



Article

MalBERTv2: Code Aware BERT-Based Model for Malware Identification

Abir Rahali and Moulay A. Akhloufi *

Perception, Robotics, and Intelligent Machines (PRIME), Department of Computer Science,
Université de Moncton, Moncton, NB E1A 3E9, Canada

* Correspondence: moulay.akhloufi@umoncton.ca

Abstract: To proactively mitigate malware threats, cybersecurity tools, such as anti-virus and anti-malware software, as well as firewalls, require frequent updates and proactive implementation. However, processing the vast amounts of dataset examples can be overwhelming when relying solely on traditional methods. In cybersecurity workflows, recent advances in natural language processing (NLP) models can aid in proactively detecting various threats. In this paper, we present a novel approach for representing the relevance and significance of the Malware/Goodware (MG) datasets, through the use of a pre-trained language model called MalBERTv2. Our model is trained on publicly available datasets, with a focus on the source code of the apps by extracting the top-ranked files that present the most relevant information. These files are then passed through a pre-tokenization feature generator, and the resulting keywords are used to train the tokenizer from scratch. Finally, we apply a classifier using bidirectional encoder representations from transformers (BERT) as a layer within the model pipeline. The performance of our model is evaluated on different datasets, achieving a weighted f1 score ranging from 82% to 99%. Our results demonstrate the effectiveness of our approach for proactively detecting malware threats using NLP techniques.

Keywords: malware detection; natural language processing; transformer-based model



Citation: Rahali, A.; Akhloufi, M.A. MalBERTv2: Code Aware BERT-Based Model for Malware Identification. *Big Data Cogn. Comput.* **2023**, *7*, 60. <https://doi.org/10.3390/bdcc7020060>

Academic Editors: Tim Schlippe and Matthias Wölfel

Received: 30 January 2023

Revised: 4 March 2023

Accepted: 10 March 2023

Published: 24 March 2023



Copyright: © 2023 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<https://creativecommons.org/licenses/by/4.0/>).

1. Introduction

A series of studies have been conducted to tackle cybersecurity threats using machine learning (ML) and deep learning (DL) tools. Multiple studies have been directed toward analyzing malware at static, dynamic, and hybrid levels [1] to extract diverse features, such as application programming interface (API) calls, permissions, and binaries. For instance, the use of DL algorithms has assisted security specialists in analyzing complex cyberattacks. DL models comprise several layers of an ML algorithm that are capable of learning high-level abstractions from enormously complex datasets. This makes DL algorithms more effective in identifying patterns and detecting malicious activities [2]. Consequently, DL has enabled many cybersecurity companies to improve the accuracy of their malware detection systems. With the exponential growth of NLP applications, DL algorithms, and specifically transformer-based (TB) architectures, have gained popularity in solving complex problems. These algorithms have shown significant advantages over other traditional ML techniques due to their ability to learn the context of the data and their capacity to process large datasets [3].

NLP is an area of artificial intelligence (AI) that works with language data that is based on text. It provides a multitude of tools for putting innovative cybersecurity-related solutions into practice. To find weaknesses in the infrastructure, NLP can find overlaps in data from a company's tech stack and threat streams. The ultimate goal of NLP is to locate the keyword, read the text, and comprehend any relevant context. Despite the fact that all of these tasks are now manual, an automated system is desperately needed. Recently, using attention weights [4] in language modeling improved the transfer learning tasks in AI-based domains. Bidirectional encoder representation from transformers (BERT) [5] is one of

the state-of-the-art TB models. BERT is an innovative, widely implemented solution used in both academia and industry. These transfer learning-based approaches have opened an opportunity to fine tune custom datasets for specific domains with pre-trained models on larger datasets. TB models have a deeper understanding of language context than traditional single-direction language models [4].

In this study, we introduce a novel approach for detecting malware using MalBERT [6,7]. We extend prior work by using larger datasets for training and modeling Android malware (AM) detection as a binary text classification problem. Our proposed method utilizes the software application's source code as a set of features and employs text preprocessing techniques to extract relevant information, such as intents and activities. We conduct extensive experiments on a preprocessed Android dataset collected from publicly available resources. In our previous study [7], we presented the architecture of an automatic system for identifying threat information and detecting malware using MalBERT. In this work, we aim to quantify the subjective relevance of text documents and their potential significance, which can be tailored to meet code-aware needs using existing NLP techniques. We propose a novel approach for identifying malware-specific content from filtered texts, and our major contributions include the following:

1. We propose MalBERTv2, an improved version of the MalBERT approach for malware detection representation by creating a full pipeline for a code-aware pre-trained language model for MG detection.
2. We propose a pre-tokenization process to present the features.
3. We apply extensive experiments and evaluations on a variety of datasets collected from public resources.

2. Related Work

One of the major threats on the internet today is malicious software attacks. Malware detection, visualization, and classification are one of the main areas of research to solve this issue. Today's era requires a system for the automatic classification of malware without de-compiling or obfuscating the code. This paper review DL methods used to classify malware in NLP. These methods focus on parsing and extracting useful information from natural languages to simplify human-computer interaction. The key to the success of NLP in cybersecurity is the availability of large datasets. Textual data in cybersecurity come from various sources, such as emails [8,9], transaction logs from different systems, source code and online social networks. Using NLP techniques has a direct impact on situational awareness from logs of different network events and user activities. Several methods are implemented for representing text in digital form, such as vector space models and distributed representation. Encoding text at the word or character level comprises preprocessing, followed by encoding as an initial step. This includes data cleaning and the transformation of unnecessary and unknown words or characters. Non-sequential and sequential inputs are the two main types of text representation. Bag of words (BoWs) [10], term-document matrices (TDMs) [11] and term frequency-inverse document frequency (TFIDF) matrices belong to the non-sequential representation. N-gram, Keras embedding, Word2vec [12], Neural-Bag-of-words, and FastText [13] belong to sequential representation, which can extract similarities in word meaning. In the cybersecurity domain, capturing sequential information is more important than word sense similarities, as most data contain temporal and spatial information. Therefore, DL approaches are adopted for effective malware detection.

Malicious software attacks are a significant threat on the internet, and malware detection, visualization, and classification are important areas of research in addressing this issue. With the increasing need for automatic malware classification without de-compiling or obfuscating the code, deep learning (DL) methods have been applied to classify malware in natural language processing (NLP). These methods focus on parsing and extracting useful information from natural language to facilitate human-computer interaction. The availability of large datasets is crucial for the success of NLP in cybersecurity, as textual data

in cybersecurity come from diverse sources, including emails, transaction logs, source code, and online social networks. Various methods for representing text in digital form, such as vector space models and distributed representation, are implemented. Preprocessing involves encoding text at the word or character level, which includes data cleaning and transformation of unnecessary and unknown words or characters. Two main types of text representation are non-sequential and sequential inputs. While non-sequential representation techniques, such as bag of words, term document matrices, and term frequency-inverse document frequency matrices are useful, sequential representation techniques, such as N-gram, Keras embedding, Word2vec, neural bag of words, and FastText, are better suited for capturing sequential information that is essential in the cybersecurity domain. Therefore, DL approaches are adopted for effective malware detection.

Table 1 shows a comprehensive overview of malware-related works that have utilized natural language processing (NLP) and deep learning models. The table contains information on the methods used, authors, descriptions, data types, and highlights of each work. The methods used include various tokenization and embedding techniques, pre-trained models, and customized learning models. The data types include malware and goodware samples, as well as URLs and executable files. These works demonstrate the effectiveness of NLP and deep learning models in malware detection and classification tasks. The use of pre-trained models and customized learning models has shown promising results in identifying different types of malware with high accuracy. Additionally, the use of attention-based mechanisms and GAN-based methods has improved the ability of these models to extract meaningful features from the data. Despite these advancements, there are still challenges to overcome, including the limitation of the maximum sequence length and the lack of benchmarks for malware/goodware identification. Further research is needed to address these challenges and improve the performance of these models. Overall, the works listed in this overview provide a valuable resource for researchers and practitioners in the field of cybersecurity.

Table 1. List of NLP and deep learning models and methods used for malware-related tasks.

Model	Authors	Description	Methods	Pros	Limitations	Data Type
Spam Detection using NLP	Swetha and Sridevi (2019) [8]	Utilizes natural language processing (NLP) techniques for spam detection in emails.	NLP techniques	Improves accuracy of spam detection compared to traditional methods.	Limited to detecting spam in emails only.	Text
NLP for Cybersecurity	Antonellis et al. (2006) [11], Zhang and Zhang (2010) [10]	Explores different text representation methods such as term-document matrix (TDM), bag-of-words (BoW), and term frequency-inverse document frequency (TFIDF) for cybersecurity applications using NLP techniques.	TDM, BoW, TFIDF	Provides an efficient way of detecting malicious content in large volumes of data.	Performance may be affected by the quality of the data used for training.	Text
Embeddings for Malware Classification	Church and Huang (2017) [12], Mahoney and Chan (2000) [13]	Uses word2vec and FastText for classifying malware based on the similarity of word meanings and subword information.	Word2vec, FastText	Embedding techniques can handle semantic relations and patterns of the malware code.	Limited to identifying similarities between malware samples, may not be effective in identifying new types of malware.	Text
MalBERT for Malware Classification	Rahali et al. (2021) [7]	Utilizes a fine-tuned BERT model, MalBERT, for binary and multi-classification of malware.	BERT	Can detect different types of malware with high accuracy.	Requires a large amount of labeled data for effective training.	Binary, text
URL Classifier using Transformer Model	Rudd et al. (2020) [14]	Implements a URL classifier using the Transformer model trained from scratch.	Transformer model	Effective in detecting malicious URLs with high accuracy.	May require large amounts of computational resources for training and testing.	Text
ALBERT for Traffic Network Classification	Han et al. (2020) [15]	Proposes two methods for traffic network classification using pre-trained ALBERT model and transfer learning.	ALBERT, transfer learning	Achieves high accuracy in traffic network classification tasks.	May require fine-tuning on new data to achieve optimal performance.	Text
I-MAD for Static Malware Detection	Li et al. (2021) [16]	Proposes I-MAD, a deep learning (DL) model for static malware detection using the Galaxy Transformer network.	DL	Achieves high accuracy in static malware detection tasks.	Limited to detecting known malware samples only.	Binary, image

Table 1. Cont.

Model	Authors	Description	Methods	Pros	Limitations	Data Type
Static Analysis for Malware Detection	Jusoh et al. (2021) [17]	Proposes a guide for researchers on detecting malware through static analysis.	Static analysis	Provides a comprehensive guide for researchers to effectively detect malware through static analysis.	May require technical expertise to effectively implement the proposed methods.	Text
Hybrid Analytic Approach for Malware Detection	Srinidhi et al. (2020) [18]	Proposes a framework for big data analysis utilizing both static and dynamic malware detection methods.	Static and dynamic analysis	Combines the strengths of both static and dynamic analysis methods for improved accuracy.	May require a large amount of computational resources for analyzing big data.	Binary, text
Attention-based Detection Model	Choi et al. (2020) [19]	Proposes a technique for extracting harmful file features based on an attention mechanism using API system calls.	Attention mechanism	Can detect malware based on API system calls.	May not be effective in detecting new types of malware.	Text
GAN-based Method for Malware Detection	Cagatay et al. (2021) [20]	Proposes a GAN-based method using API call graphs obtained from malicious and benign Android files.	GAN	Can detect previously unknown types of malware.	May require a large amount of computational resources for training and testing.	Image
HAWK for Adaptive Android Apps	Hei et al. (2021) [21]	Proposes HAWK, a malware detection tool for adaptive Android apps using heterogeneous GANs.	GAN	Can detect previously unknown types of malware.	May require a large amount of computational resources for training and testing.	Image
Attention-based BiLSTM for Malware Detection	Pathak et al. (2021) [22]	Uses two attention-based BiLSTM models to find the most predictive API calls for malware detection.	Attention mechanism, BiLSTM	Can effectively detect malware based on API calls.	May not be effective in detecting new types of malware.	Text
SLAM for Malware Detection	Chen et al. (2020) [23]	Builds a malware detection technique called SLAM on attention methods that use the semantics of API calls.	Attention mechanism	Can detect malware based on semantic analysis of API calls.	May not be effective in detecting new types of malware.	Text

Table 1. Cont.

