybrid architecture (CNN-LSTM) by introducing convolutional layersfor the automatic extraction of spatial features and noise reduction prior to temporal processing;

4. Conduct a comparative analysis between the two models using robust evaluationmetrics (F1-Score, Recall, and Confusion Matrices) to demonstrate the efficacy of the hybrid approach in accurately detecting complex malware families, such as Trojans and Spyware.

1.2 Contributions

Dynamic malware analysis via Windows API calls is currently one of the most reliable detection techniques; however, it naturally generates excessively long data sequences. Purely sequential models, such as standard LSTMs, face significant difficulties in processing this volume of information without losing long-term dependencies or becoming saturated by semantic noise. This work is justified by proposing an architectural solution to this problem: utilizing a Convolutional Neural Network (CNN) as a spatial filter to extract behavioral patterns from smaller API blocks, thereby facilitating and optimizing the temporal learning phase of the LSTM.

The main contributions of this research can be outlined as follows:

1. Replication and critique: A rigorous replication of the reference single-layerLSTM model (Catak et al.), demonstrating its architectural limitations in accurately classifying complex malware families (such as Spyware and Trojans);

2. Architectural proposition: The development of a hybrid model (CNN-LSTM) that provides a more robust approach to malware behavioral detection;

3. Enhanced feature extraction: The application of convolutional filtering to enablethe neural network to effectively learn the subtle structural differences between highly similar malware types;

4. Comparative validation: Empirical proof, through robust evaluation metrics,that the proposed hybrid approach overcomes the limitations of the purely sequential baseline model.

The remainder of this work is organized into the following chapters:

Chapter 2 - Literature Review. This chapter presents the fundamental concepts of dynamic malware analysis via Windows API calls, alongside the theoretical foundations of Deep Learning, LSTM architectures, and Convolutional Neural Networks (CNNs).

Chapter 3 - Methodology. This chapter details the proposed methodology, describing the data pre-processing techniques and the underlying configurations of the implemented neural network architectures (specifically, the single-layer LSTM replication and the proposed hybrid CNN-LSTM model).

Chapter 4 - Experiments and Results. This chapter introduces the utilized dataset and presents the experimental setup. Subsequently, it provides a comprehensive comparative analysis of the evaluation metrics and results obtained by both the reference and the proposed models.

Chapter 5 - Conclusion. The final chapter summarizes the findings of this research, reinforcing the study's contributions to the field.

2 Literature Review

This chapter presents the literature review, establishing the theoretical foundation for this research. The primary concepts and technologies pertinent to the field are introduced to provide the necessary background for understanding the addressed problem. Furthermore, related works exploring similar approaches are analyzed, enabling the identification of research gaps, methodological inspirations, and differentiating factors that guide the objectives of this study. Ultimately, the subsequent sections contextualize the central elements of the research, providing the theoretical framework that underpins the applied methodologies.

2.1 Malware

Malware is defined as any software intentionally designed to cause harm to a computer, network, system, or user (SIKORSKI; HONIG, 2012). It encompasses a wide variety of classifications, including Trojans, Spyware, Adware, Backdoors, Downloaders, Droppers, Viruses, and Worms, among others. Both this research and the reference study by Catak et al. (CATAK et al., 2020) focus specifically on these eight categories, aiming to identify and classify malware samples into these distinct groups based on their behavioral patterns.

The detection methodology analyzes sequences of Windows API calls to extract the fundamental behavioral patterns inherent to these threats. This dynamic approach is effective even against polymorphic malware. Polymorphism refers to malicious software capable of mutating its decryptors into millions of different forms (SZOR, 2005). By continuously rewriting its own code, polymorphic malware successfully evades traditional signature-based antivirus software; however, its underlying execution behavior (the sequence of API calls it makes to the operating system) remains identifiable.

Adware: Malicious software designed to automatically display or download forced and unwanted advertisements to the user. Although often perceived merely as a nuisance, adware frequently violates user privacy and security, typically installing itself without explicit consent.

Backdoor: A type of malware that covertly bypasses normal authentication procedures to provide an attacker with unauthorized remote access to a system. Backdoors are highly prevalent and versatile; their code often includes a comprehensive set of capabilities, negating the need for the attacker to download additional malicious payloads.

Trojan: Software that misleads the user regarding its true intent. A Trojan disguises itself as a legitimate and benign program, tricking the user into executing it, thereby deploying its hidden malicious payload.

Spyware: Malware specifically engineered to covertly gather information from a user's machine without their knowledge or consent. Common espionage capabilities include keystroke logging (keylogging), screen capturing, credential theft, and monitoring web browsing habits.

Downloader: A specialized script or program whose sole purpose is to fetch and execute additional malicious code from a remote server. Downloaders are typically utilized in the initial stages of a compromise to establish a foothold, pulling heavier malware variants into the infected system.

Dropper: A delivery vehicle designed to stealthily install other malware. Unlike downloaders, droppers usually contain the malicious payload compressed or encrypted within their own structure to evade antivirus detection. Upon execution, the dropper extracts and launches the hidden code.

Virus: A self-replicating malicious program that spreads by inserting copies of itself into other executable files or documents. A virus requires human interaction (such as running an infected program or opening a file) to propagate to other systems.

Worm: Similar to a virus, a worm is a self-replicating threat. However, its primary distinction is that it operates as a standalone program. It does not need to attach itself to a host file, nor does it require human intervention to spread, often exploiting network vulnerabilities to propagate autonomously.

2.2 Malware Detection

Malware detection is the continuous process of identifying and neutralizing malicious software designed to compromise systems, exfiltrate data, or cause infrastructural damage. Traditionally, this defense mechanism has relied on signature-based static analysis, a method in which suspicious files are inspected without being executed. Under this paradigm, security tools, such as classic antivirus software, compare the file's binary code and structural metadata against a global database of known threats. Although this represents a fast and computationally efficient approach against previously mapped threats, it possesses a critical limitation: it is easily bypassed by modern malware variants that utilize polymorphism to dynamically alter their own structure. This structural mutation renders the threat invisible to detectors that search exclusively for static code patterns (GIBERT; MATEU; PLANES, 2020).

To overcome the inherent limitations of static analysis, the field of cybersecurity has evolved towards dynamic and behavioral analysis. This methodology focuses on the execution activity of a file rather than its structural appearance. In this approach, the suspect program is executed within an isolated and controlled environment (commonly referred to as a sandbox)

so that its true behavioral footprint can be strictly monitored. By leveraging advanced analytical models, complex algorithms can learn to recognize the underlying behavioral patterns of various malware families based on their actions during execution. This facilitates a highly effective and proactive detection of sophisticated malware that would otherwise evade traditional, signature-based defense systems (SIHWAIL; OMAR; ARIFFIN, 2018).

2.2.1 Statistical Analysis

Statistical analysis for malware detection relies on extracting features from a file or its execution behavior to construct a baseline profile. When evaluating a new sample, the system calculates statistical deviations or employs probabilistic models to identify anomalies that deviate from established benign patterns, thereby indicating the potential presence of malicious code (YE et al., 2017).

