# Graph-Based Android Malware Detection and Categorization through BERT Transformer

Marco Simoni
marco.simoni@iit.cnr.it
Università degli studi di Roma La Sapienza and Consiglio Nazionale delle Ricerche
Pisa, Italy

Andrea Saracino
andrea.saracino@iit.cnr.it
Istituto di Informatica e Telematica, Consiglio Nazionale delle Ricerche
Pisa, Italy

## ABSTRACT

In this paper, we propose a novel approach to Android malware analysis and categorization that leverages the power of BERT (Bidirectional Encoder Representations from Transformers) to classify API call sequences generated from Android API Call Graph. By utilizing the API Call Graph, our approach captures the intricate relationships and dependencies between API calls, enabling a deeper understanding of the behavior exhibited by Android malware. Our results show that our approach achieves high accuracy in classifying API call sequences as malicious or benign and the method provides a promising solution also for categorizing Android malware and can help mitigate the risks posed by malicious Android applications.

## KEYWORDS

Cybersecurity, Malware, Android, BERT Transformer, API Call Graph

**ACM Reference Format:**
Marco Simoni and Andrea Saracino. 2023. Graph-Based Android Malware Detection and Categorization through BERT Transformer. In *The 18th International Conference on Availability, Reliability and Security (ARES 2023), August 29–September 01, 2023, Benevento, Italy.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3600160.3605057

## 1 INTRODUCTION

With the rise of Android-based smartphones and tablets, the number of malicious applications targeting these devices has also increased significantly [12]. As a result, there is a pressing need for effective methods to detect and categorize Android malware, to understand their behavior and plan counteractions. Despite the existence of many Android malware families, categorizing Android malware into specific classes has proven to be a daunting task. In fact, the complexity of Android malware and the varying techniques employed by attackers to circumvent security measures and compromise user devices make it challenging to accurately label and classify malware because they continually adapt their methods to exploit vulnerabilities in the Android ecosystem, leveraging novel attack vectors. Such techniques involve obfuscation, encryption, code injection, dynamic loading, and polymorphism.

In this paper, we propose a novel approach to Android malware analysis that leverages the BERT (Bidirectional Encoder Representations from Transformers) classifier [2] to detect and categorize malware. BERT is a state-of-the-art language model that has shown good results in a wide range of natural language processing tasks and it could be used in order to detect and categorize malware in Android since it has the ability to understand the context of a sequence of words and the ability to handle long sequences and capture the relationships between them. We have employed BERT to classify sequences of Android API calls, extracted from API Call Graphs, in order to identify patterns of behavior that are indicative of different malicious activities, and so different malware categories. The API Call Graph represents the relationships between API calls within an Android application. It describes the set of function calls that occur between the various components of the application. The use of the API Call Graph is particularly useful for malware analysis, as it allows for the identification of patterns of behavior that are indicative of different malicious activities. By analyzing the sequence of API calls and their relationships, it is possible to uncover the intent of the malware and categorize it into different types of threats. We have decided to carry out our analysis on 3 different malware categories which covered more than 90% of the benchmarking dataset, which are *SMS-Trojan*, *Rootkit* and *Spyware*. By applying BERT to the field of Android malware analysis, we have developed a method for detecting and categorizing malware, which achieves 98% of accuracy on detection and 94% of accuracy on categorization. Compared to signature-based and machine learning-based approaches, our approach using BERT has the advantage of not relying on predefined rules or features. Signature-based approaches rely on manually created signatures for known malware variants, which may not be effective in detecting new and unknown malware. On the other hand, machine learning-based approaches typically require extensive feature engineering to extract meaningful information from the data, which can be time-consuming and may not be effective in capturing the complexity of Android malware. Compared to CNN (Convolutional Neural Networks) and GCN-based (Graph Convolutional Networks) approaches, our approach using BERT has the advantage of being better suited for capturing the sequential nature of API calls in Android malware. CNNs and GCNs are primarily designed for processing data with a grid-like structure such as images and graphs, respectively. While these approaches have been applied to sequential data, such as time-series data and natural language processing tasks, they may not be as effective as BERT for capturing long-term dependencies and relationships


*ARES 2023, August 29–September 01, 2023, Benevento, Italy*

ACM ISBN 979-8-4007-0772-8/23/08…$15.00
https://doi.org/10.1145/3600160.3605057

between API calls in malware sequences. Beyond collecting benign APKs for detection, we have leveraged the association between families and categories made by MADAM [12] to build the categorization model. The rest of the paper is structured as follows: Section 2 presents background of the employment of Transformer for NLP problems. Section 3 shows the methodology employed to address the challenge. Section 4 shows the obtained results and Section 5 presents related works on automated malware analysis and categorization. The Section 6 shows the conclusions and future works.