Model	Authors	Description	Methods	Pros	Limitations	Data Type
Residual Attention-based Method for Malware Detection	Ganesan et al. (2021) [24]	Uses residual attention methods to find malware by focusing on its key features.	Residual attention mechanism	Can effectively detect malware by focusing on key features.	May not be effective in detecting new types of malware.	Text
ATT-CNN-BiLSTM for Identifying DGA Attacks	Ren et al. (2020) [25]	Proposes ATT-CNN-BiLSTM, a DL framework for identifying domain generation algorithm (DGA) attacks.	DL, ATT, CNN, BiLSTM	Can effectively identify DGA attacks with high accuracy.	Limited to identifying DGA attacks only.	Text
DeepRan for Ransomware Classification	Lao et al. (2021)	Proposes DeepRan, which uses a fully connected layer and an attention-based BiLSTM for the classification of ransomware.	Attention mechanism, fully connected layer, BiLSTM	Achieves high accuracy in ransomware classification tasks.	Limited to classifying ransomware only.	Text
System Designs for Malware Classification	Rupali et al. (2020) [26]	Proposes a survey of the category of malware images.	Image processing techniques	Provides a comprehensive survey of the category of malware images.	Limited to analyzing images of malware samples.	Image
Executable Files Analysis of Malware Samples	Singh et al. (2021) [27]	Conducts analysis of executable files of malware samples.	Dynamic analysis	Can effectively analyze executable files of malware samples.	May require a large amount of computational resources for analyzing large datasets.	Binary
Ensemble Model for Malware Classification	Kouliaridis et al. (2021) [28]	Proposes an ensemble model that combines static and dynamic analysis.	Static and dynamic analysis, ensemble learning	Can effectively detect different types of malware with high accuracy.	May require a large amount of computational resources for training and testing.	Binary
DL Model for Malware Detection	Syed et al. (2021) [29]	Proposes DeepAMD, a DL model for malware detection.	DL	Achieves high accuracy in detecting different types of malware.	Requires a large amount of labeled data for effective training.	Binary, image
Customized Learning Models for AM Detection	Amin et al. (2020) [30]	Proposes an anti-malware system that uses customized learning models.	Customized learning models	Can effectively detect different types of malware with high accuracy.	May require a large amount of computational resources for training and testing.	Binary, image

Table 1. Cont.

Model	Authors	Description	Methods	Pros	Limitations	Data Type
PetaDroid for AM Detection	Karbab et al. (2021) [31]	Proposes PetaDroid, a static analysis-based method for detecting AM.	Static analysis	Can detect previously unknown types of malware.	May not be effective in detecting new types of malware.	Binary
Performance Comparison of Pre-trained CNN Models	Pooja et al. (2022) [32]	Conducts a performance comparison of 26 pre-trained CNN models in AM detection.	Pre-trained CNN models	Provides a comprehensive comparison of different pre-trained CNN models.	May require a large amount of computational resources for training and testing.	Image
Anomaly Detection Approach	Chong et al. (2022) [33]	Proposes an anomaly detection approach based on a two-head neural network.	Neural network	Can effectively detect anomalies in malware samples.	May require a large amount of computational resources for training and testing.	Binary, image
GNN-based Method for AM Detection	Weng Lo et al. (2022) [34]	Proposes a GNN-based method for AM detection by capturing meaningful intra-procedural call path patterns.	GNN	Can detect previously unknown types of malware.	May require a large amount of computational resources for training and testing.	Binary, image

2.1. Deep Learning-Based Methods

Researchers proposed several studies and research projects in academia to solve the problem of malware identification and classification. Rupali et al. [26] studied a considerable number of previous research papers covering the research characteristics of system designs for the malware classification technique. They proposed a survey of the category of malware images, while Singh et al. [27] focused on analyzing executable files of malware samples. Kouliridis et al. [28] combined static and dynamic analysis. They introduced an ensemble model by averaging the output of all base classification models per malware instance separately. Syed et al. [29] proposed DeepAMD, a DL model for malware detection. Their experiments showed that DeepAMD outperformed other approaches in detecting and identifying malware attacks on both static and dynamic layers for the malware category classification, and malware family classification. Amin et al. [30] proposed an anti-malware system that uses customized learning models, which detect and attribute the AM via opcodes extracted from application byte-code. The results show that bidirectional long short-term memory (BiLSTMs) neural networks can detect the static behavior of AM.

Karbab et al. [31] proposed PetaDroid, a static analysis based method for detecting AM. The method also apply clustering of malware families. The framework makes use of novel techniques built on top of NLP, such as an ensemble of convolutional neural networks (CNNs). Pooja et al. [32] presented a performance comparison of 26 state-of-the-art pre-trained CNN models in AM detection. It also included the performance obtained by large-scale learning with support vector machine (SVM) and random forest (RF) classifiers and stacking with CNN models. Based on their results, an EfficientNet-B4 CNN-based model can accurately detect AM using image-based malware representations of the Android DEX file. Chong et al. [33] proposed an anomaly detection approach based on a two-head neural network. The model identify the time developed samples that the previously trained DL models misclassified. In addition, Weng Lo et al. [34] proposed a graph neural networks (GNNs) based method for AM detection by capturing meaningful intra-procedural call path patterns. In addition, a jumping-knowledge technique applies to minimize the effect of the over-smoothing problem, which is common in GNNs.

2.2. Attention-Based Methods

AI methods based on attention mechanisms have significantly advanced the field of NLP, and as a result, the applications relating to textual-based cybersecurity. Choi et al. [19] suggested a technique for extracting harmful file features based on an attention mechanism using API system calls. Their results demonstrated that this strategy outperformed two common baselines: a skip-connected long short-term memory-based detection model and a CNN-based detection model. Cagatay et al. [20] employed API call graphs obtained from malicious and benign Android files in a graph attention network (GAN) to detect malware threats. Hei et al. [21] introduced HAWK, a malware detection tool for adaptive Android apps using heterogeneous GANs. In order to express implicit higher-order links, they employed the Android entities and behavioral relationships as a heterogeneous information network (HIN). Pathak et al. [22] proposed a study that used two attention-based BiLSTM model to find the most predictive API calls. They discovered a set of API calls that could aid the community in discovering new malware signatures. Chen et al. [23] proposed a malware detection technique called SLAM built on attention methods that use the semantics of API calls. Based on the semantics and structure data of the API execution sequence, the characteristics were retrieved. They looked into the execution sequence's properties and categorized them into 17 groups.

Ganesan et al. [24] utilized residual attention methods to find malware. They used the global data that were taken from a picture, known as GIST features, to compare the model with CNN-based techniques and standard ML algorithms. The suggested a strategy concentrated on drawing attention to the malware's key features that let it stand out from safe files, thus lowering the number of false positives. Ren et al. [25] suggested ATT-CNN-BiLSTM, a DL framework for identifying domain generation algorithm (DGA) attacks to

identify the danger. By producing different network locations, DGA is utilized to confirm the responsible points to the command-and-control servers. The weight of the collected deep information from the domain names is assigned by the attention layer. The CNN and BiLSTM neural network layers extract the features from the domain sequence data. Lao et al. [35] proposed DeepRan, which uses a fully connected layer and an attention-based BiLSTM for the classification of ransomware. By adding a conditional random field (CRF) model to the attention-based BiLSTM, it additionally labels anomalous activity as one of the potential ransomware attacks. They took high-dimensional host logging data and extracted semantic information using the term frequency-inverse document frequency (TFIDF) approach. The attention mechanism is the main architectural emphasis of TB models, and the attention block performs its computations repeatedly in parallel. Each of these is called an attention head.

2.3. Transformer-Based Methods

Few works have applied the TB [4] architecture in the domain of cybersecurity. Our first paper, MalBERT [7] showed interesting performance results when fine tuning BERT to classify malware for both binary and multi-classification. Rudd et al. [14] implemented a URL classifier into malicious and benign by training the transformer model from scratch. They indicated that auxiliary auto-regressive loss improved the model performance. Han et al. [15] proposed two methods for traffic network classification, first using the pre-train Albert model with unlabeled traffic and then fine tuning the model with labeled traffic data. Second, transfer the Albert pre-trained language model with language data and fine tune the model with the labeled network traffic. They revealed that the ALBERT network traffic model pre-trained with network traffic data has faster convergence speeds, higher accuracy rates, and fewer false alarms. Li et al. [16] proposed I-MAD, a DL model for static malware detection using the Galaxy Transformer network. It can understand assembly code at the basic block, function, and executable levels. It can also provide interpretation for its detection results, locate malicious payloads, and find consistent patterns in malware samples. Jusoh et al. [17] focused on static analysis for malware detection. They presented a guide for researchers by proposing novel methods to detect malware through static analysis. They discussed the articles published from 2009 until 2019 and analyzed the steps in static analysis, reverse engineering, features, and classification.

Srinidhi et al. [18] proposed a framework for big data analysis utilizing both static and dynamic malware detection methods. They used the two methods to categorize and locate zero-day malware. On sample binary files comprising several different malware families, they tested the framework. They created a subset of three candidate features for static analysis using permissions and intents as static features and three feature selection techniques [36–38]. Using the training multi-feature data, they eventually applied the suggested hybrid analytic approach to identify AM and categorize the samples into families. Using auto-encoders as a generative model, Mahmood et al. [39] proposed a feature learning model for cybersecurity tasks that learns a latent representation of various feature sets. By using the feature vector, the auto-encoders were able to extract a code vector that accurately reflected the semantic similarity between the feature vectors. Later, they developed an unsupervised model for identifying malware.

In this research, we expand a language model developed on MG datasets and concentrate primarily on binary classification. We also offer a workaround for the sequence length restrictions by putting forth a fresh pre-tokenization technique. We place more emphasis on the malware and goodware samples' textual representations. Compared to other evaluated baselines, our suggested technique performs the best, according to our data. Complex processes must necessarily be used for static and dynamic evaluations. For malware identification, the feature representation phase of the procedure is still a research priority. Due to the popularity of datasets based on Android, many more studies concentrated on AM identification tasks. In order to improve feature representation, we did actually combine the most recent datasets and fine-tune BERT utilizing a brand-new pre-tokenization.

3. Proposed System Architecture

Since this research extends our previous work [7], the dataset used before will be used to test and compare the improvements in the model architecture. We build the model in two phases: training and testing. The training process is distinct from the prediction or testing phase. We used a training–validation set to build a multilevel MalBERTv2 classifier, which is then evaluated on a separate test set at a latter phase.

3.1. Data Creation

The data creation phase passes through the data selection and ranking for the most useful datasets. Since there are no known benchmarks for malware analysis, we collected the datasets proposed online with extracted features in different formats. We also downloaded the samples from the collected Android package file (APK) list using the Androzoo platform. We then extracted the most important files to pass through the feature generator. Figure 1 provides an overview of the data collection process for a research study. The figure shows two levels of preprocessing: Level 1 handles feature-based datasets (FBs), while Level 2 deals with data extracted from the state-of-the-art sets. For Level 2, the data is passed through VirusTotal to check labeling and then the Manifest.xml files are extracted. The goodwill APKs are collected from Google Play and then processed through the two levels of preprocessing. At the end of the process, text files are generated for each sample.

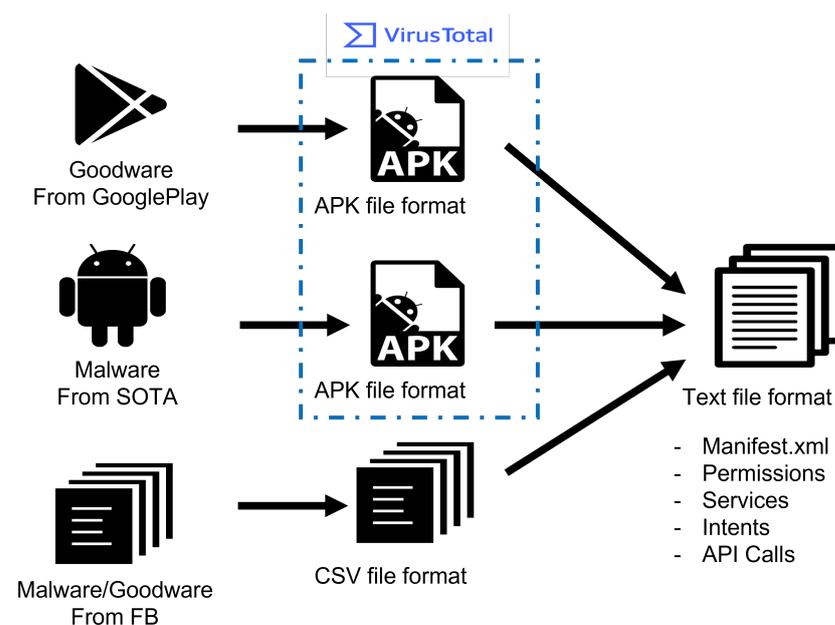


Figure 1. Overview of the data collection process. We collected the goodwill APKs from Google Play [40] and set two levels of preprocessing. Level 2 handling the data extracted from the collection of the state-of-the-art sets, we passed these data through VirusTotal [41] to check the labeling, then we extracted the *Manifest.xml* files. Level 1 handling feature-based datasets (FBs), where we passed directly to the reformatting phase. The final samples are text files for each sample.

3.2. Feature Creation Module

The feature creation has two levels, as shown in Figure 2. These levels depend on the source of the collected datasets. The publicly available AM datasets have two different formats.

- The datasets that the researchers share with the samples in APK format come first, where every sample has a distinct hash identifier that serves as a kind of fingerprint. Malware is typically identified via a technique called hashing. A hashing application is used to run the malicious software, producing a distinct hash that serves as the malware's identification.