2.2.2 Log Analysis

Log analysis involves the systematic inspection of log files generated by operating systems and applications. This technique searches for forensic traces left by malicious activities, such as failed login attempts or the unauthorized creation of hidden services. By correlating temporal events within these extensive volumes of textual data, it is possible to reconstruct the attack kill chain and detect infections in their early stages (OPREA et al., 2015).

2.2.3 Network Traffic Analysis

Network traffic analysis focuses on the interception and inspection of data packets entering and exiting an infrastructure, without relying on the analysis of the host executable. It is a vital technique for identifying malware variants that attempt to exfiltrate data or establish backdoors, thereby detecting malicious behavior directly at the network layer (GARCIA et al., 2014).

2.2.4 API Calls

Application Programming Interface (API) calls represent the communication bridge between software applications and the operating system. When malware executes malicious actions, it must invoke native Windows APIs. Consequently, the extraction and sequential analysis of these calls during execution within an isolated environment provide the behavioral fingerprint of the software. This dynamic footprint reveals the program's true intentions in a manner that is immune to traditional code obfuscation techniques (AMER; ZELINKA, 2020).

Various computational approaches are employed to interpret the massive volume of data generated by these sequences. Data mining serves as the theoretical foundation for discovering hidden patterns and frequent association rules within vast API logs. Following feature extraction, clustering techniques are utilized to group samples exhibiting similar API behaviors, which is crucial for discovering novel, unlabeled malware variants. Conversely, classification algorithms categorize API sequences into known malware families using previously labeled examples.

More recently, deep neural networks (such as CNNs and LSTMs) have achieved superior results in this domain. These architectures are capable of learning not just the isolated presence of an API call, but also the complex spatial and temporal dependencies across thousands of consecutive calls, thereby outperforming classical machine learning methods in the detection of highly sophisticated threats (KOLOSNJAJI et al., 2016).

2.2.5 Recurrent Neural Networks and LSTM Architecture

Recurrent Neural Networks (RNNs) were originally designed to process sequential data, utilizing architectural loops to create a short-term "memory"of the information flow. However, when processing excessively long sequences—such as vast amounts of Windows API calls—traditional RNNs fail due to a mathematical limitation known as the vanishing gradient problem. Consequently, during training, the learning signal dissipates across the earlier timesteps, causing the network to lose the ability to connect distant events and effectively "forgetting"initial sequential information.

To overcome this architectural restriction, Hochreiter and Schmidhuber developed the Long Short-Term Memory (LSTM) network (HOCHREITER; SCHMIDHUBER, 1997). The LSTM is a direct evolution of the standard RNN that introduces a more sophisticated internal structure: a "memory cell"regulated by three logic gates (the forget gate, the input gate, and the output gate).

Instead of passively passing sequential information forward, these gates utilize mathematical operations to actively determine which data becomes irrelevant and should be discarded, and which features are vital and must be retained over the long term. In the context of dynamic malware analysis, the LSTM mechanism is fundamental. It enables the correlation of a malicious action occurring at the end of a file's execution with the initial system configuration calls, thereby preserving the global context of the infection and substantially overcoming the limitations of traditional RNNs.

2.3 Related Works

This section presents relevant state-of-the-art studies in the classification and detection of malware using machine learning and deep neural network techniques. The review of these works aims not only to substantiate the architectural choices adopted in this study but also to highlight existing gaps in the current research landscape. Specifically, it emphasizes the challenges associated with accurately classifying complex malware families from extensive sequential data, thereby justifying the methodological contributions proposed in this monograph.

2.3.1 LSTM Malware Classification

The work developed by Catak et al. (2020) (CATAK et al., 2020) represents the foundational baseline for this research. The authors proposed a dynamic analysis methodology for multiclass malware classification, based on the sequential extraction of Windows operating system API calls. Utilizing the Cuckoo Sandbox environment, the researchers monitored the execution of thousands of malicious samples, compiling a public dataset containing 7,107 instances categorized into eight distinct families (such as Trojans, Worms, and Spyware).

To process these temporal sequences, the authors implemented a recurrent neural network architecture based on a single-layer LSTM, trained under a one-vs-rest paradigm. Although the study achieved satisfactory global metrics and demonstrated the viability of using LSTMs to model malicious software behavior, the baseline architecture exhibited critical limitations when processing excessively long API sequences. The purely sequential model struggled with semantic noise, resulting in reduced detection rates for more complex malware families. This specific architectural limitation serves as the primary motivation for this thesis, which proposes a hybrid architecture (CNN-LSTM) designed to enhance spatial feature extraction and improve the overall robustness of the classification.

2.3.2 Space-Temporal Characteristic Extraction

Kolosnjaji et al. (2016) (KOLOSNJAJI et al., 2016) presented one of the pioneering approaches combining convolutional and recurrent neural networks for cybersecurity analysis. The study utilized sequential system calls to construct a hybrid architecture comprising Convolutional Neural Network (CNN) layers followed by LSTM layers. The primary methodological contribution of this research was demonstrating that convolutional layers act as highly effective n-gram feature extractors (identifying blocks of API calls that frequently co-occur), while the subsequent LSTM layers capture the complex temporal dependencies between these extracted blocks. The findings of Kolosnjaji et al. directly inspired the

formulation of the hybrid architecture proposed in this monograph, consolidating the premise that a hybrid approach significantly outperforms purely recurrent baseline models.

2.3.3 Fundamentation of Recurrent Architecture

Although not exclusively focused on information security, the seminal paper by Hochreiter and Schmidhuber (1997) (HOCHREITER; SCHMIDHUBER, 1997) serves as the foundational literature that enables the methodological approach of this monograph. The authors identified and solved the vanishing gradient problem inherent in classical recurrent neural networks by introducing the Long Short-Term Memory (LSTM) architecture. By establishing an information flow regulated by logic gates (forget, input, and output), this work mathematically demonstrated how to preserve context across extensively long sequences. The direct application of this literature in the present research lies in the model's ability to correlate Windows API calls executed at the onset of an infection with subsequent malicious actions, thereby justifying the selection of the single-layer LSTM as the baseline reference model.

2.3.4 Malicious Software Detection with Hybrid Model CNN-LSTM

In direct alignment with the objectives of this monograph, Qi et al. (2023) (QI et al., 2023) developed a hybrid CNN-LSTM model for malware classification based on API call sequences extracted within a sandbox environment. The study applied rigorous data pre-processing techniques, including repetition removal and the implementation of Embedding layers. Their research demonstrated that the application of a Convolutional Neural Network (CNN) prior to recurrent processing (LSTM) enables the optimized extraction of spatial features—specifically, the identification of malicious blocks of API calls that frequently co-occur. This hybrid model yielded significant improvements in accuracy and F1-Score when compared to purely sequential baselines, such as single-layer LSTM architectures. Consequently, their findings provide state-of-the-art theoretical validation that scientifically justifies the architectural design of the model proposed in this work.

2.3.5 Ecosystem and Evaluation of Machine Learning

To orchestrate the processing of high-dimensional tensors and guarantee the computational efficiency of the trained models, this research relies on foundational machine learning engineering literature. Abadi et al. (2016) (ABADI et al., 2016) substantiate the use of the TensorFlow framework, detailing how the parallel execution of computational graphs on Graphics Processing Units (GPUs) enables the efficient training of deep neural architectures. The relevance of their work to this monograph is strictly operational and architectural: the large-scale processing mechanics validated by Abadi et al. facilitate the compilation, weight

optimization, and computationally viable training of the hybrid CNN-LSTM network detailed in this study's methodology.