## 2 BACKGROUND

### 2.1 Text-Classification Techniques

Text classification is a crucial task in natural language processing that involves categorizing text into predefined classes based on its content. Traditional machine learning algorithms for text classification, such as Naive Bayes, Support Vector Machines (SVMs), and Decision Trees[5], require hand-engineered features like bag of words, TF-IDF, or n-grams to represent the text. In contrast, deep learning models like Convolutional Neural Networks (CNNs) [6] and Recurrent Neural Networks (RNNs) can learn features automatically from the text, resulting in better performance on complex text data. However, Transformers [17] have emerged as the most advanced method for text classification in recent years. Transformers are a type of neural network architecture that is well-suited for handling sequential data, making them ideal for NLP problems like text classification. Transformers can capture the relationships between words in a sentence more effectively than RNNs or CNNs, resulting in state-of-the-art performance on various text classification tasks. BERT, a transformer-based model, has become the most widely used model for text classification in NLP. BERT [15], has achieved state-of-the-art results in various text classification tasks, demonstrating the superiority of transformers over traditional machine learning algorithms and other deep learning models.

### 2.2 BERT Model for Graph Classification

Transformer is an attention mechanism that learns the contextual relationships between words (or subwords) in a text and is used by BERT [2]. Transformer's basic design consists of two independent mechanisms: an encoder that reads the text input and a decoder that generates a job prediction. Only the encoder mechanism is required because BERT's aim is to produce a language model. The Transformer encoder reads the entire sequence of words at once, in contrast to directional models, which read the text input sequentially (from right to left or left to right). This trait enables the model to understand a word's context depending on all of its surroundings (left and right of the word). Word sequences are changed with a [MASK] token before being fed into the BERT. Based on the context offered by the other, non-masked, words in the sequence, the model then makes an attempt to forecast the original value of the masked words. BERT can be used for the classification of topological representations of a graph. Graph classification involves predicting the properties or labels associated with a given graph. BERT can be adapted for this task by representing the graph as a sequence of tokens and utilizing the transformer-based architecture and by leveraging the power of BERT's contextual understanding and the ability to capture long-range dependencies in a graph, this approach can provide effective solutions for graph classification tasks.

### 2.3 Sequence Categorization and Kahn Algorithm

One of the key challenges in this task is transforming the API call graph of an Android APK into a sequence that is consistent with the topology of the graph. AndroGuard [1] is a well-known tool for generating API call graphs of Android APKs. To transform an Android APK's API call graph into a sequence that respects its topology, we use the Kahn Algorithm. It generates a linear ordering of API calls by sorting the graph's vertices. The algorithm starts with a directed graph $G = (V, E)$ and initializes two sets: $L$ (sorted nodes) and $S$ (nodes with no incoming edges). It iteratively removes outgoing edges from popped nodes in $S$ and adds nodes with no incoming edges to $S$. When $S$ is empty and $E$ is also empty, it returns $L$ as the ordered sequence of API calls. Otherwise, if $E$ is not empty, a cycle is detected. To handle long sequences and capture API call relationships, we employ BERT, a transformer-based model. BERT processes the input sequence bidirectionally and is effective in categorizing API call graph sequences. We fine-tune BERT by training a classifier on pre-trained representations, enabling it to learn specific API call relationships in the graph.

## 3 METHODOLOGY

The methodology described aims to use deep learning to classify APKs as either benign or malicious and to categorize malwares into specific classes such as Spyware [10], SMS Trojan and Rootkit [18]. The following is a summary of the training steps need to be performed for each APK:

- *Collection of APKs*: We have collected both benign and malicious APKs for the study.
- *Obtaining API call graphs*: The Androguard tool was used to obtain API call graphs for each APK. This provides a graphical representation of the interactions between API of an Android app.
- *Transformation of graphs into sequences*: The API call graphs were then transformed into sequences using the Kahn algorithm.
- *Inputs to a BERT model*: The sequences generated in the previous step were used as inputs to a BERT model. In this case, the BERT model was fine-tuned to classify the APKs as benign or malicious and, in the case in which the model recognizes an APK as a malware, it tries to assign a category to the malware.

The API call graph provides a more comprehensive representation of the behavior of an Android app compared to other methods, such as signature-based or static analysis techniques. The graph captures the interactions between different APIs in the app and can reveal malicious behavior that may not be evident from individual API calls. One advantage of using the API Call Graph is that it provides a high-level view of the behavior of an app. Instead of looking at individual API calls, which can be overwhelming, the graph allows us to see the interactions between the APIs in a more structured

[1] https://androguard.readthedocs.io/en/latest/

and intuitive way. This can make it easier to identify patterns and anomalies that may indicate malicious behavior.