- Second, depending on the extraction method they suggested, the dataset authors share the preprocessed features. These characteristics were primarily displayed as CSV files.

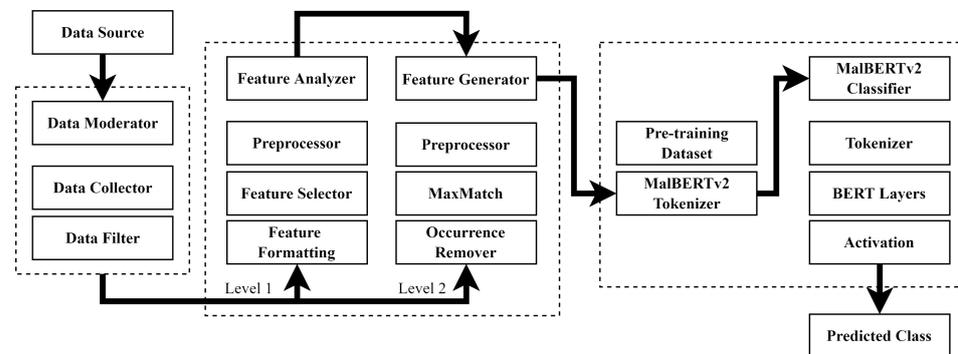


Figure 2. Overview of the proposed MalBERTv2 approach, Level 1 handling feature-based datasets, where reformatting of the samples is performed. Level 2 is handling the data extracted from the collection of state-of-the-art sets. The pretraining dataset is used to train the tokenizer. We set the predicted class probability threshold to 0.5. We label the classes as 1 for malware and 0 for goodware.

We configured our feature generator on two levels to handle the various formats in order to cover larger datasets. The first stage in NLP analysis is the extraction of features and representative keywords. TFIDF is the keyword extraction algorithm that is most frequently employed. A variety of techniques are available, ranging from tokenization utilizing learned language models, such as BERT, to word embeddings. In order to assess the performance of our suggested strategy and compare it to it, we built several degrees of feature representation in this work. In addition to developing our own unique feature generator, we used TFIDF, Fasttext word embeddings trained on Wikipedia datasets, and the pre-trained BERT representation.

3.2.1. Tokenization

During the preprocessing and tokenization phase, raw text is first split into words or subwords, which are then converted to unique integer IDs through a lookup table. Tokenization is a crucial step in natural language processing and machine learning tasks that involve text. There are three main types of tokenizers used in transformer-based models, which include byte-pair encoding (BPE) [42], WordPiece (linear time WordPiece tokenization and fast WordPiece tokenization [43]), and SentencePiece [44]. The BertTokenizer class provides a higher-level interface that includes the BERT token splitting algorithm and a WordPieceTokenizer. It takes sentences as input and returns token IDs. One limitation of BERT is the maximum sequence length of 512 tokens. Sequences shorter than the maximum length require padding with [PAD] tokens, while longer sequences must be truncated. Our previous work [7] addresses this limitation. Contextualized embeddings in BERT provide a representation of words that depends on the sentence's position, leading to distinct clusters corresponding to word senses. This characteristic showed success in word sense disambiguation tasks. However, the extent to which BERT can capture patterns in malware datasets requires further investigation.

3.2.2. MalBERTv2 Feature Analyzer

To detail the specific nature of the datasets, we proposed an initial tokenizer on top of the transformer encoder tokenizer. Splitting a code text into smaller chunks is a more difficult task than it appears, and there are several methods for doing so. The coding syntax varies depending on the programming language used. This is a sound first step. We notice that the punctuation is not properly attached to the words. We should consider punctuation so that a model does not have to learn a different representation of a word and every punctuation symbol that might come after it. However, it is dis-

advantageous how the tokenization handled the word "android.permission.INTERNET" stands for "android" "permission" "INTERNET", so it would be better tokenized as ["android", "permission", "INTERNET"]. Depending on the rules we apply for tokenizing a text, a different tokenized output is generated for the same text.

Table 2 shows an example of the samples after preprocessing. A pre-trained model only performs properly if you feed it an input that was tokenized with the same rules that were used to tokenize its training data. A big vocabulary size forces the model to have an enormous embedding matrix as the input and output layer, which causes both increased memory and time complexity. We use predefined algorithms to customize the tokenization process for our dataset. First, there is the MaxMatch [45] algorithm that stands for the maximum matching algorithm, which extracts the maximum matched words that exist in the provided dictionary at the start from the relevant language acting as a knowledge base of ground truth words. Additionally, Sennrich et al. [46] introduced the BPE, a neural machine translation of rare words with sub-word units. After pre-tokenization, a set of unique words are created, and the frequency of each word that occurred in the training data is determined.

Table 2. Example of sample after applying the preprocessing without the occurrences remover module.

Original File	Preprocessed File
<pre><?xml version="1.0" encoding="utf-8" standalone="no"?> <manifest xmlns:android="http://schemas.android.com/APK/res/android" package="com.lbcsoft.subway"> <uses-permission android:name="android.permission.INTERNET"/> <uses-permission android:name="android.permission.CALL_PHONE"/> <application android:allowBackup="true" android:debuggable="true" android:icon="@drawable/icon_youke_subway" android:label="@string/app_name" android:theme="@android:style/Theme.NoTitleBar"></pre>	<pre>xml version encoding utf standalone nomanifest xmlns android http schemas android com APK res android package com lbcsoft subway uses permission android name android permission internet uses permission android name android permission call phone application android allow backup true android debuggable true android icon drawable icon youke subway android label string app name android theme android style theme no title bar</pre>

Next, BPE creates a base vocabulary comprising all symbols that occur in the set of unique words and learns to merge rules to form a new symbol from two symbols in the base vocabulary. BPE is already used by the BERT tokenizer, so our pre-tokenizer is a first level-specific word moderator. The proposed tokenizer, as shown in Figure 3 applies preprocessing methods to clean the code text and keeps only the useful keywords. Additionally, we use the MaxMatch algorithm, as shown in the algorithm of the MalBERTv2 tokenizer. Then, an occurrences remover is added. This module is detailed in Section 4. The full algorithm of the tokenizer is presented in Algorithm 1.

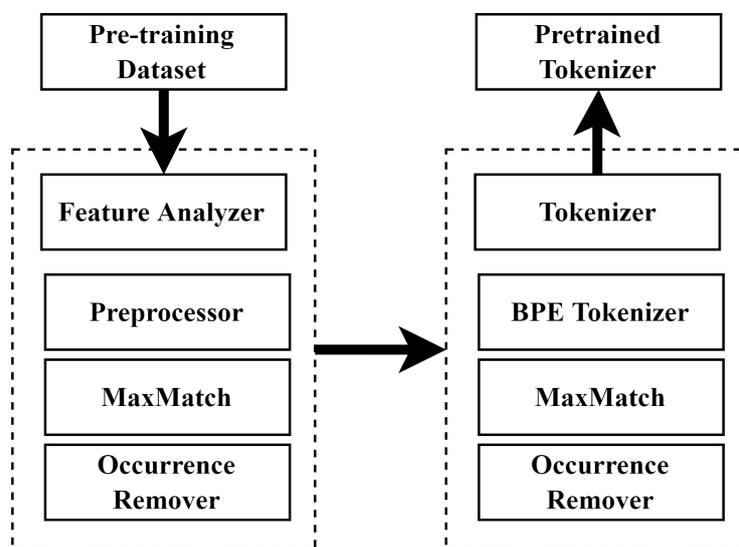


Figure 3. Overview of the feature creation phase using the feature generator then applying the MalBERTv2 tokenizer.

Algorithm 1 Proposed feature generator pre-tokenizer. We used both the MaxMatch [45] algorithm and BPE [42] in the tokenization process. We collected the given dictionary manually after processing the unique words in the collected pretraining datasets.

```

1: procedure SEGMENT STRING C INTO UNIQUE WORD LIST W USING DICTIONARY D.
2:   C ← input string
3:   W ← output tokens list
4:   D ← given dictionary
5:   BPE ← Byte-Pair Encoding
6:   OR ← OccurrencesRemover Module
7:   Loop:
8:   while C is not empty do
9:     Find longest match w in D from start of C
10:    Condition:
11:    if w is not empty then
12:      C ← C − w.
13:      W ← W + w.
14:    else
15:      Remove first character from C and add to W.
16:    end if
17:    BPE(W).
18:  end while
19: return OR(W).
20: end procedure

```

To evaluate the previous pre-trained tokenizers, we compared their coverage percentages for the collected dataset. Figures 4 and 5 show TB tokenizers and others, such as Fasttext and Glove. The best TB tokenizer is the BERT-uncased pretrained model, just after the Fasttext embeddings. However, all these models can only cover about 50% of the existent vocabulary of the datasets. The used embeddings are namely BERT-base-uncased, BERT-base-cased, BERT-base-multilingual-cased, BERT-base-multilingual-uncased, RoBERTa-base-vocab, GPT2-xl-vocab, XLM-mlm-en-2048, Word2Vec (GoogleNews-vectors), Glove (glove.6B.300d) and Fasttext (wiki-en). Figures 4 and 5 show the embeddings coverage for the whole datasets samples text and the datasets' vocabulary after preprocessing.

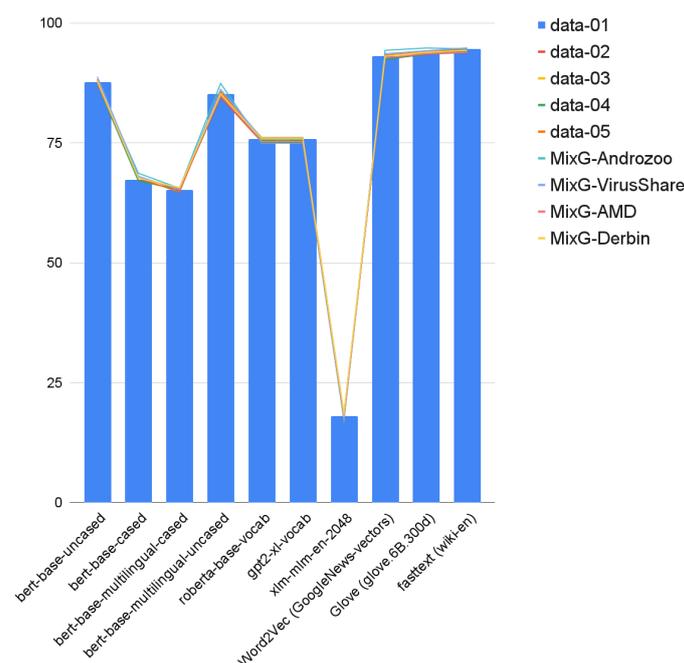


Figure 4. Percentage % of embeddings coverage for the whole datasets samples text after preprocessing.

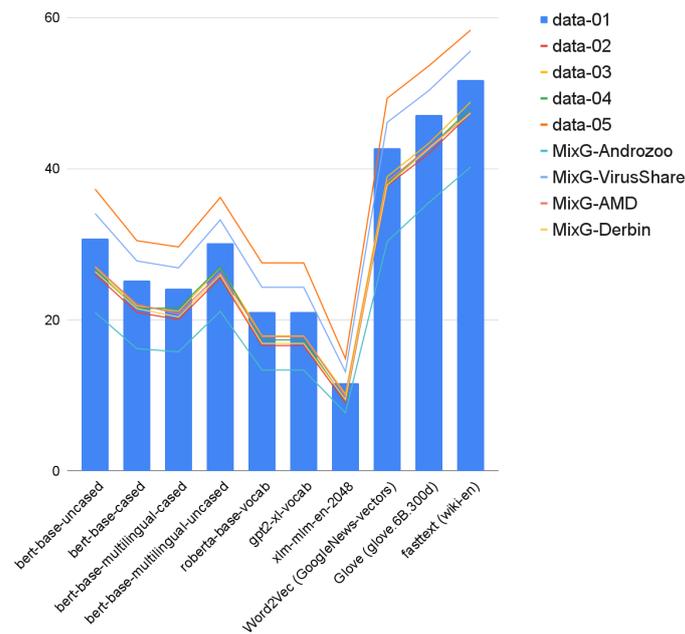


Figure 5. Percentage % of embeddings coverage for the datasets’ vocabulary after preprocessing.

3.3. Model Creation

The MalBERTv2 framework comprises a TB architecture for MG classification. It is designed as a general-purpose malware classifier system. The model building and training process are distinct from the prediction or testing phase. We use a different dataset for the training and validation step for both the tokenizer and the layers to build the final classifier model, which is then evaluated on a separate test dataset. The model as Figure 6 shows the training of the tokenizer used to map the features for the BERT-based layer inside the main model.