2.4 Technologies and Tools

This section outlines the methodological infrastructure and technological resources employed to conduct the empirical experiments of this research. Ensuring scientific transparency and the reproducibility of the proposed models requires a precise description of the experimental setup. To achieve this, the following subsections systematically detail the fundamental components utilized in this study. Section 2.4.1 describes the origin, structure, and behavioral characteristics of the dataset used to train and validate the neural networks. Subsequently, Section 2.4.2 specifies the hardware environment, highlighting the computational processing power strictly necessary for deep learning convergence. Finally, Section 2.4.3 outlines the software ecosystem, detailing the programming language, frameworks, and auxiliary libraries that supported the structural development and rigorous evaluation of the architectures.

2.4.1 Dataset

The dataset utilized in this research was originally introduced in the reference work by Catak et al. (CATAK et al., 2020). The repository comprises two primary files: one containing the sequential Windows API calls (all_analysis_data.txt) and another containing their corresponding target classifications (labels.csv). The dataset consists of 7,107 total instances, categorized into eight distinct malware families: Adware, Backdoor, Downloader, Dropper, Spyware, Trojan, Virus, and Worm. For the experimental methodology, the data was partitioned into an 80% training set and a 20% testing set.

2.4.2 Hardware Environment

The experiments were conducted on a hardware platform equipped with an 11th Gen Intel Core i7-11800H processor (2.30 GHz), 16 GB of RAM, and an NVIDIA GeForce
RTX 3050 Ti Laptop GPU. Hardware acceleration was enabled to leverage the Graphics Processing Unit (GPU), providing the necessary parallel processing capabilities for the efficient training of the proposed deep learning architectures.

2.4.3 Software and Programming Language

Regarding the software environment, the entire implementation was developed using the Python programming language. The construction, training, and evaluation of the neural networks—including both CNN and LSTM layers—were conducted using the TensorFlow and Keras libraries. Additionally, the Scikit-learn library was utilized for data pre-processing, class weighting strategies, and the calculation of evaluation metrics. For tabular data manipulation and the visualization of results, such as the generation of confusion matrices and training plots, the NumPy, Pandas, Matplotlib, and Seaborn libraries were employed.

The following chapter reviews the original model proposed by Catak et al. and introduces the hybrid architecture developed in this research. It details the architectural modifications and methodological enhancements that enabled superior performance in classification precision.

3 Methodology

This chapter details the proposed methodology and the technical procedures adopted for the development, training, and optimization of the neural networks applied to malware classification. The research design was structured into logical and sequential phases to ensure scientific rigor and the exact reproducibility of the experiments. Initially, the dataset utilized in this study is presented, detailing its composition and its inherent class distribution. Subsequently, the pre-processing steps are described; these steps are essential for transforming the textual sequences of API calls into the high-dimensional mathematical tensors required by deep learning algorithms. Finally, the chapter details the class weighting strategies implemented and the specific neural network architectures constructed, encompassing both the replication of the baseline reference model (singlelayer LSTM) and the proposed hybrid model (CNN-LSTM).

3.1 Data Pre-processing

The raw data extracted from the dynamic analysis, consisting of textual API call sequences, requires a series of mathematical and structural transformations before it can be processed by neural network architectures. The pre-processing pipeline applied in this study was divided into cleaning, vectorization, and geometric data standardization phases.

The first step involved behavioral noise filtering through the removal of consecutive repetitions. In malicious executions, it is common for the same API call to be invoked repeatedly within a loop. To optimize the representation of program state transitions, a cleaning routine was developed to collapse identical consecutive calls. This approach significantly reduces the sequence length without losing its semantic value regarding the malware's overall behavior.

Following the cleaning phase, vocabulary scanning was performed via tokenization. Utilizing the Tokenizer class, each distinct textual API call was mapped to a unique integer value, constructing a dictionary based on occurrence frequencies across the entire dataset. The vocabulary size was restricted to the top 5,000 most frequent API calls, ensuring that the neural networks focus their processing capacity on the most relevant operating system functions. Rare or unmapped calls were replaced with a standard outof-vocabulary (OOV) token.

Subsequently, padding and truncating techniques were applied to resolve length variations among the sequences. Because deep learning models require input tensors with

fixed dimensions, the maximum sequence length had to be standardized. For the reference baseline (single-layer LSTM), the length was capped at 100 calls, strictly respecting its original architectural constraints. In contrast, for the proposed hybrid architecture (CNNLSTM), the maximum length was expanded to 1,000 calls, granting the convolutional layers a broader receptive field for spatial feature extraction. In both scenarios, shorter sequences were padded with zeros at the end (post-padding), while longer sequences were truncated to their respective maximum capacities (post-truncating).

Lastly, the target labels identifying the eight malware families, originally in textual format, were converted to numerical values (ranging from 0 to 7) using the LabelEncoder method. With the input features (*X*) and the target labels (*y*) properly formatted, the complete dataset was randomly partitioned into an 80% training set (for optimizing network weights) and a 20% testing set (for validation). This strict segregation ensures that the final performance evaluation of the models is conducted exclusively on unseen data.

3.2 Training and Balancing Strategies

The inherent class distribution within the dataset presented a potential risk for classification bias. If naively trained, a neural network might disproportionately learn generic malware behaviors, favoring the majority families. To mitigate this effect and compel the models to extract the subtle, specific behavioral signatures of each complex family, two distinct training strategies were adopted, tailored strictly to the specific architecture under evaluation.

For the reference baseline replication (single-layer LSTM), a one-vs-rest (OvR) binary classification strategy was implemented. In this approach, rather than utilizing a single network to classify all categories simultaneously, eight independent "specialist"models were trained. Each model was tasked with determining whether a given sequence belonged to a specific target family (positive instance) or to any of the other families (negative instance). To prevent the target class from being overshadowed by the aggregate majority, a strict random undersampling technique was applied at a 1:1 ratio. For instance, to train the Adware specialist model, a training subset was generated comprising all Adware instances alongside an exactly equal number of randomly selected instances from the remaining groups. This strict equilibrium compelled the network to disregard generic API calls common to all malware and focus exclusively on the specific sequential patterns that differentiate the target class. This strategy maximized the discriminative capability of an architecture with inherently limited feature extraction capacities.

Conversely, for the proposed hybrid architecture (CNN-LSTM), the data elimination imposed by strict undersampling would represent a significant loss of sequential information, which is essential for deep feature extraction. Therefore, a direct multiclass classification

strategy was chosen, enabling a single unified model to process the entirety of the training dataset simultaneously.

To address the class distribution disparities in this second approach, data reduction was substituted with the application of algorithmic class weighting. Utilizing scikit-learn functions, a specific mathematical weight was calculated for each malware family, inversely proportional to its frequency within the training dataset. During the training phase, these weights were injected directly into the neural network's loss function. Consequently, the optimization algorithm was configured to apply a significantly heavier mathematical penalty when the model incorrectly classified an instance from a highly complex, underrepresented family (such as Spyware or Trojans) compared to a misclassification of a majority class. This technique ensured that these specific families achieved equivalent gradient importance during the network's weight updates, fostering an equitable learning landscape without discarding any original sequential data.