## 3.1 Toolchain Overview

The Fig. 1 The toolchain depicted in Figure 1 is designed to identify patterns of behavior in Android applications that can indicate different categories of malware. It consists of two classifiers: the Binary Classifier and the Malware Classifier. Here is a more detailed description of each classifier:

(1) **Binary Classifier**: The purpose of the Binary Classifier is to determine whether a given sequence of Android API calls, extracted from an APK (Android application package), belongs to a malicious or benign application. It uses machine learning techniques to analyze the API call sequence and make a classification decision. If the Binary Classifier identifies the APK as malware, it forwards the corresponding sequence to the Malware Classifier for further analysis. Conversely, if the Binary Classifier determines that the APK is benign, it labels it as a Benign Application.
(2) **Malware Classifier**: Once an APK is identified as malware by the Binary Classifier, it is passed on to the Malware Classifier for more specific analysis. The Malware Classifier examines the API sequence forwarded by the Binary Classifier and focuses on identifying the particular type of malware present in the application. The possible categories of malware that the Malware Classifier can identify in this context are Spyware, Rootkit, or SMS-Trojan. By applying appropriate machine learning techniques, the Malware Classifier determines the type of malware exhibited by the APK.

The preprocessing operation is an essential step in the toolchain, responsible for preparing the data for classification. It consists of two macro steps:

- **API sequence generation**: In this step, API sequences are generated from the API Call Graph. The API Call Graph represents the dependencies and relationships between different Android APIs. By considering these dependencies, the toolchain generates sequences of API calls from the graph, capturing the order in which the APIs are invoked during the execution of the application.
- **Sequence Evaluation**: Once the API sequences are generated, they are passed through the Sequence Evaluation step. In this step, a pre-trained deep learning model called BERT (Bidirectional Encoder Representations from Transformers) is applied to the API sequences for classification, enabling the classification of the APKs into the desired categories (malicious or benign) and the identification of the specific type of malware.

Overall, this toolchain utilizes machine learning and deep learning techniques to extract features from API call sequences, classify APKs as malicious or benign, and further identify the specific type of malware if the APK is flagged as malicious.

## 3.2 API Sequence Generation

In an API Call Graph, an API may depend on another API if it invokes or calls the other API during its execution. When an Android application is running, it interacts with the system and other components by making API calls. The API Call Graph represents these interactions as a directed graph, where the nodes are the APIs, and the edges represent the API calls. For example, let's consider an Android app that retrieves weather data for a given location. The app uses the following API call sequence to accomplish this task:

(1) The app first calls the API getLocation() to get the device's current location.
(2) It then makes an API call to checkSelfPermission() to check whether the app has the necessary permission to access the device's location.
(3) If the app has the required permission, it proceeds to make an API call to getWeatherData() with the obtained location as a parameter to retrieve the weather data.
(4) Finally, the app displays the weather data to the user using the displayWeatherData() API.

In this case, we can say that getLocation() depends on checkSelfPermission() since getLocation() needs to call checkSelfPermission() first to verify if it has the required permission. Similarly, getWeatherData() depends on both getLocation() and checkSelfPermission(). The Fig. 2 shows a graph which depicts an API call graph of an Android application. In this graph, each node represents an Android API that is being used by the application. The nodes are labeled A through F and are represented as circles with bold letters inside. The lines connecting the nodes represent the API calls made by the application, and the direction of the lines indicates the flow of the calls. For example, node A makes API calls to nodes B and C, node B makes API calls to nodes D and E, node C makes an API call to node F, and node E makes an API call to node F. The graph can be used to analyze the dependencies between different APIs used by the application, identify possible bottlenecks and optimize the application's performance. The algorithm repeats this process

**Algorithm 1:** Api Sequence Generation

**Input** : graph, a dictionary where the keys are API names and the values are lists of API names that the key API depends on
**Output**: Sequence, a list of API names in the order they should be executed
**Graph Structure Format** ← {`parent node: [child nodes]`};
**Example Graph** ← {`A: [B, C], B: [D, E], C: [F], D: [], E: [F], F: []`};
**Sequence obtained by Kahn Algorithm** ← {`A, C, F, B, E, D`}

until there are no more APIs in the graph. The order in which the APIs are selected will determine the order of execution of the APIs.