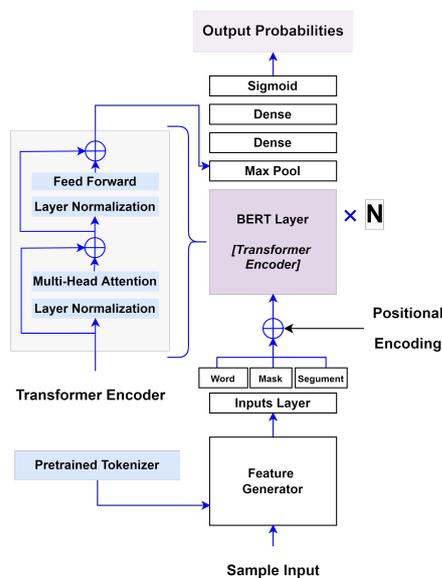


Figure 6. Overview of the pre-training proposed approach based on BERT-like block embeddings and representations. N is the number of transformer blocks. For the classification, we use only the encoder block of the transformer architecture.

The model takes the generated features and fine tunes the model. The BERT base model uses transformer blocks and a number of self-attention heads. For every input

token in a sequence, each head computes *key*, *value*, and *query* vectors used to create a weighted representation. The outputs of all heads in the same layer are combined and run through a fully connected layer. Each layer is wrapped with a skip connection and followed by layer normalization. The first layer of BERT receives as input a combination of the token, segment, and positional embeddings. Let X be the total number of instances with M malware and G goodwill samples, where the M samples possess a label $L = 1$ denoting malware and the G samples from X possess the label $L = 0$ denoting goodwill or benign. All X samples are extracted as a full-text file containing the features. We applied the level 2 feature generator on the features. Finally, the samples are represented as text file F representations.

MalBERT focuses on attention layers as presented in Figure 7. Attention sees their input as a set of vectors, with no sequential order. This model also does not contain any recurrent or convolutional layers. Transformers are built to process sequential input data, much like recurrent neural networks (RNNs). However, unlike RNNs, transformers do not process data in order but use positional encoding (PE). PE is added to give the model some information about the relative position of the tokens in the sentence. The PE vector is added to the embedding vector. Embeddings represent a token in a d -dimensional space where tokens with similar meaning will be closer to each other. However, the embeddings do not encode the relative position of tokens in a sentence. The formula for calculating the PE is as shown in Equations (1) and (2). The attention function used by the transformer takes three inputs: Q (*query*), K (*key*), V (*value*) as shown in Equation (3). The dot product attention is scaled by a factor of the square root of the depth. This is done because, for large values of depth, the dot product grows large in magnitude, pushing the softmax function to where it has small gradients. BERT uses the Adam optimizer with a custom learning rate schedule, as shown in Equation (4):

$$PE_{(pos,2i)} = \sin(pos/10000^{2i/d_{model}}) \tag{1}$$

$$PE_{(pos,2i+1)} = \cos(pos/10000^{2i/d_{model}}) \tag{2}$$

$$Attention(Q, K, V) = softmax_k \left(\frac{QK^T}{\sqrt{d_k}} \right) V \tag{3}$$

$$lrate = d_{model}^{-0.5} * \min(step_num^{-0.5}, step_num \cdot warmup_steps^{-1.5}) \tag{4}$$

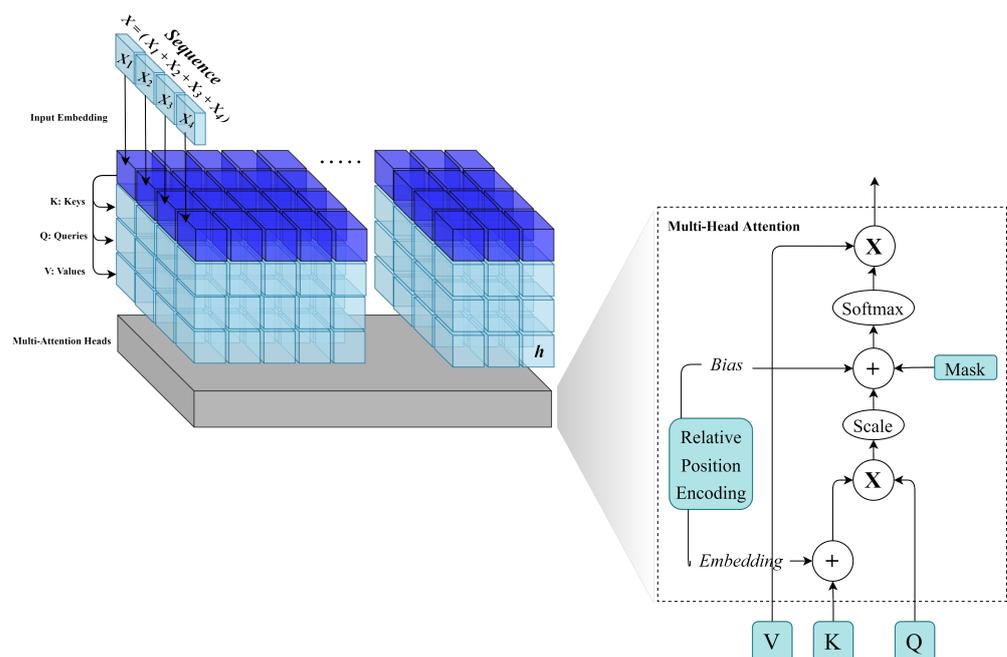


Figure 7. Overview of the attentions mechanism used in MalBERT, where h is the number of heads.

4. Implementation

In this section, we detail the implementation steps for our proposed approach. We start by the dataset details, then the feature engineering steps, and finally we detail the occurrences removal method.

4.1. Datasets

The performance of the pre-trained models is largely determined by the scale and quality of datasets used for training. Therefore, we need a large-scale scanned document dataset to pre-train the MalBERTv2 model. We finally selected 85,000 apps from the following datasets distributed as Figure 8 shows.

- AMD dataset [47]. It contains 24,553 samples categorized in 135 varieties among 71 malware families ranging from 2010 to 2016. The dataset is publicly shared with the research community.
- Drebin dataset [48]. It contains 5560 applications from 179 different malware families. The samples were collected from August 2010 to October 2012 and were made available by the MobileSandbox project [48]. The authors made the datasets from the project publicly available to foster research on AM and to enable a comparison of different detection approaches.
- VirusShare dataset [49]. It is a repository of malware samples that gives security researchers access to live malicious code samples. It is a static data dataset obtained from the VirusShare malware repository. The samples were gathered using the AM datasets debiasing guidelines [50].
- Androzoo dataset [51]. It is a growing collection of Android apps gathered from various sources, including the official Google Play app market, with the goal of facilitating Android-related research. It currently contains over 15 million APKs, each of which was or will be analyzed by tens of different anti-virus products to determine which applications are malware. Each app contains more than 20 different types of metadata, such as VirusTotal reports.

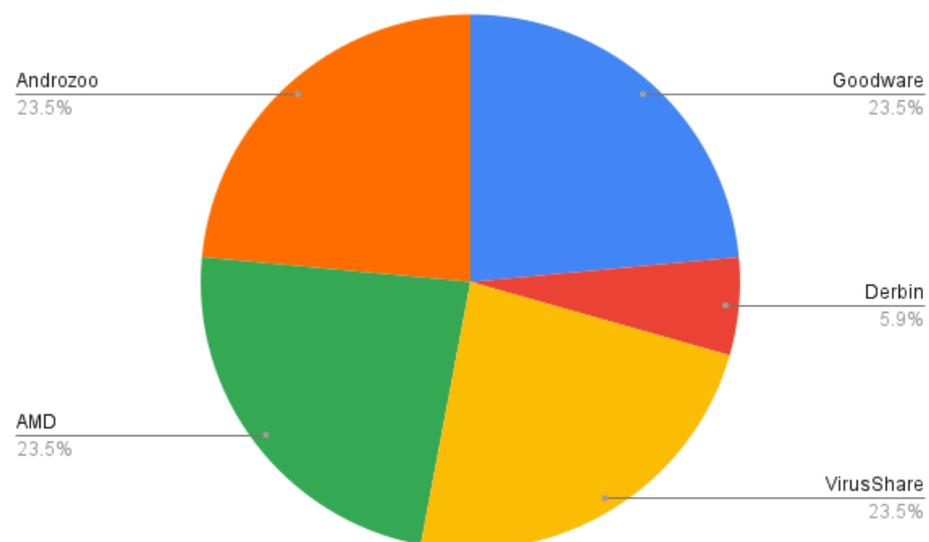


Figure 8. Pre-training. Datasets Sizes.

Fine tuning datasets can be summarized into two types: state-of-the-art datasets, including MixG-Androzoo, MixG-VirusShare, MixG-AMD, and MixG-Derbin, and feature-based datasets (FBs), including D01, D02, D03, D04, D05.

- Android permissions and API calls during dynamic analysis [52]. This dataset includes 50,000 Android apps and 10,000 malware apps gathered from various sources. We note this dataset as **D01**.
- Android malware detection [53]. These data contain APKs from various sources, including malicious and benign applications. They were created after selecting a sufficient number of apps. Using the pyaxmlparser and Androguard [54] framework, we analyze each application in the array. On the set of each feature, we used a binary vector format, and in the last column labeled class, we marked it 1 (Malicious) or 0 (Benign). We note this dataset as **D02**.
- Android malware dataset for machine learning [55]. These data contain 215 feature vectors extracted from 15,036 applications: 5560 malware apps from the Drebin project and 9476 benign apps. The dataset was used to develop and test the multilevel classifier fusion approach for AM detection. The supporting file contains a more detailed description of the feature vectors or attributes discovered through static code analysis of Android apps. We note this dataset as **D03**.
- Android malware and normal permissions dataset [56]. These data contain 18,850 normal android application packages and 10,000 malware android packages, which are used to identify the behavior of malware applications on the permission they need at run time. We note this dataset as **D04**.
- Android permission dataset [57]. These data contain android apps and their permissions. They are classified as 1 (Malicious) or 0 (Benign). We note this dataset as **D05**.

We indeed collected samples from the state-of-the-art datasets as suggested by DADA [58] labeling guideline. These samples are a mix of goodware and malware samples. The authors proposed these APK lists to solve the problem of biased malware datasets. The collected datasets are, namely **MixG-Androzoo**, **MixG-VirusShare**, **MixG-AMD**, and **MixG-Derbin**. Figure 9 shows the distribution of all the test datasets and their main features also the publication dates of the apps.

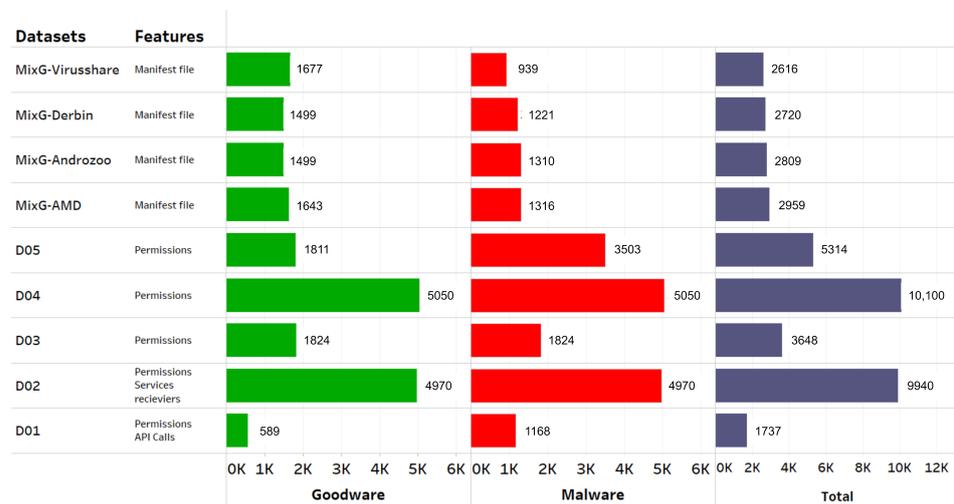


Figure 9. Fine-tuning datasets.

4.2. Preprocessing and Feature Representations

The data preprocessing phase is the same as in MalBERT [7], where it comprises basic punctuation removal. Once the list of APKs is defined, we can write a script to download the files. Then, we decompiled the downloaded APKs using Jdax [59] and Apktool [60], which creates folders of the apps’ files [61]. We extracted the *Manifest.xml* file from each sample. This file presents essential information about the application, including the list of permissions, the activities, services, broadcast receivers, content providers, the version, and the meta-data. These files are then parsed to text format and passed through the

preprocessing phase. In this step, we apply specific cleaning of the not important, mostly repeated words. We manually analyzed different examples and created a list of words and expressions that do not provide additional information. The purpose of the preprocessing is to reduce the size of the input to the limit of tokens specified by the transformer. The final dataset format has three columns, the *ID* column, represented by the APK hash name, the *text* column representing the manifest files after preprocessing, the *label* column, a binary format equal to 1 if the app is malware, and 0 if not, while for the rest of the datasets, since the CSV of the features are binary. We concatenate the names of the features into the text if the feature exists.