3.3 Neural Network Architectures

The foundation of this research lies in the implementation and comparative analysis of two distinct deep learning architectures designed for processing sequential API calls. The primary objective was to establish a quantitative baseline by rigorously replicating a reference methodology (the single-layer LSTM). Subsequently, this study proposes and evaluates an optimized hybrid architecture (CNN-LSTM) designed specifically to overcome the baseline's inherent deficiencies in accurately classifying complex malware families.

3.3.1 Reference Model (Single-Layer LSTM)

To validate the reproducibility of the study by Catak et al. (CATAK et al., 2020), the purely sequential reference architecture, characterized by a single Long Short-Term Memory (LSTM) layer, was implemented. The network receives standardized sequences of 100 API calls as input, which are first processed by an Embedding layer configured to generate dense 32-dimensional vectors. This compact vector representation is subsequently passed to the LSTM layer, equipped with 200 memory units, which is responsible for extracting the fundamental temporal dependencies among the functions invoked by the executable.

The selection of an Embedding layer is justified by its capacity to project discrete numerical identifiers into a continuous vector space. This mathematical transformation enables the model to learn the semantic similarities between API calls that frequently cooccur during execution. From a structural perspective, the input sequence is transformed into a dense three-dimensional tensor. As this tensor is processed chronologically, the internal logic

gates of the LSTM dynamically update the cell state at each timestep, ultimately constructing a final vector representation that encapsulates the behavioral essence of the malware. Figure 1 illustrates the single-layer LSTM baseline architecture.

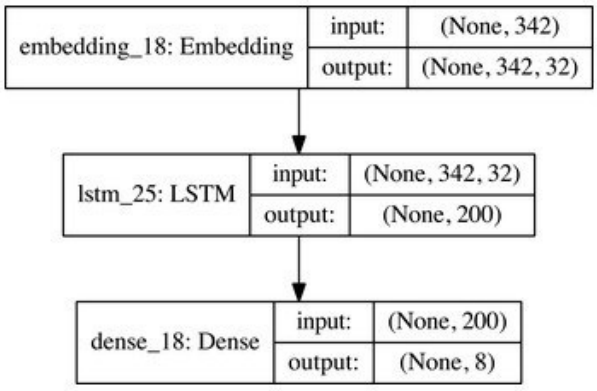

Figure 1 – Schematic representation of the reference single-layer LSTM architecture (CATAK et al., 2020).

To mitigate overfitting, a dropout rate of 0.2 was applied to the LSTM connections. Because this architecture was trained under the one-vs-rest (OvR) paradigm, the final network layer consists of a single dense neuron utilizing a Sigmoid activation function. This configuration is designed to output a continuous probability between 0 and 1, indicating whether the analyzed sequence belongs to the target class. The model was compiled using the Binary Crossentropy loss function and the Adam optimizer.

The dropout mechanism enhances the architecture's generalization capacity. By randomly deactivating a fraction of the neural connections during training, this technique prevents the excessive co-adaptation of neurons to noisy patterns within the training data, thereby forcing the network to learn robust and redundant behavioral signatures. Finally, the adoption of the Adam (Adaptive Moment Estimation) algorithm for weight optimization is justified by its ability to dynamically adapt the learning rate for each individual parameter. By calculating the first and second-order moments of the gradients, Adam facilitates stable convergence within the highly complex feature space of system calls.

3.3.2 Proposed Model (Hybrid CNN-LSTM)

Catak et al. (2020) demonstrated that the single-layer LSTM architecture exhibited limitations in differentiating malware families with highly similar behaviors, primarily due to the semantic noise generated by excessively long sequences of generic API calls. To address this limitation, a hybrid multiclass architecture integrating one-dimensional convolutional layers (1D-CNN) and bidirectional recurrent layers (Bi-LSTM) was proposed and implemented.

The structural hypothesis underpinning this model posits that the CNN acts as an automated spatial feature extractor—identifying suspicious blocks or *n*-grams of API calls—prior to the recurrent temporal analysis.

The experimental setup expanded the maximum sequence length from 100 (CATAK et al., 2020) to 1,000 system calls. These sequences are subsequently processed by an Embedding layer expanded to 128 dimensions, thereby creating a richer, more expressive continuous vector space. The resulting vector sequence is then fed into two successive convolutional blocks. The first block utilizes 128 filters (with a kernel size of 5), followed by MaxPooling and Batch Normalization layers. This configuration reduces the spatial dimensionality while simultaneously highlighting the most salient local behavioral patterns. The second block deepens the feature extraction process using 256 filters (with a kernel size of 3).

The resulting feature map, now filtered of peripheral semantic noise, is injected into a 128-unit bidirectional LSTM (Bi-LSTM) layer. The bidirectional architecture enables the network to analyze the context of API calls in both chronological and reverse order, significantly enhancing the robustness of the behavioral modeling. Furthermore, the temporal output is processed by fully connected dense layers applying an aggressive dropout rate of 0.4. This culminates in an output layer comprising 8 neurons with a Softmax activation function, which calculates the probability distribution indicating the malware's classification among the eight families present in the dataset. The final model was compiled using the Adam optimizer and the Sparse Categorical Crossentropy loss function. Figure 2 illustrates the architecture of this proposed hybrid model, where the highlighted red background explicitly denotes the convolutional (CNN) modules integrated within the network.

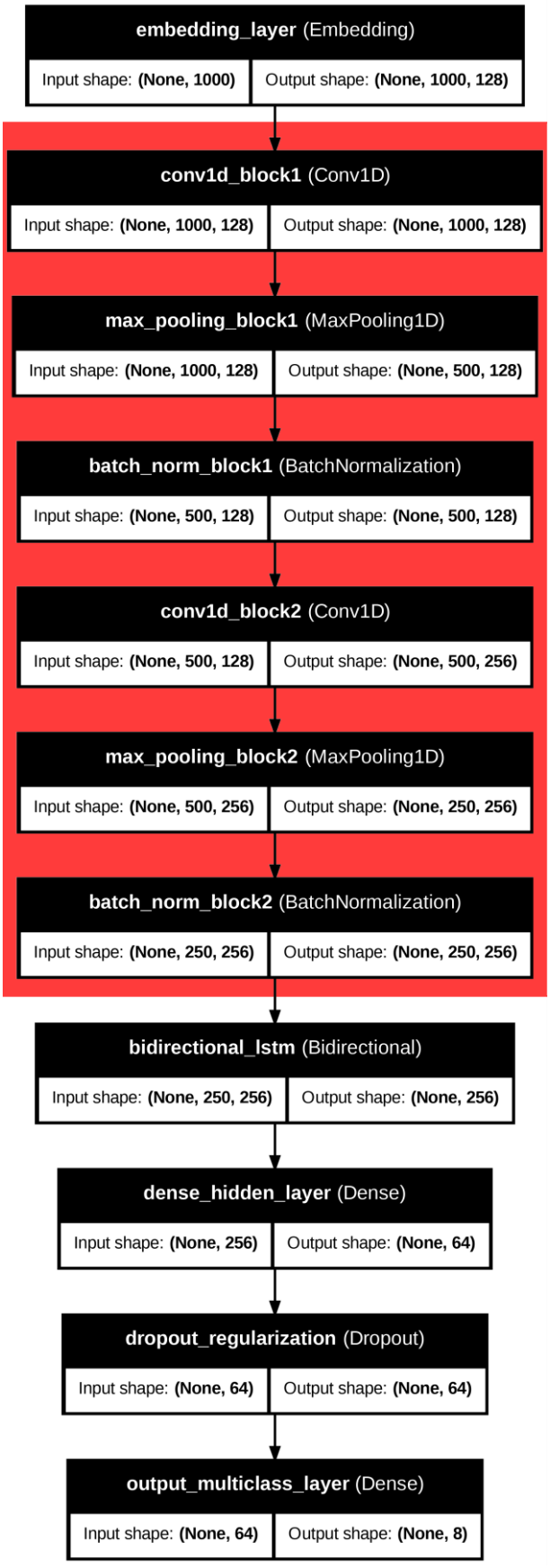

Figure 2 – Schematic representation of the proposed hybrid CNN-LSTM architecture.

4 Experiments and Results

This chapter presents the experiments conducted and discusses the results obtained, with the primary objective of empirically evaluating the efficacy of the proposed hybrid architecture (CNN-LSTM) in comparison to the reference baseline (single-layer LSTM).

To ensure the transparency and reproducibility of the research, Section 4.1 details the structural composition of the dataset utilized, while Section 4.2 specifies the hardware infrastructure and software ecosystem comprising the experimental environment. Subsequently, Section 4.3 formalizes the statistical metrics adopted to evaluate algorithmic performance.

Following this foundation, Sections 4.4 and 4.5 detail the empirical results of the reference baseline and the proposed hybrid architecture, respectively. These sections utilize confusion matrices and classification reports to illustrate the networks' behavior when navigating the dataset's inherent class distribution. Lastly, Section 4.6 consolidates the study through a comprehensive comparative analysis. It highlights the impact of architectural hybridization on deep feature extraction and the resulting improvements in detection precision, particularly concerning highly complex malware families.

4.1 Dataset

To conduct the experiments proposed in this study, a public dataset originally provided by Catak et al. (CATAK et al., 2020) was utilized. This dataset was constructed using dynamic malware analysis techniques, wherein malicious samples were executed within an isolated and controlled environment (Cuckoo Sandbox) (OKTAVIANTO; MUHARDIANTO, 2013). During execution, the behavior of each sample was continuously monitored to capture the exact sequence of Windows operating system API calls invoked by the program, thereby generating a distinct behavioral "fingerprint"for each threat.

The raw data was provided in two primary files: all_analysis_data.txt, which contains the temporal sequences (where each line represents the chronological order of API calls invoked by a single executable), and labels.csv, which contains the corresponding classification labels for each sequence. The complete dataset comprises 7,107 unique instances. Crucially, 100% of this dataset consists of malicious software, containing no benign samples (goodware). Consequently, the objective of the predictive models developed in this research is not binary threat detection (distinguishing between safe and malicious software), but rather multiclass

categorization: identifying the specific family to which a malware instance belongs based exclusively on its execution behavior.

The instances within the dataset are categorized into eight distinct malware families: Adware, Backdoor, Downloader, Dropper, Spyware, Trojan, Virus, and Worm. Figure 3 details the exact quantitative distribution of instances across each of these categories.

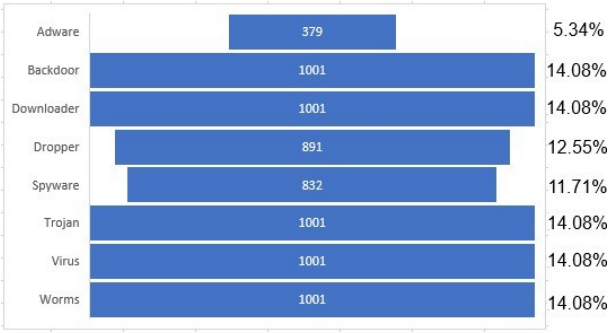

Figure 3 – Malware class distribution

As illustrated in the data distribution, the dataset exhibits an inherent structural asymmetry among its classes. Highly complex families, such as Adware and Spyware, possess a substantially lower representation compared to majority classes like Worms and Backdoors. Although this numerical disparity accurately reflects the real-world prevalence of various threats in information security environments, it introduces a potential learning bias toward the more frequent and generalized malware behaviors. This characteristic of the dataset, coupled with the intricate behavioral patterns of the underrepresented families, justifies the implementation of the specialized mitigation strategies detailed in the subsequent sections.

4.2 Experimental Environment and Technologies

The experiments presented in this work were conducted using a computational environment with the following hardware specifications: an 11th-generation Intel Core i7-11800H processor operating at 2.30 GHz, 16 GB of RAM, and an NVIDIA GeForce RTX 3050 Ti Laptop GPU. Hardware acceleration was enabled to leverage the Graphics Processing Unit (GPU), providing the parallel processing power strictly necessary for the efficient training and convergence of the proposed deep neural architectures.

Regarding the software ecosystem, all empirical implementations were developed using the Python programming language. The construction, training, and evaluation of the neural networks were executed utilizing the open-source TensorFlow and Keras libraries. Additionally, the Scikit-learn library was employed during the data pre-processing phase, for the application of class weighting techniques, and for the mathematical computation of the

evaluation metrics. Lastly, the NumPy, Pandas, Matplotlib, and Seaborn libraries were utilized collectively for tabular data manipulation and the generation of detailed visual confusion matrices.

4.3 Evaluation Metrics

To measure the predictive performance of the developed neural network architectures and facilitate a quantitative comparison between the reference baseline and the proposed model, this research adopted a set of statistical metrics well-established in the machine learning literature.

Because the evaluated dataset exhibits an inherent class distribution reflective of real-world malware scenarios, relying exclusively on the global accuracy metric can yield statistically biased interpretations. Consequently, a detailed analysis utilizing a Confusion Matrix, alongside secondary metrics (Precision, Recall, and F1-Score), is required to rigorously evaluate the model's specific performance when classifying complex, underrepresented families.

Confusion Matrix

The Confusion Matrix is a tabular structure that summarizes the algorithm's predictions relative to the actual ground-truth classifications. In multiclass scenarios, it enables the precise tracking of exactly which malware families the network is misclassifying. To calculate the derivative evaluation metrics, the following essential components are extracted with respect to a given target class:

- **TP (True Positive):** The model correctly predicts that the instance belongs to the target class.
- **TN (True Negative):** The model correctly predicts that the instance does not belong to the target class.
- **FP (False Positive):** The model incorrectly predicts that the instance belongs to the target class, whereas it actually belongs to another family (False Alarm).
- **FN (False Negative):** The model incorrectly predicts that the instance does not belong to the target class, whereas it actually does (Missed Threat).

Derived from these components, the following mathematical formulations were implemented for empirical validation:

Accuracy

Accuracy represents the overall proportion of correct predictions (both positive and negative) relative to the total volume of evaluated instances.