4.3. Occurrences Remover

We handle code text, not normal text, so the existence of some keys is more important than the number of times this word shows up in the file. Based on this hypothesis, we remove all the occurred words. We assume that a keyword existence is more important than key word occurrence. Figure 10 shows the density of the document words inside the different documents of the dataset, while Figure 11 shows the density after applying the proposed tokenizer. Now, 98% of the samples fit under the limitation of the 512 tokens thresholds.

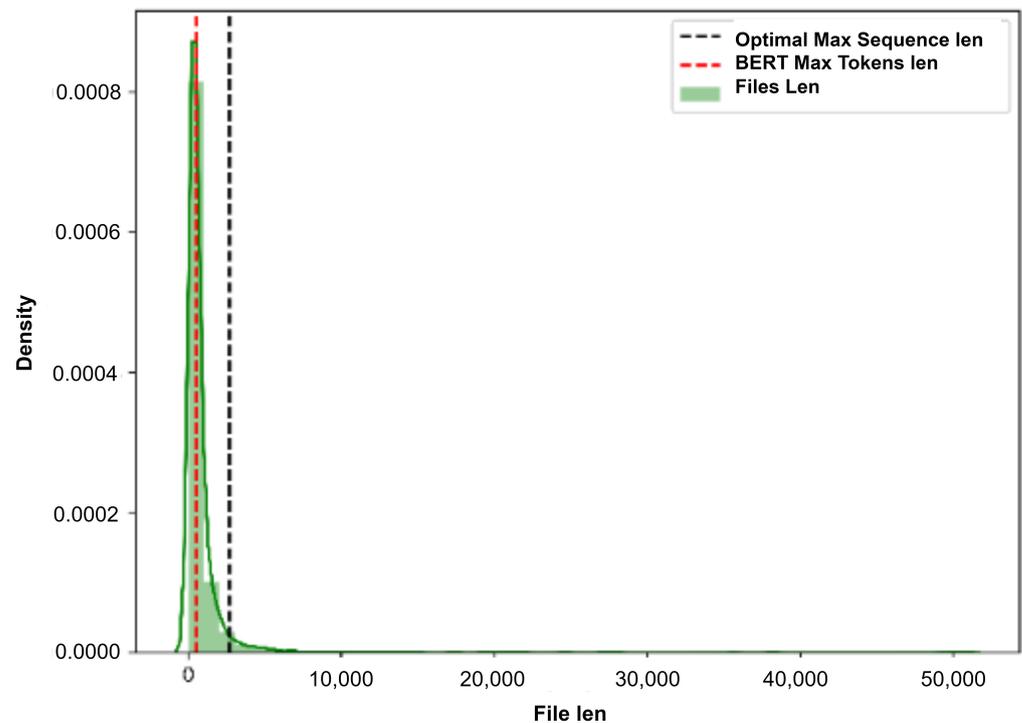


Figure 10. Files lengths before applying the proposed preprocessing.

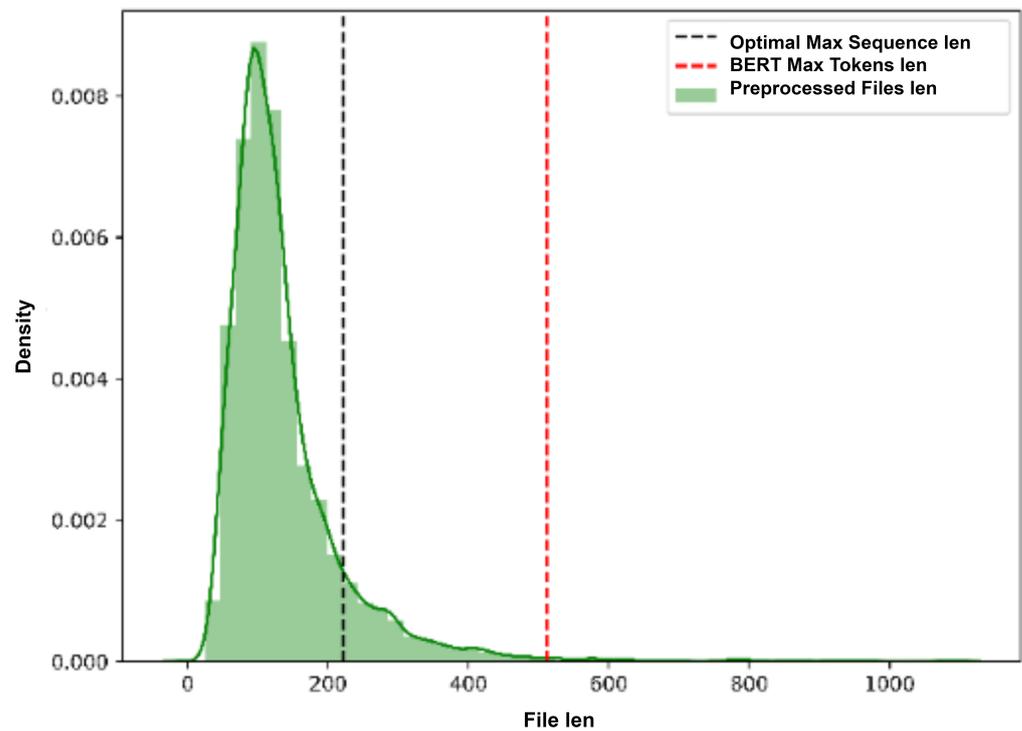


Figure 11. Files lengths after applying the proposed preprocessing.

5. Experimental Results

This section focuses on the evaluation of the conducted experiment's results using proposed evaluation metrics. To compare our proposed approach, we define baselines based on related studies. Evaluation metrics are defined to assess the performance of the proposed approach and the baselines. The chosen metrics are carefully selected to ensure that they accurately reflect the performance of the models. Finally, the experimental results are presented and analyzed to evaluate the proposed approach's effectiveness in detecting malware samples.

5.1. Baselines

In this study, we compare the performance of our MalBERTv2 model to several state-of-the-art malware detection models previously reported in the literature. Specifically, the models selected for comparison include an SVM model with TFIDF feature representation, a CNN model with Fasttext pre-trained embeddings, MalBERTv1, a transformer layer model with TFIDF features, and our proposed MalBERTv2 model. The chosen models represent a variety of machine learning approaches and feature representations commonly employed in malware detection tasks. By comparing the performance of our model to these established approaches, we aim to provide a comprehensive evaluation of its effectiveness and highlight any potential advantages or limitations.

5.1.1. TFIDF + SVM

We conducted a preliminary experiment using a basic machine learning model (SVM) and a simple text representation (TFIDF) to evaluate the malware datasets. However, analyzing these datasets poses a significant challenge as the goodware samples are typically duplicates or distinct from the collected malware samples in terms of time or package types. This limitation makes it challenging to achieve accurate and reliable results in malware detection and classification tasks.

5.1.2. Fasttext + CNN

Fasttext [62] is a widely used library for learning word representations and sentence classification. A fundamental idea in modern machine learning is to represent words as vectors that capture hidden information about language, such as word analogies or semantics, improving the performance of text classifiers. Fasttext's wiki-en model embeddings are trained on Wikipedia articles, making it an attractive choice for natural language processing tasks. Figures 4 and 5 demonstrate that Fasttext provides better coverage than other pre-trained embeddings, further motivating our selection of this model. In this study, we propose a simple CNN model consisting of a 1D convolutional network and two dense layers for malware classification. Previous related works [63,64] reported promising results using CNN-based models. Therefore, it is crucial to compare our proposed approach's performance with these state-of-the-art models.

5.1.3. MalBERTv1

MalBERT is a fine-tuned BERT model that is specifically designed for malware classification. The model is trained on a large corpus of malware and goodware samples, with the goal of identifying and differentiating between the two classes. MalBERT achieves state-of-the-art performance on several benchmark datasets, including Androzoo, Derbin, AMD, and VirusShare. MalBERT's architecture is based on BERT, a pre-trained transformer-based language model that is widely used in natural language processing tasks. However, unlike BERT, MalBERT is trained on a dataset of malware and goodware samples, which makes it more effective for malware classification tasks. MalBERT's training process involves fine tuning the pre-trained BERT model on a large corpus of malware and goodware samples, followed by training a classification layer on top of the fine-tuned BERT model. The MalBERT model has several advantages over traditional signature-based malware detection methods. It can detect previously unknown types of malware and is effective in identifying different types of malware with high accuracy. Furthermore, the model can handle large amounts of unstructured text-like data, making it a useful tool for cybersecurity professionals. Figure 12 that shows an overview of the MalBERTv1 methodology steps is a graphical representation of the process of identifying and classifying malware samples using the MalBERTv1 approach and Figure 13 show the model fine-tuning process in MalBERTv1.

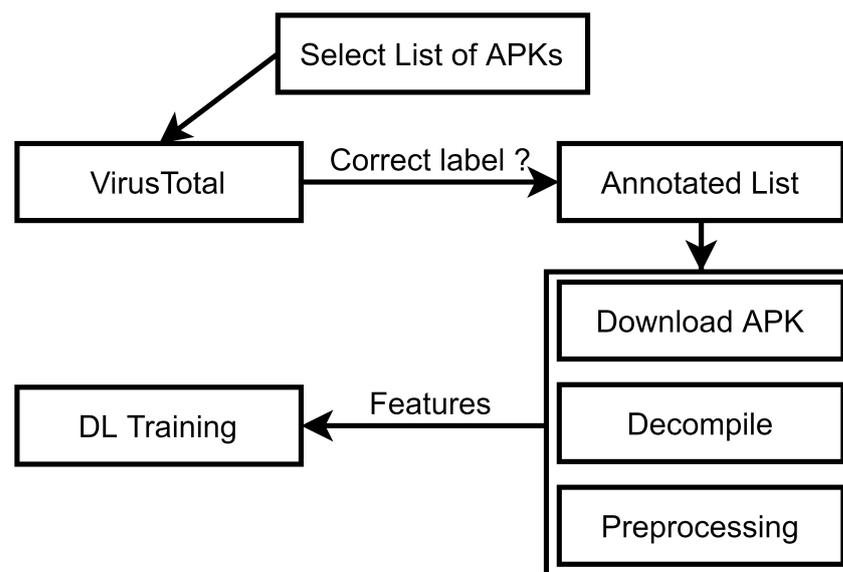


Figure 12. Overview of MalBERTv1 methodology steps [7].

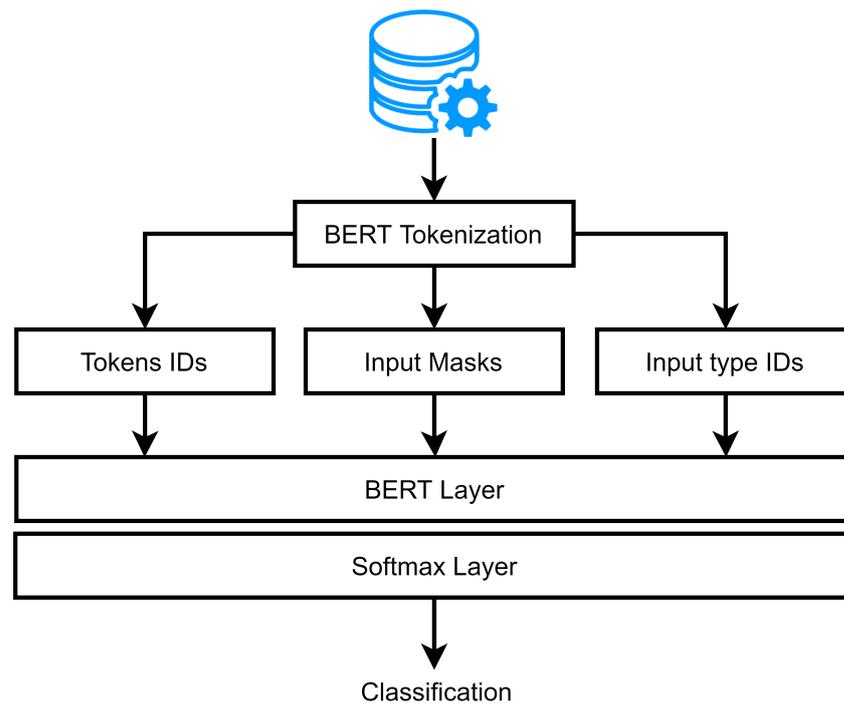


Figure 13. BERT model fine-tuning process in MalBERTv1 [7].

In our study, we utilized the MalBERT implementation described in [7]. The MalBERT model fine-tunes the BERT model using a specific feature representation at the beginning of the process. To address the 512-token limit of BERT, our approach reorders the text by prioritizing the most important words and selecting the first 512 tokens. BERT [5] is a neural network architecture that consists of a stack of transformer blocks. The transformer blocks utilize self-attention to establish relationships between words in the input sequence and generate meaningful embeddings.