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN} \tag{4.1}$$

Precision

Precision quantifies the success rate of the model's positive predictions. In the context of information security, it answers the following question: of all the files that the network classified as Spyware, how many were actually Spyware? A high precision indicates a low rate of false positives.

$$\text{Precision} = \frac{TP}{TP + FP} \tag{4.2}$$

Recall

Recall measures the model's capacity to identify all the actual positive instances that exist within the dataset. It is a critical metric in intrusion detection systems (IDS), as a low recall value indicates a high rate of false negatives, meaning that malicious instances bypassed the system undetected.

$$\text{Recall} = \frac{TP}{TP + FN} \tag{4.3}$$

F1-Score

The F1-Score is the harmonic mean of Precision and Recall. Unlike the arithmetic mean, the harmonic formulation severely penalizes the final result if there is a significant disparity between Recall and Precision. By enforcing a balance between false positives and false negatives, the F1-Score serves as the definitive metric in this monograph to attest to the generalization power of the analyzed architectures.

$$\text{F1-Score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \tag{4.4}$$

4.4 Empirical Results and Architectural Evaluation

This section presents the empirical findings obtained from the rigorous evaluation of the implemented neural network architectures. To systematically demonstrate the impact of spatial feature extraction on complex malware classification, the results presentation is divided

into three logical stages. First, Section 4.4.1 details the predictive performance and the inherent limitations of the purely sequential reference baseline. Subsequently, Section 4.4.2 presents the classification metrics achieved by the proposed hybrid CNN-LSTM architecture. Finally, Section 4.4.3 consolidates these findings through a comprehensive comparative analysis, highlighting the specific advantages of architectural hybridization in overcoming semantic noise and improving the detection of intricate threat families.

4.4.1 Reference Baseline Results (Single-Layer LSTM)

The first empirical step of this study was to evaluate the replicated reference architecture (single-layer LSTM). To strictly reflect the methodology established by Catak et al. (2020) (CATAK et al., 2020), the experiment was conducted using the one-vs-rest (OvR) approach, effectively dividing the multiclass problem into eight independent binary classifiers, each culminating in a dense output layer with a Sigmoid activation function. To ensure maximum model convergence and achieve the optimal results required by the methodological rigor of this research, the training phase was extended to 500 epochs, demanding a total computational processing time of 407 minutes.

During this extensive training, the models were exposed to the real dataset distribution, strictly preserving its inherent structural asymmetry. Figure 4 presents the consolidated confusion matrix for the 1,422 instances in the testing set, while Table 1 summarizes the resulting performance metrics.

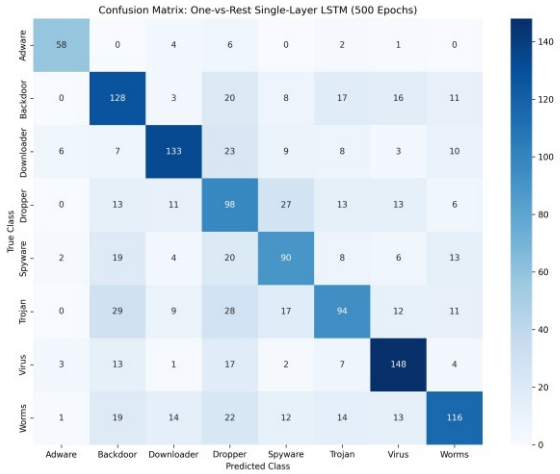

Figure 4 – Confusion Matrix of Single-Layer LSTM

Class	Precision	Recall	F1-Score
Adware	0.83	0.82	0.82
Backdoor	0.56	0.63	0.59
Downloader	0.74	0.67	0.70
Dropper	0.42	0.54	0.47

Spyware	0.55	0.56	0.55
Trojan	0.58	0.47	0.52
Virus	0.70	0.76	0.73
Worms	0.68	0.55	0.61
Accuracy		**-**	**0.61**
Weighted Average	**0.62**	**0.61**	**0.61**

Table 1 – Performance metrics of the reference single-layer LSTM architecture.

The analysis of the results demonstrates the architectural limitations of the baseline model when processing the dataset's inherent structural asymmetry. Although the One-vs-Rest (OvR) strategy and an extended training regimen—which demanded 407 minutes of continuous computational processing—allowed for robust classification indicators in distinct families such as Adware (F1-Score of 0.82) and Virus (F1-Score of 0.73), the global accuracy plateaued at 61%.

The most critical insight revealed by the empirical evaluation lies within the confusion matrix (Figure 4). Rather than exhibiting a singular predictive bias, the neural network demonstrates severe semantic confusion among highly complex malware families that share overlapping execution behaviors. For instance, the Trojan class (F1-Score of 0.52) suffers from a low recall rate of 0.47, with 29 instances misclassified as Backdoors and 28 as Droppers. Similarly, 27 Dropper instances were incorrectly labeled as Spyware. This cross-misclassification elegantly highlights the "semantic noise"generated by long sequences of generic API calls: because Trojans frequently exhibit backdoor-like network activities or drop malicious payloads during infection, their raw sequential footprints become largely indistinguishable to a purely recurrent model.

Consequently, families characterized by intricate and multifaceted execution patterns suffered critical performance degradation. The Dropper class, deeply affected by this behavioral overlap, achieved a precision of only 0.42, while Spyware reached an F1-Score of merely 0.55.

In this context, it is imperative to address the reproducibility divergence observed in this experiment. Despite the rigorous replication methodology and extended training regimen, the model could not sustain the near-perfect F1-Scores reported in the original reference study. This discrepancy suggests that achieving such optimal metrics with a single-layer LSTM may require undocumented pre-processing steps or artificial data balancing, thereby reinforcing the necessity for an inherently more robust architectural approach.

Ultimately, these empirical findings validate the central hypothesis of this monograph: the temporal memory mechanism of the LSTM, when exposed to raw, unfiltered API sequences, is highly susceptible to behavioral obfuscation. This predictive ceiling empirically

demonstrates that purely sequential processing is insufficient; it necessitates a preliminary phase of spatial feature extraction to filter peripheral noise and identify definitive malicious blocks (*n*-grams) prior to temporal analysis, thoroughly justifying the proposed hybrid CNN-LSTM architecture.

4.4.2 Proposed Model Results (CNN-LSTM Hybrid)

The culminating phase of the empirical experimentation consisted of evaluating the hybrid architecture proposed in this research. This model transcends purely temporal limitations by integrating deep spatial feature extraction blocks (two 1D-CNN layers coupled with Batch Normalization), followed by a bidirectional temporal processing mechanism (Bi-LSTM). To address the dataset's inherent structural asymmetry without artificially distorting the real-world data distribution (which could inadvertently introduce data leakage), the training process was strictly optimized utilizing algorithmic class weighting. This technique applies significantly heavier mathematical penalties to the loss function when misclassifications occur within highly complex, underrepresented families. Figure 5 illustrates the resulting multiclass confusion matrix, and Table 2 details the corresponding performance metrics.