5.1.4. TFIDF + Transformer from Scratch

The primary objective of this baseline is to evaluate the effectiveness of training the transformer encoder from scratch using basic word tokenization, specifically the term frequency-inverse document frequency (TFIDF) method. The transformer block is constructed using multi-head attention, feed-forward, and normalization techniques. For this experiment, we set the number of heads to two and added the transformer block as a layer to two fully connected layers. To perform binary classification, a sigmoid activation function is applied. This experimental setup allows us to compare the performance of our proposed model against a baseline method and determine the extent to which the additional features and techniques improve the model's performance.

5.2. Training MalBERTv2

As illustrated in Figure 2, the training process comprises several phases, including data collection and feature creation, which are described in detail in Section 4. In the subsequent sections, we focus on the final phase of model creation.

5.2.1. Train MalBERTv2 Tokenizer

In this study, we are dealing with files written in various programming languages, rather than traditional English text with structured paragraphs and continuous context. Figure 11 demonstrates that the vocabulary coverage percentages of state-of-the-art datasets using transformer-based (TB) or traditional deep learning embeddings are below 50%, indicating the inadequacy of these models for our specific domain. To address this issue, we developed a RoBERTa [65]-based tokenizer model trained from scratch to serve as

both the encoder and decoder of our proposed MalBERTv2 model. As our data domain is application specific, it is essential to include relevant words and symbols related to application descriptions, while avoiding irrelevant general vocabulary from other domains. To achieve this, we created a code-aware specific vocabulary for the tokenizer. The training phase of our model can be described in four main steps:

1. Apply the feature generator on the dataset and use the generated features as input to the tokenizer. The feature generator can be considered as an initial tokenizer since it applies tokenization to the original text to obtain the most relevant and English-related words without losing the most important keywords in the files.
2. Create and train a byte-level byte-pair encoding tokenizer with the same special tokens as RoBERTa.
3. Train the defined RoBERTa model from scratch using masked language modeling (MLM).
4. Save the tokenizer to map the features of the test datasets later to fine-tune the MalBERTv2 classifier.

5.2.2. Train MalBERTv2 Classifier

To build our model for the classification task, we added two untrained dense layers of neurons at the end of BERT and fine tuned it. We initialized the weight of the MalBERTv2 model with the pre-trained BERT base model, which already encodes a lot of information about our training data vocabulary. This allowed us to quickly fine tune the model by mapping the features to the proposed tokenizer. Fine tuning BERT has been shown to achieve state-of-the-art results with minimal task-specific adjustments for the malware identification task, as demonstrated in [7]. This approach is preferable to implementing custom and sometimes obscure architectures that work well on a specific task. The parameters of our model include the pre-trained base uncased BERT model weights for the BERT block layer, a maximum sequence length of 512, a batch size of 8, and a learning rate for the Adam optimizer [4] with a custom learning rate scheduler according to Formula (4) at the final sigmoid activation function, which is set to $2e - 5$. These parameters were carefully chosen to ensure the best performance of our model.

5.3. Evaluation Metrics

To assess the model's effectiveness and avoid the contingency caused by the partitioning of the training and test sets. The models are evaluated using the following performance metrics: accuracy, Matthews correlation coefficient (*MCC*), precision (*mc*), recall (*mc*), F1-score (*mc*), and AUC are common classification problem metrics. The malware class is denoted by the (mc) tag. We base our assessment on the model's ability to detect malware patterns. We use the terms *TP* (true positives), *FN* (false negatives), *FP* (false positives), and *TN* (true negatives). The accuracy (*ACC*) metric is used to evaluate classification models. It is calculated by dividing the number of correct predictions by the total number of predictions. *MCC* is used for binary classification with an unbalanced dataset. It has a range of -1 to $+1$. We chose *MCC* over the F1-score for binary classification as recommended in Chicco et al. [66]. The formula for the standard F1-score is the harmonic mean of precision and recall. A perfect model has an F-score of 1. We used macro-averaging in the results we presented in this paper. *AUC* [67] is calculated as the area under the math curve. *AUC* ranges in value from 0 to 1. Equations (5)–(9) show the different details of the used metrics.

$$ACC = \frac{TP + TN}{TP + TN + FP + FN} \quad (5)$$

$$Precision = \frac{TP}{TP + FP} \quad (6)$$

$$Recall = \frac{TP}{TP + FN} \quad (7)$$

$$F1 = \frac{2 * TP}{2 * TP + FP + FN} \quad (8)$$

$$MCC = \frac{TP * TN - FP * FN}{\sqrt{(TP + FP) \cdot (TP + FN) \cdot (TN + FP) \cdot (TN + FN)}} \quad (9)$$

5.4. Experiments Results and Discussion

This section outlines the results of experiments conducted to examine the feasibility of using MalBERTv2 to improve accuracy performance through training the transformer tokenizer from scratch. Table 3 presents the test set results of the baseline classifiers alongside those of MalBERTv2 for both datasets.

Table 3. Table of the metrics results for the models for the state-of-the-art collected test sets.

Model	Data	Accuracy	f1 (mc)	mcc	Precision (mc)	Recall (mc)	auc
TFIDF + SVM	MixG-Androzoo	0.969589	0.969805	0.939478	0.957895	0.982014	0.969655
	MixG-VirusShare	0.858803	0.857143	0.717704	0.864964	0.849462	0.858778
	MixG-AMD	0.931127	0.932751	0.863799	0.906621	0.960432	0.931283
	MixG-Derbin	0.935599	0.935018	0.871212	0.938406	0.931655	0.935578
Fasttext + CNN	MixG-Androzoo	0.953488	0.955403	0.906829	0.958692	0.952137	0.953554
	MixG-VirusShare	0.801609	0.763326	0.644415	0.962366	0.632509	0.803596
	MixG-AMD	0.927549	0.933113	0.856619	0.902556	0.965812	0.925683
	MixG-Derbin	0.929338	0.930396	0.860468	0.96	0.902564	0.930644
MalBERT	MixG-Androzoo	0.975689	0.976183	0.971568	0.98651	0.966068	0.976158
	MixG-VirusShare	0.924039	0.924712	0.84808	0.927176	0.922261	0.92406
	MixG-AMD	0.970483	0.971478	0.941157	0.982517	0.960684	0.970961
	MixG-Derbin	0.966905	0.968076	0.933909	0.977352	0.958974	0.967292
TFIDF + Transformer From Scratch	MixG-Androzoo	0.9558	0.954981	0.943421	0.952922	0.963211	0.954092
	MixG-VirusShare	0.9231125	0.925467	0.884563	0.923224	0.927892	0.92343
	MixG-AMD	0.9576809	0.9540987	0.945896	0.977345	0.973099	0.967554
	MixG-Derbin	0.9567821	0.9560983	0.958763	0.967812	0.964398	0.968989
MalBERTv2 = FeatureAnalyzer + MalBERT	MixG-Androzoo	0.990744	0.998341	0.991149	0.99765	0.999033	0.998901
	MixG-VirusShare	0.956782	0.957819	0.945887	0.957164	0.956292	0.944226
	MixG-AMD	0.988787	0.989742	0.961892	0.999834	0.988987	0.985977
	MixG-Derbin	0.988954	0.989645	0.974889	0.998328	0.978884	0.987329

Table 3 presents the evaluation results of five models (TFIDF + SVM, Fasttext + CNN, MalBERT, TFIDF + Transformer From Scratch, and MalBERTv2) for malware identification using four datasets (MixG-Androzoo, MixG-VirusShare, MixG-AMD, and MixG-Derbin). The metrics used to evaluate the models were data accuracy, F1-score, Matthews correlation coefficient (MCC), precision, recall, and area under the curve (AUC). Overall, MalBERTv2 had the highest performance in all the datasets, with an average accuracy of 97.1%, f1 score of 97.2%, MCC of 95.8%, precision of 98.4%, recall of 96.7%, and AUC of 98.6%. MalBERTv2 outperformed all other models in terms of accuracy, f1 score, precision, and recall. TFIDF + Transformer From Scratch had the second-best performance, followed by MalBERT, Fasttext + CNN, and TFIDF + SVM. It is worth noting that the MalBERTv2 model was fine tuned on the mixed collected dataset, which could have contributed to its superior performance in the evaluation. Additionally, the dataset split ratios were 90%:10% for training-validation and testing, which could also have an impact on the performance of the models. Overall, the table presents a clear and organized comparison of the performance of five models for malware identification, providing valuable insights for researchers in the field.

Table 4 presents the performance metrics of different deep learning models applied to classify malware and goodware samples in five collected datasets. MalBERTv2, which combines feature analysis with the MalBERT model, achieved the best performance on all five datasets, with high accuracy, F1-score, precision, recall, and area under the curve (AUC) values. The traditional machine learning approach, TFIDF with SVM, showed lower performance than the deep learning models. The Fasttext + CNN model achieved high accuracy and F1-score on some datasets but relatively low precision and recall rates. The MalBERT model alone performed well on some datasets, but its performance varied depending on the dataset. Overall, the results suggest that combining deep learning models with feature analysis can improve the performance of malware detection systems, and MalBERTv2 is a promising approach in this regard.

Table 4. Table of the metrics results for the models for the feature-based collected test sets.

Model	Data	Accuracy	f1 (mc)	mcc	Precision (mc)	Recall (mc)	auc
TFIDF + SVM	D01	0.570881	0.721393	0.188245	0.653153	0.805556	0.427469
	D02	0.582226	0.728216	0.122358	0.576176	0.989259	0.520577
	D03	0.582226	0.728216	0.122358	0.576176	0.989259	0.520577
	D04	0.814212	0.838955	0.619591	0.832186	0.845834	0.808882
	D05	0.599373	0.74552	0.123743	0.641096	0.89058	0.463673
Fasttext + CNN	D01	0.617084	0.727308	0.158007	0.627276	0.865297	0.562497
	D02	0.886327	0.989815	0.872379	0.876578	0.844387	0.834099
	D03	0.681754	0.628078	0.59842	0.586972	0.614239	0.607119
	D04	0.888516	0.889971	0.876653	0.883587	0.896437	0.887252
	D05	0.664133	0.798173	0.562563	0.664133	0.758021	0.758021
MalBERT	D01	0.694449	0.734708	0.656146	0.698335	0.951598	0.615904
	D02	0.799747	0.799775	0.799485	0.699775	0.899775	0.899743
	D03	0.798821	0.79815	0.797286	0.698766	0.897535	0.898479
	D04	0.899875	0.79989	0.699745	0.79978	0.899855	0.898855
	D05	0.759333	0.794697	0.659333	0.679373	0.669332	0.568801
TFIDF + Transformer From Scratch	D01	0.623949	0.664798	0.554896	0.688923	0.551898	0.593549
	D02	0.783359	0.742259	0.669237	0.682342	0.789149	0.778833
	D03	0.824719	0.829188	0.779938	0.738336	0.793799	0.812268
	D04	0.903338	0.894773	0.823442	0.813492	0.813457	0.848735
	D05	0.775727	0.764993	0.754489	0.749271	0.773246	0.735923
MalBERTv2 = FeatureAnalyzer + MalBERT	D01	0.824623	0.793342	0.784459	0.824836	0.821458	0.813454
	D02	0.883678	0.857334	0.782653	0.773456	0.889922	0.879653
	D03	0.894577	0.889882	0.848883	0.928921	0.893939	0.881948
	D04	0.937643	0.894388	0.799922	0.923562	0.973252	0.928798
	D05	0.834465	0.834781	0.835549	0.872873	0.873984	0.833654

5.5. MalBERTv2 Performance on Mixed Datasets

We present the evaluation of MalBERTv2 on the mixed collected dataset. Table 4 shows the predictive metrics on the four proposed baselines compared to our proposed approach. The split ratios for the training–validation and testing sets were 90%:10% for all four datasets. From Table 3, MalBERTv2 had the best recall and precision rates for the malware class, while we do not include the rates for the goodware class in the table because we want to focus on the malware identification task. On the training–validation set, the weighted F1-measure for the first baseline for the Androzoo dataset was good, 0.9695, and it improved when testing with MalBERTv2. It is important to note that we did not invest

time to apply a parameter engineering process to the baselines. We focused basically on evaluating the proposed approach's performance.

5.6. Malbertv2 Performance on Feature-Based Dataset

We present the evaluation of MalBERTv2 on the feature-based collected dataset, the datasets namely are D01 to D05. Table 4 shows the predictive metrics on the four proposed baselines compared to our proposed approach. The split ratios for the training-validation and testing sets were also 90%:10% for all four datasets. It is important to note also that these datasets have a smaller document size than the previous datasets. Since the text created is created by combining the names of the features that exist in the provided feature vectors. In this, also the statement of feature existence is more important than feature occurrences, and both feature generation levels are applied on the datasets. From Table 4, MalBERTv2 improves the accuracy and weighted F-measure compared to the other baselines for the five datasets, and the accuracy ranges between 0.8224 and 0.9376. One of the main advantages of our proposed approach is its flexibility in introducing any format of application code as the desirable features. The results of the feature-based datasets illustrate this advantage.