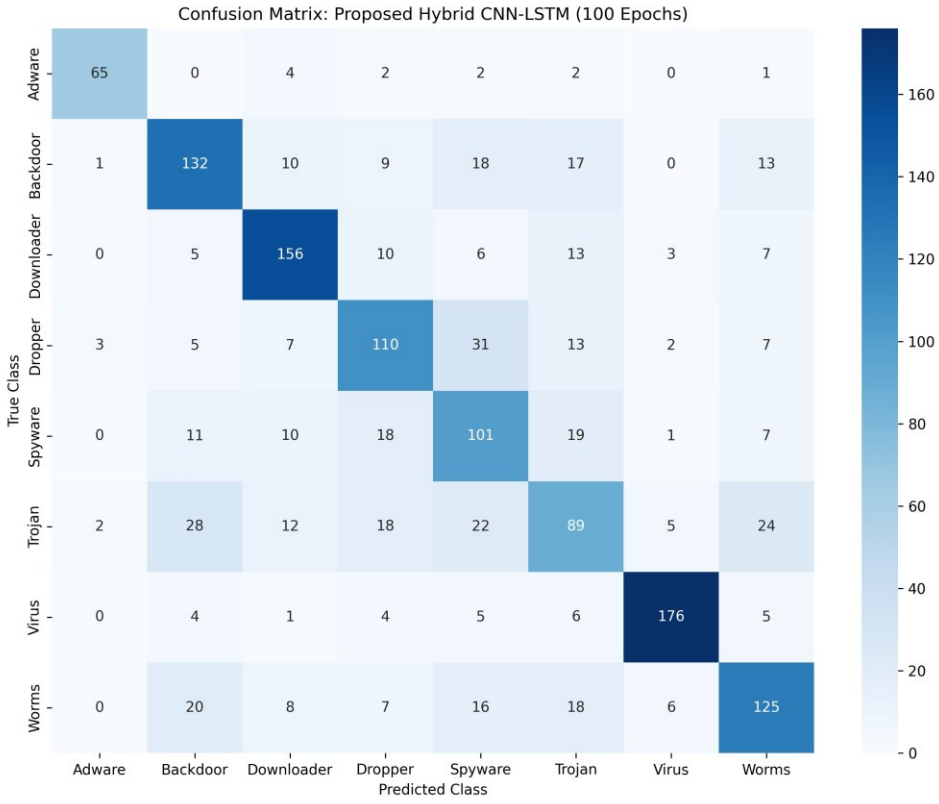

Figure 5 – Confusion Matrix of the proposed CNN-LSTM Hybrid architecture over 100 epochs.

Class	Precision	Recall	F1-Score
Adware	0.92	0.86	0.88

Backdoor	0.64	0.66	0.65
Downloader	0.75	0.78	0.76
Dropper	0.62	0.62	0.62
Spyware	0.50	0.60	0.55
Trojan	0.50	0.45	0.47
Virus	0.91	0.88	0.89
Worms	0.66	0.62	0.64
Accuracy		**-**	**0.67**
Weighted Average	**0.67**	**0.67**	**0.67**

Table 2 – Performance metrics of the proposed CNN-LSTM Hybrid architecture.

The analysis of the results reveals a substantial performance gain and a predictive behavior fundamentally superior to the reference baseline. Achieving a global accuracy of 67.09% in just 12.4 minutes of computational training time, the network demonstrates an absolute gain of approximately 6 percentage points while drastically reducing the time cost compared to the 500-epoch single-layer LSTM (which required 407 minutes). Notably, the model does not merely overfit to the majority classes; instead, it distributes its learning effectively across the dataset's inherent structural asymmetry.

The confusion matrix (Figure 5) robustly attests to the efficacy of the convolutional filtering (1D-CNN). The severe semantic confusion observed in the purely recurrent baseline was significantly mitigated. The predictions are now distinctly concentrated along the main diagonal of the matrix, empirically proving that the CNN filters successfully mapped local blocks of API calls (*n*-grams). This spatial extraction isolated peripheral noise and captured the unique behavioral signatures of each family prior to the temporal inference. This capability is highlighted by the exceptional classification metrics for Virus (F1-Score of 0.89) and Adware (F1-Score of 0.88).

The primary empirical triumph of this architecture, however, materializes in the detection of highly complex, underrepresented families such as Spyware. In purely recurrent evaluations, families with intricate and multifaceted execution patterns frequently suffered from predictive degradation. In the proposed architecture, enhanced by the algorithmic class weighting technique, the recall for Spyware reached 0.60, achieving a robust F1-Score of 0.55. Although the Trojan class remains the most complex behavioral challenge (F1-Score of 0.47), this result still heavily surpasses any previously evaluated baseline scenario.

Ultimately, the experiments confirm that the logical topology of malicious API sequences contains strong local dependencies that cannot be fully comprehended through strictly sequential processing. The CNN-LSTM hybridization bridges this gap by first processing

the malware behavior spatially—extracting localized contextual blocks—and subsequently mapping the narrative evolution of these blocks temporally. This dual-phase approach proves to be a highly viable and computationally efficient solution for multiclass threat categorization in environments characterized by inherent structural asymmetries.

4.4.3 Comparative Analysis and General Discussion

To consolidate the experimental findings of this research and empirically validate the methodological proposition, this section presents a comprehensive analytical comparison between the evaluated deep learning architectures: the reference baseline (single-layer LSTM) and the proposed hybrid model (CNN-LSTM enhanced with algorithmic class weighting).

Because the evaluated dataset reflects the inherent structural asymmetry characteristic of real-world cybersecurity environments, the F1-Score was adopted as the primary robust indicator for this comparison, as it strictly penalizes biased distributions of false positives and false negatives. Table 3 compiles the performance of each architecture by malware family, alongside their respective global metrics and the relative mathematical variation between the models.

Malware Family	**Single-Layer LSTM**	**Hybrid (CNN-LSTM)**	**Relative Change**
Adware	0.82	0.88	+7.3%
Backdoor	0.59	0.65	+10.2%
Downloader	0.70	0.76	+8.6%
Dropper	0.47	0.62	+31.9%
Spyware	0.55	0.55	0.0%
Trojan	0.52	0.47	-9.6%
Virus	0.73	0.89	+21.9%
Worms	0.61	0.64	+4.9%
Global Accuracy	**61.0%**	**67.1%**	**+10.0%**
Weighted Average F1	**0.61**	**0.67**	**+9.8%**

Table 3 – Comparative performance analysis between the reference baseline and the proposed hybrid architecture.

The observation of the data reveals the architectural superiority of the hybrid model, particularly regarding computational efficiency and predictive capability. The attempt to force convergence in the purely recurrent baseline through an exhaustive 500-epoch training regimen—which consumed 407 minutes of processing—resulted in a plateaued global accuracy of 61.0%. In stark contrast, the proposed CNN-LSTM architecture achieved a superior global accuracy of 67.1% in merely 12.4 minutes. This empirically validates the primary argument of this research: the bottleneck in processing long sequences of API calls is not a lack

of temporal training duration or memory depth, but rather the absence of a preliminary spatial extraction phase to reduce dimensionality and filter semantic noise.

The impact of convolutional hybridization (1D-CNN) coupled with algorithmic class weighting becomes absolute when analyzing malware families with overlapping, complex behavioral traits. The Dropper class, which suffered from severe cross-misclassification in the reference baseline, exhibited a remarkable relative growth of +31.9% in its F1-Score (evolving from 0.47 to 0.62). Similarly, the Virus class experienced a substantial +21.9% enhancement. This statistically proves that the CNN successfully acted as a high-level spatial filter, identifying localized blocks of API calls (*n*-grams) that define the unique execution signature of each threat, thereby delivering a purified semantic tensor to the subsequent LSTM layers.