5.7. Analysis of Time Performance

As previously mentioned, we employed a Python-based feature extractor tool described in our previous work [7] to extract features from the apps. The processing time required to extract features from apps depends on their size, which can range from a few kilobytes to several megabytes. On average, it took 70 s to unzip and disassemble an app using Apktool, while the average time to analyze the manifest and generate the features was 10 s. Therefore, the total average processing time for an app was approximately 80 s. During our experiments, we fed the feature vectors obtained from the tool into the models for testing. After loading both the tokenizer and the MalBERTv2 model, the testing time for classification of the feature vectors was 0.5 s on a single GPU machine.

5.8. Analysis of Baselines Performance

We can conclude based on the evaluation results presented in Tables 3 and 4 that for the proposed preprocessing, based on the evaluation results presented in Tables 3 and 4, it can be concluded that the proposed preprocessing technique using the feature analyzer in MalBERTv2 enhances the feature representation and helps the model to focus on the most relevant patterns by utilizing the different stages of the tokenization process. The use of a simple machine learning model resulted in poor performance compared to the rest of the baselines. The TFIDF + SVM approach achieved the best accuracy of 0.88 for D04, while it was only 0.81 using SVM alone. To further improve performance, we applied a CNN-based model to the dataset using Fasttext embeddings for feature representation. As shown in Table 3, this model outperformed the machine learning model, and utilizing TFIDF n-grams with a transformer layer from scratch improved the accuracy by approximately 0.2%. MalBERTv1 exhibited the best performance, outperforming all other baselines.

5.9. Qualitative Analysis

In this study, we conducted a comprehensive evaluation of the quality of the datasets used. The datasets were collected in multiple stages from various sources. Only reliable datasets were selected for analysis, and a first baseline model was employed to eliminate datasets with limited sizes or those yielding high classification results with simple training, where the malware and goodware are different. To further evaluate the quality of the data, we split the MixG-VirusShare dataset into malware and goodware subsets. We then applied a knowledge graph on the output text generated by the feature generator, with the occurrences remover disabled to gain a deeper understanding of the dataset characteristics. This approach allowed us to assess the dataset quality and identify any potential issues or biases that may impact our analysis.

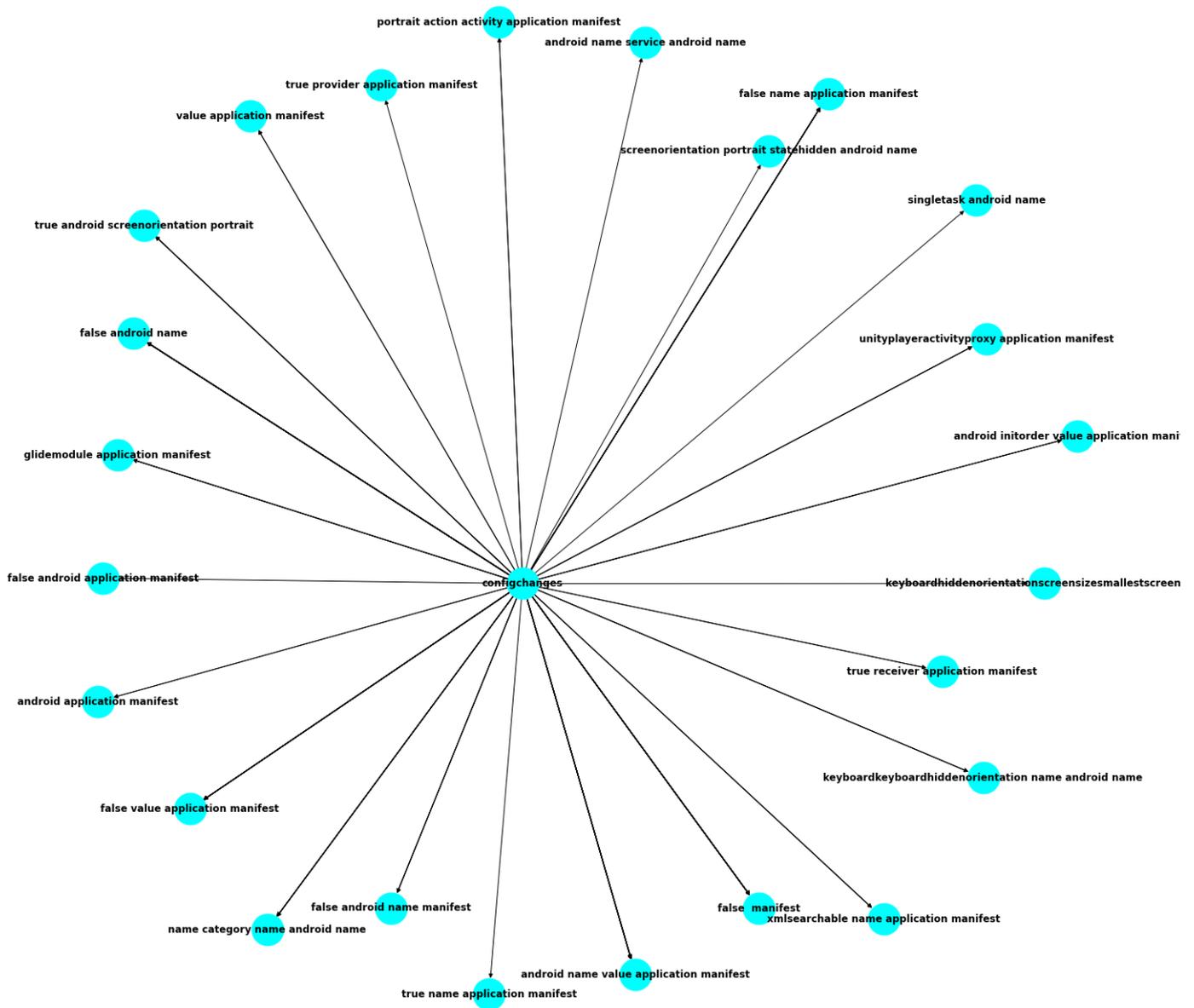


Figure 17. Example of the same connection by the "ConfigChanges" entity for the MalBERTv2 goodwill dataset knowledge graph.

6. Conclusions

In this paper, we proposed a novel approach to tokenize the data sources extracted from malware and goodwill datasets. This approach helps to focus on the relevance and significance of unstructured code from different programming languages that presents subjunctive importance to the app features. We believe that our approach could address the problem of processing a massive amount of unstructured text-like malware/goodwill samples for the cybersecurity domain. The idea was to use a feature generator to play the role of a pre-tokenizer for our main classification. We applied this code-aware tokenization process during training and then in the testing phase for mapping the features. The novel feature representation considers the software applications' source code as a set of features. We apply text preprocessing on these features to keep the important information, such as permissions, intents, and activities. We trained from scratch our BERT-based tokenizer with the extracted features of 85,000 samples from different datasets, normally Androzoo, Derbin, AMD, VirusShare and a collection of goodwill samples, where the list is provided by DADA [58]. Finally, we trained the MalBERTv2 classifier; it has a BERT layer block with

the same weights as the pre-trained BERT, and we added a fully connected layer for the prediction probabilities. Combining all these pieces, we present MalBERTv2, an end-to-end malware/goodware language model for malware classification. The combination of the proposed methods had interesting results. Because of the constraints, such as the lack of benchmarks for malware/goodware identification, we could not effectively compare the model with existing methods. Meanwhile, we tried to select the best approaches based on the previous works and set it as baselines for comparison. Additionally, for numerous datasets, researchers provided the extracted features not in the format of logs. We reformatted the ones that include the feature names and occurrences, but the rest of datasets that do not fit our requirements were eliminated at the datasets collection step. Besides addressing these flows, in the future, we will extend the research by improving the full platform execution time and reducing the feature generation and prediction process. Finally, the whole system could run a full cycle in less time.

Author Contributions: Conceptualization, M.A.A. and A.R.; methodology, A.R. and M.A.A.; validation, A.R. and M.A.A.; formal analysis, A.R. and M.A.A.; writing—original draft preparation, A.R.; writing—review and editing, M.A.A.; funding acquisition, M.A.A. All authors have read and agreed to the published version of the manuscript.

Funding: This research was enabled in part by support provided by the Natural Sciences and Engineering Research Council of Canada (NSERC), funding reference number RGPIN-2018-06233.

Data Availability Statement: All data used in this work are available publicly. See Section 4.1 for the sources of the used data.

Conflicts of Interest: The authors declare no conflict of interest.

Abbreviations

The following abbreviations are used in this manuscript:

AM	Android Malware
API	Application programming interface
APK	Android Package File
BERT	Bidirectional Encoder Representations from Transformers
BiLSTM	Bidirectional Long Short-Term Memory
BoW	Bag of Words
BPE	Byte Pair Encoding
CNN	Convolutional Neural Networks
CRF	Conditional Random Field
DL	Deep Learning
FB	Feature-Based
GAN	Graph Attention Network
HIN	Heterogeneous Information Network
MG	Malware/Goodware
ML	Machine Learning
NLP	Natural Language Processing
RF	Random Forest
SVM	Support Vector Machine
TB	Transformer Based
TDM	Term Document Matrices
TFIDF	Term Frequency Inverse Document Frequency

References

1. Damodaran, A.; Di Troia, F.; Visaggio, C.A.; Austin, T.H.; Stamp, M. A comparison of static, dynamic, and hybrid analysis for malware detection. *J. Comput. Virol. Hacking Tech.* **2017**, *13*, 1–12. [[CrossRef](#)]
2. MahdaviFar, S.; Ghorbani, A.A. Application of deep learning to cybersecurity: A survey. *Neurocomputing* **2019**, *347*, 149–176. [[CrossRef](#)]

3. Karita, S.; Chen, N.; Hayashi, T.; Hori, T.; Inaguma, H.; Jiang, Z.; Someki, M.; Soplín, N.E.Y.; Yamamoto, R.; Wang, X.; et al. A comparative study on transformer vs rnn in speech applications. In Proceedings of the 2019 IEEE Automatic Speech Recognition and Understanding Workshop (ASRU), Singapore, 14–18 December 2019; pp. 449–456.
4. Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A.N.; Kaiser, Ł.; Polosukhin, I. Attention is all you need. In Proceedings of the Advances in Neural Information Processing Systems 30 (NIPS 2017), Long Beach, CA, USA, 4–9 December 2017; pp. 5998–6008. Available online: <https://papers.nips.cc/paper/2017/hash/3f5ee243547dee91fbd053c1c4a845aa-Abstract.html> (accessed on 13 March 2023).
5. Devlin, J.; Chang, M.W.; Lee, K.; Toutanova, K. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv* **2018**, arXiv:1810.04805.
6. Rahali, A.; Akhloufi, M.A. MalBERT: Using transformers for cybersecurity and malicious software detection. *arXiv* **2021**, arXiv:2103.03806.
7. Rahali, A.; Akhloufi, M.A. MalBERT: Malware Detection using Bidirectional Encoder Representations from Transformers. In Proceedings of the 2021 IEEE International Conference on Systems, Man, and Cybernetics (SMC), Melbourne, Australia, 17–20 October 2021; pp. 3226–3231.
8. Swetha, M.; Sarraf, G. Spam email and malware elimination employing various classification techniques. In Proceedings of the 2019 4th International Conference on Recent Trends on Electronics, Information, Communication & Technology (RTEICT), Bengaluru, India, 17–18 May 2019; pp. 140–145.
9. Mohammad, R.M.A. A lifelong spam emails classification model. *Appl. Comput. Informatics* **2020**, ahead-of-print. [[CrossRef](#)]
10. Zhang, Y.; Jin, R.; Zhou, Z.H. Understanding bag-of-words model: A statistical framework. *Int. J. Mach. Learn. Cybern.* **2010**, *1*, 43–52. [[CrossRef](#)]
11. Antonellis, I.; Gallopoulos, E. Exploring term-document matrices from matrix models in text mining. *arXiv* **2006**, arXiv:cs/0602076.
12. Church, K.W. Word2Vec. *Nat. Lang. Eng.* **2017**, *23*, 155–162. [[CrossRef](#)]
13. Mahoney, M.V. Fast Text Compression with Neural Networks. In Proceedings of the FLAIRS Conference, Orlando, FL, USA, 22–24 May 2000; pp. 230–234.
14. Rudd, E.M.; Abdallah, A. Training Transformers for Information Security Tasks: A Case Study on Malicious URL Prediction. *arXiv* **2020**, arXiv:2011.03040.
15. Han, L.; Zeng, X.; Song, L. A novel transfer learning based on albert for malicious network traffic classification. *Int. J. Innov. Comput. Inf. Control.* **2020**, *16*, 2103–2119.
16. Li, M.Q.; Fung, B.C.; Charland, P.; Ding, S.H. I-MAD: Interpretable Malware Detector Using Galaxy Transformer. *Comput. Secur.* **2021**, *108*, 102371. [[CrossRef](#)]
17. Jusoh, R.; Firdaus, A.; Anwar, S.; Osman, M.Z.; Darmawan, M.F.; Ab Razak, M.F. Malware detection using static analysis in Android: A review of FeCO (features, classification, and obfuscation). *PeerJ Comput. Sci.* **2021**, *7*, e522. [[CrossRef](#)]
18. Niveditha, V.; Ananthan, T.; Amudha, S.; Sam, D.; Srinidhi, S. Detect and classify zero day Malware efficiently in big data platform. *Int. J. Adv. Sci. Technol.* **2020**, *29*, 1947–1954.
19. Choi, S.; Bae, J.; Lee, C.; Kim, Y.; Kim, J. Attention-based automated feature extraction for malware analysis. *Sensors* **2020**, *20*, 2893. [[CrossRef](#)]
20. Catal, C.; Gunduz, H.; Ozcan, A. Malware Detection Based on Graph Attention Networks for Intelligent Transportation Systems. *Electronics* **2021**, *10*, 2534. [[CrossRef](#)]
21. Hei, Y.; Yang, R.; Peng, H.; Wang, L.; Xu, X.; Liu, J.; Liu, H.; Xu, J.; Sun, L. Hawk: Rapid android malware detection through heterogeneous graph attention networks. *IEEE Trans. Neural Netw. Learn. Syst.* **2021**, 1–15. [[CrossRef](#)]
22. Pathak, P. Leveraging Attention-Based Deep Neural Networks for Security Vetting of Android Applications. Ph.D. Thesis, Bowling Green State University, Bowling Green, OH, USA, 2021; Volume 8, Number 29. [[CrossRef](#)]
23. Chen, J.; Guo, S.; Ma, X.; Li, H.; Guo, J.; Chen, M.; Pan, Z. SLAM: A Malware Detection Method Based on Sliding Local Attention Mechanism. *Secur. Commun. Netw.* **2020**, *2020*, 6724513. [[CrossRef](#)]
24. Ganesan, S.; Ravi, V.; Krichen, M.; Sowmya, V.; Alroobaea, R.; Soman, K. Robust Malware Detection using Residual Attention Network. In Proceedings of the 2021 IEEE International Conference on Consumer Electronics (ICCE), Las Vegas, NV, USA, 10–12 January 2021; pp. 1–6.
25. Ren, F.; Jiang, Z.; Wang, X.; Liu, J. A DGA domain names detection modeling method based on integrating an attention mechanism and deep neural network. *Cybersecurity* **2020**, *3*, 4. [[CrossRef](#)]
26. Komatwar, R.; Kokare, M. A Survey on Malware Detection and Classification. *J. Appl. Secur. Res.* **2021**, *16*, 390–420. [[CrossRef](#)]
27. Singh, J.; Singh, J. A survey on machine learning-based malware detection in executable files. *J. Syst. Archit.* **2021**, *112*, 101861. [[CrossRef](#)]
28. Kouliaridis, V.; Kambourakis, G.; Geneiatakis, D.; Potha, N. Two Anatomists Are Better than One—Dual-Level Android Malware Detection. *Symmetry* **2020**, *12*, 1128. [[CrossRef](#)]
29. Imtiaz, S.I.; ur Rehman, S.; Javed, A.R.; Jalil, Z.; Liu, X.; Alnumay, W.S. DeepAMD: Detection and identification of Android malware using high-efficient Deep Artificial Neural Network. *Future Gener. Comput. Syst.* **2021**, *115*, 844–856. [[CrossRef](#)]
30. Amin, M.; Tanveer, T.A.; Tehseen, M.; Khan, M.; Khan, F.A.; Anwar, S. Static malware detection and attribution in android byte-code through an end-to-end deep system. *Future Gener. Comput. Syst.* **2020**, *102*, 112–126. [[CrossRef](#)]

31. Karbab, E.B.; Debbabi, M. PetaDroid: Adaptive Android Malware Detection Using Deep Learning. In Proceedings of the International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment, Online, 14–16 July 2021; pp. 319–340.
32. Yadav, P.; Menon, N.; Ravi, V.; Vishvanathan, S.; Pham, T.D. EfficientNet convolutional neural networks-based Android malware detection. *Comput. Secur.* **2022**, *115*, 102622. [CrossRef]
33. Yuan, C.; Cai, J.; Tian, D.; Ma, R.; Jia, X.; Liu, W. Towards time evolved malware identification using two-head neural network. *J. Inf. Secur. Appl.* **2022**, *65*, 103098. [CrossRef]
34. Weng Lo, W.; Layeghy, S.; Sarhan, M.; Gallagher, M.; Portmann, M. Graph Neural Network-based Android Malware Classification with Jumping Knowledge. In Proceedings of the 2022 IEEE Conference on Dependable and Secure Computing (DSC), Edinburgh, UK, 22–24 June 2022. [CrossRef]
35. Roy, K.C.; Chen, Q. DeepPran: Attention-based bilstm and crf for ransomware early detection and classification. *Inf. Syst. Front.* **2021**, *23*, 299–315. [CrossRef]
36. Korine, R.; Hendler, D. DAEMON: Dataset/Platform-Agnostic Explainable Malware Classification Using Multi-Stage Feature Mining. *IEEE Access* **2021**, *9*, 78382–78399. [CrossRef]
37. Lu, T.; Du, Y.; Ouyang, L.; Chen, Q.; Wang, X. Android malware detection based on a hybrid deep learning model. *Secur. Commun. Networks* **2020**, *2020*, 8863617. [CrossRef]
38. Yoo, S.; Kim, S.; Kim, S.; Kang, B.B. AI-HydRa: Advanced hybrid approach using random forest and deep learning for malware classification. *Inf. Sci.* **2021**, *546*, 420–435. [CrossRef]
39. Yousefi-Azar, M.; Varadharajan, V.; Hamey, L.; Tupakula, U. Autoencoder-based feature learning for cyber security applications. In Proceedings of the 2017 International Joint Conference on Neural Networks (IJCNN), Anchorage, AK, USA, 14–19 May 2017; pp. 3854–3861. [CrossRef]
40. Viennot, N.; Garcia, E.; Nieh, J. A measurement study of google play. In Proceedings of the 2014 ACM International Conference on Measurement and Modeling of Computer Systems, Austin, TX, USA, 16–20 June 2014; pp. 221–233.
41. Peng, P.; Yang, L.; Song, L.; Wang, G. Opening the blackbox of virustotal: Analyzing online phishing scan engines. In Proceedings of the Internet Measurement Conference, Amsterdam, The Netherlands, 21–23 October 2019; pp. 478–485.
42. Shibata, Y.; Kida, T.; Fukamachi, S.; Takeda, M.; Shinohara, A.; Shinohara, T.; Arikawa, S. Byte Pair Encoding: A Text Compression Scheme That Accelerates Pattern Matching. Researchgate. 1999. Available online: https://www.researchgate.net/publication/2310624_Byte_Pair_Encoding_A_Text_Compression_Scheme_That_Accelerates_Pattern_Matching (accessed on 12 March 2023).
43. Song, X.; Salcianu, A.; Song, Y.; Dopson, D.; Zhou, D. Fast WordPiece Tokenization. In Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing, Punta Cana, Dominican Republic, 7–11 November 2021; pp. 2089–2103.
44. Kudo, T.; Richardson, J. Sentencepiece: A simple and language independent subword tokenizer and detokenizer for neural text processing. *arXiv* **2018**, arXiv:1808.06226.
45. Chang, P.C.; Galley, M.; Manning, C.D. Optimizing Chinese word segmentation for machine translation performance. In Proceedings of the Third Workshop on Statistical Machine Translation, Columbus, OH, USA, 19 June 2008; pp. 224–232.
46. Sennrich, R.; Haddow, B.; Birch, A. Neural machine translation of rare words with subword units. *arXiv* **2015**, arXiv:1508.07909.
47. Li, Y.; Jang, J.; Hu, X.; Ou, X. Android malware clustering through malicious payload mining. In Proceedings of the International symposium on Research in Attacks, Intrusions, and Defenses, Atlanta, GA, USA, 18–20 September 2017; pp. 192–214.
48. Arp, D.; Spreitzenbarth, M.; Hubner, M.; Gascon, H.; Rieck, K.; Siemens, C. Drebin: Effective and explainable detection of android malware in your pocket. In Proceedings of the Network and Distributed System Security Symposium (NDSS) '14, San Diego, CA, USA, 23–26 February 2014.
49. Roberts, J.M. Automatic Analysis of Malware Behaviour using Machine Learning. *J. Comput. Secur.* **2011**, *19*, 639–668.
50. Miranda, T.C.; Gimenez, P.F.; Lalande, J.F.; Tong, V.V.T.; Wilke, P. Debiasing Android Malware Datasets: How Can I Trust Your Results If Your Dataset Is Biased? *IEEE Trans. Inf. Forensics Secur.* **2022**, *17*, 2182–2197. [CrossRef]
51. Li, L.; Gao, J.; Hurier, M.; Kong, P.; Bissyandé, T.F.; Bartel, A.; Klein, J.; Traon, Y.L. Androzo++: Collecting millions of android apps and their metadata for the research community. *arXiv* **2017**, arXiv:1709.05281.
52. Arvind, M. Android Permissions and API Calls during Dynamic Analysis. Available online: <https://data.mendeley.com/datasets/vng8wg9n65/1> (accessed on 12 March 2023).
53. Colaco, C.W.; Bagwe, M.D.; Bose, S.A.; Jain, K. DefenseDroid: A Modern Approach to Android Malware Detection. *Strad Res.* **2021**, *8*, 271–282. [CrossRef]
54. Desnos, A.; Gueguen, G. Androguard-Reverse Engineering, Malware and Goodware Analysis of Android Applications. Available online: <https://androguard.readthedocs.io/en/latest/> (accessed on 12 March 2023).
55. Yerima, S. Android Malware Dataset for Machine Learning. Figshare. 2018. Available online: https://figshare.com/articles/dataset/Android_malware_dataset_for_machine_learning_2/5854653 (accessed on 12 March 2023).
56. Arvind, M. A Android Malware and Normal Permissions Dataset. 2018. Available online: <https://data.mendeley.com/datasets/958wvr38gy/5> (accessed on 27 July 2022).
57. Arvind, M. Android Permission Dataset. 2018. Available online: <https://data.mendeley.com/datasets/8y543xvnsy/1> (accessed on 27 July 2022).
58. Concepcion Miranda, T.; Gimenez, P.F.; Lalande, J.F.; Viet Triem Tong, V.; Wilke, P. Dada: Debaised Android Datasets. 2021. Available online: <https://ieee-dataport.org/open-access/dada-debaised-android-datasets> (accessed on 27 July 2022).

59. Hozan, E. Android APK Reverse Engineering: Using JADX, October 4, 2019. Available online: <https://www.secplicity.org/2019/10/04/android-apk-reverse-engineering-using-jadx/> (accessed on 30 March 2021).
60. Winsniewski, R. Apktool: A Tool for Reverse Engineering Android apk Files. Available online: <https://ibotpeaches.github.io/Apktool/> (accessed on 27 July 2022).
61. Harrand, N.; Soto-Valero, C.; Monperrus, M.; Baudry, B. Java decompiler diversity and its application to meta-decompilation. *J. Syst. Softw.* **2020**, *168*, 110645. [[CrossRef](#)]
62. Bojanowski, P.; Grave, E.; Joulin, A.; Mikolov, T. Enriching Word Vectors with Subword Information. *arXiv* **2016**, arXiv:1607.04606.
63. Zhang, B.; Xiao, W.; Xiao, X.; Sangaiah, A.K.; Zhang, W.; Zhang, J. Ransomware classification using patch-based CNN and self-attention network on embedded N-grams of opcodes. *Future Gener. Comput. Syst.* **2020**, *110*, 708–720. [[CrossRef](#)]
64. Rahali, A.; Lashkari, A.H.; Kaur, G.; Taheri, L.; GAGNON, F.; Massicotte, F. DIDroid: Android Malware Classification and Characterization Using Deep Image Learning. In Proceedings of the 2020 the 10th International Conference on Communication and Network Security, Tokyo, Japan, 27–29 November 2020; pp. 70–82. [[CrossRef](#)]
65. Liu, Y.; Ott, M.; Goyal, N.; Du, J.; Joshi, M.; Chen, D.; Levy, O.; Lewis, M.; Zettlemoyer, L.; Stoyanov, V. Roberta: A robustly optimized bert pretraining approach. *arXiv* **2019**, arXiv:1907.11692.
66. Chicco, D.; Jurman, G. The advantages of the Matthews correlation coefficient (MCC) over F1 score and accuracy in binary classification evaluation. *BMC Genom.* **2020**, *21*, 6. [[CrossRef](#)]
67. Narkhede, S. Understanding auc-roc curve. *Towards Data Sci.* **2018**, *26*, 220–227.
68. Jia, Y.; Qi, Y.; Shang, H.; Jiang, R.; Li, A. A practical approach to constructing a knowledge graph for cybersecurity. *Engineering* **2018**, *4*, 53–60. [[CrossRef](#)]

Disclaimer/Publisher’s Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.