Furthermore, while the proposed model robustly maintained the detection baseline for highly complex, underrepresented families (such as Spyware at 0.55 F1-Score), the global Weighted Average F1-Score rose by nearly 10% (from 0.61 to 0.67). This demonstrates that the hybrid architecture did not simply memorize majority classes to artificially inflate global metrics; instead, it distributed its cognitive capacity effectively across the dataset's inherent structural asymmetry.

Through this empirical experimentation, it is definitively concluded that raw, unfiltered sequences of API calls inherently saturate the temporal memory mechanisms of purely recurrent networks, regardless of extensive training times. The hybrid CNN-LSTM architecture proposed and validated in this chapter elegantly overcomes this deficit. By processing malware behavior spatially before mapping its narrative temporally, the model establishes a highly viable, computationally efficient, and robust paradigm for multiclass threat categorization in complex, real-world security scenarios.

5 Conclusion

The detection and classification of malicious software based on the behavioral analysis of API calls represents one of the most prominent cybersecurity challenges today. The structural complexity of these threats, combined with the inherent structural asymmetry of real-world datasets, demands machine learning mechanisms capable of abstracting deep behavioral patterns from a vast volume of semantic noise. In this context, this research aimed to empirically evaluate deep learning architectures, proposing a hybrid model (CNN-LSTM) to overcome the predictive limitations of purely recurrent networks established in the reference literature.

The experiments conducted using the dataset provided by Catak et al. (2020) (CATAK et al., 2020) demonstrated that the methodological approach based exclusively on single-layer LSTM networks suffers from critical vulnerabilities in deep feature extraction. The rigorous replication of the reference baseline revealed that, despite an exhaustive 407minute training regimen, the global accuracy plateaued at 61.0%. It was observed that the isolated recurrent neural network struggled to process lengthy, unfiltered temporal sequences of API calls. This resulted in severe semantic confusion and cross-misclassification, particularly among complex families such as Trojans and Droppers, as the network failed to disambiguate overlapping behavioral footprints.

To mitigate this representational deficit, the hybrid architecture (CNN-LSTM) developed and evaluated in this study introduced deep 1D-convolutional blocks as a preliminary spatial filtering phase prior to temporal processing, coupled with an algorithmic class weighting strategy. The empirical results definitively validated the central hypothesis of this monograph: the proposed hybrid model achieved a superior global accuracy of 67.1% in merely 12.4 minutes of computational training time. This result not only surpassed the baseline's predictive capabilities but did so while demanding less than 3% of its computational processing cost.

The hybridization proved exceptionally effective in isolating the distinct behavioral signatures (*n*-grams) of highly complex, underrepresented families. The substantial relative F1-Score growth observed in the Dropper (+31.9%) and Virus (+21.9%) classes attests that the topological logic of malware execution strictly requires spatial dimensionality reduction. By allowing the CNN to filter semantic noise, the temporal memory mechanisms of the LSTM are protected from contextual gradient saturation. Ultimately, the CNN-LSTM model constitutes a highly viable, computationally efficient, and fundamentally more robust paradigm for multiclass threat categorization in complex, real-world scenarios.

References

ABADI, M.; AGARWAL, A.; BARHAM, P.; BREVDO, E.; CHEN, Z.; CITRO, C.; CORRADO, G. S.; DAVIS, A.; DEAN, J.; DEVIN, M. et al. Tensorflow:
Large-scale machine learning on heterogeneous distributed systems. **arXiv preprint arXiv:1603.04467**, 2016. Cited on page 20.

AMER, E.; ZELINKA, I. A dynamic windows malware detection and prediction method based on contextual understanding of api call sequence. **Computers & Security**, Elsevier, v. 92, p. 101760, 2020. Cited on page 18.

CATAK, F. O.; YAZI, A. F.; ELEZAJ, O.; AHMED, J. Deep learning based sequential model for malware analysis using windows exe api calls. **PeerJ computer science**, PeerJ Inc., v. 6, p. e285, 2020. Cited 12 times on pages 6, 11, 12, 15, 19, 21, 25, 26, 27, 29, 33, and 39.

GARCIA, S.; GRILL, M.; STIBOREK, J.; ZUNINO, A. An empirical comparison of botnet detection methods. **computers & security**, Elsevier, v. 45, p. 100–123, 2014. Cited on page 17.

GIBERT, D.; MATEU, C.; PLANES, J. The rise of machine learning for detection and classification of malware: Research developments, trends and challenges. **Journal of Network and Computer Applications**, Elsevier, v. 153, p. 102526, 2020. Cited on page 16.

HOCHREITER, S.; SCHMIDHUBER, J. Long short-term memory. **Neural computation**, MIT press, v. 9, n. 8, p. 1735–1780, 1997. Cited 2 times on pages 18 and 20.

KOLOSNJAJI, B.; ZARRAS, A.; WEBSTER, G.; ECKERT, C. Deep learning for classification of malware system call sequences. In: SPRINGER. **Australasian joint conference on artificial intelligence**. [S.l.], 2016. p. 137–149. Cited 2 times on pages 18 and 19.

OKTAVIANTO, D.; MUHARDIANTO, I. **Cuckoo malware analysis**. [S.l.]: Packt Publishing Birmingham, 2013. Cited on page 29.

OPREA, A.; LI, Z.; YEN, T.-F.; CHIN, S. H.; ALRWAIS, S. Detection of early-stage enterprise infection by mining large-scale log data. In: IEEE. **2015 45th Annual IEEE/IFIP International Conference on Dependable Systems and Networks**. [S.l.], 2015. p. 45–56. Cited on page 17.

QI, X.; JOHAR, M. G. M.; KHATIBI, A.; THAM, J.; ZHANG, R.; MAO, S. Research on malicious software detection based on cnn-lstm hybrid model. In: IEEE. **2023 5th International Conference on Machine Learning, Big Data and Business Intelligence (MLBDBI)**. [S.l.], 2023. p. 300–306. Cited 2 times on pages 12 and 20.

SIHWAIL, R.; OMAR, K.; ARIFFIN, K. Z. A survey on malware analysis techniques: Static, dynamic, hybrid and memory analysis. **Int. J. Adv. Sci. Eng. Inf. Technol**, v. 8, n. 4-2, p. 1662–1671, 2018. Cited on page 17.

SIKORSKI, M.; HONIG, A. **Practical malware analysis: the hands-on guide to dissecting malicious software**. [S.l.]: no starch press, 2012. Cited on page 15.

SZOR, P. **The art of computer virus research and defense**. [S.l.]: Pearson Education, 2005. Cited on page 15.

TAHA, B.; DIAS, J.; WERGHI, N. Convolutional neural networkasa feature extractor for automatic polyp detection. In: IEEE. **2017 IEEE International Conference on Image Processing (ICIP)**. [S.l.], 2017. p. 2060–2064. Cited on page 12.

YE, Y.; LI, T.; ADJEROH, D.; IYENGAR, S. S. A survey on malware detection using data mining techniques. **ACM Computing Surveys (CSUR)**, ACM New York, NY, USA, v. 50, n. 3, p. 1–40, 2017. Cited on page 17.