In the case of the provided example, the sequence is generated as follows:

(1) API A is inserted first into the queue since it has no dependencies;
(2) API C is inserted after A because it depends only on A and has no dependencies on other APIs;

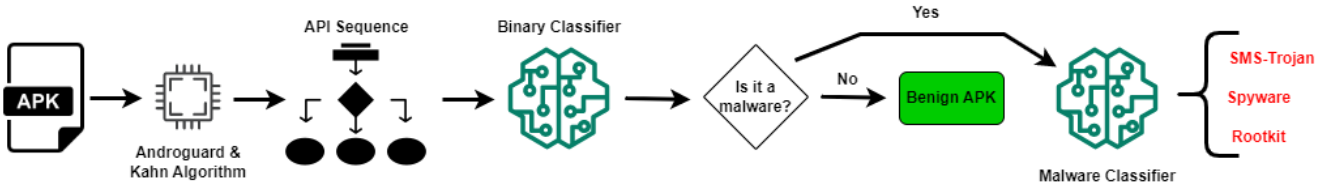


**Figure 1: Complete Toolchain**

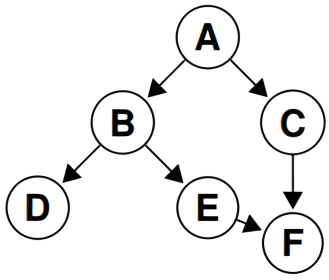


**Figure 2: API Call Graph Example**

(3) API F is inserted after C because it depends only on C and has no dependencies on APIs B and E;
(4) API B is inserted after F because it depends only on A and has no dependencies on APIs C and F;
(5) API E is inserted after B because it depends only on B and has no dependencies on APIs C and F;
(6) API D is inserted after E because it has no dependencies on any other APIs;

## 3.3 Sequence Evaluation

The toolchain for classifying sequences of Android APIs into benign or malware, and further classifying malware into SMS Trojan, Rootkit, or Spyware can be described as follows:

- *Sequence Collection and Preprocessing*: Collect a large dataset of Android API sequences, including both benign and malware sequences removing any unnecessary features, tokenizing the data, and converting it into a suitable format for BERT input. The dataset come from both Drebin [1][14] and Maldroid [8] [7].
- *Training the BERT Classifier for Malware Detection*: Next, a BERT binary classifier is trained on the preprocessed data to classify sequences as either benign or malware. The classifier learns to identify patterns in the sequences that indicate malicious behavior.
- *Classification of Malware*: If a sequence is classified as malware, it is then passed to a second BERT classifier which is trained to classify the malware into one of the three categories: SMS Trojan, Rootkit, Spyware.
- *Validation and Testing*: Validate the accuracy of the BERT classifiers on a separate test set of data to ensure that the classifiers are not overfitting the training data and are generalizable to new, unseen data.
- *Deployment*: Deploy the trained classifiers in a environment for real-time classification of Android API sequences.

## 3.4 Classification Model

The proposed deep learning model is designed using TensorFlow [4] and the Keras API. It is made up of two main components: the BERT encoder and a neural network. **BERT Encoder**: The input to the model is represented by raw text, which is first preprocessed by the `bert_preprocess` layer, which is a pre-trained TensorFlow Hub module [2]. The pre-processed text is then passed through the BERT encoder, which is a pre-trained TensorFlow Hub module [3], and is responsible for generating a high-level representation of the input text. There are little differences between the Neural Network dedicated to malware categorization and the Neural Network dedicated to malware detection.

*3.4.1 Categorization.* **Neural Network**: The output from the BERT encoder passes through a series of neural network layers. The first layer is a dropout layer with a dropout rate of 0.1. This layer helps to prevent overfitting by randomly setting some input neurons to zero during training. The next four layers are dense, fully-connected layers with increasing number of neurons (1024, 2048, 4096, 4096) and ReLU activation functions. Finally, the last layer is a dense layer with 3 neurons and a softmax activation function, which generates the final output of the model.

*3.4.2 Detection.* **Neural Network**: A number of neural network layers process the output from the BERT encoder. The top layer has a dropout rate of 0.1 and is a dropout layer. The following three layers are highly linked and dense, with increasing numbers of neurons (1024, 2048, and 4096), as well as ReLU activation capabilities. The model's final output is produced by a dense layer with one neuron and a sigmoid activation function in the last layer.

## 3.5 Data Augmentation

In the context of an Android API call sequence, data augmentation for a BERT model involves generating new sequences by manipulating existing ones. The goal is to balance the dataset, particularly for the Spyware or SMS Trojan classes. The process entails cutting two or more APIs from the original sequences and combining them to form new sequences that belong to the desired classes. The purpose of this data augmentation strategy is twofold. Firstly, it aims to increase the size of the dataset. By creating additional samples, the model has access to more diverse examples during training, which can enhance its ability to generalize and make accurate predictions. Secondly, the augmentation strategy assists in addressing the issue of class imbalance. In many machine learning tasks, including malware detection, certain classes may have significantly

[2] https://tfhub.dev/tensorflow/bert_en_uncased_preprocess/3
[3] https://tfhub.dev/tensorflow/lambert_en_uncased_L-24_H-1024_A-16/2

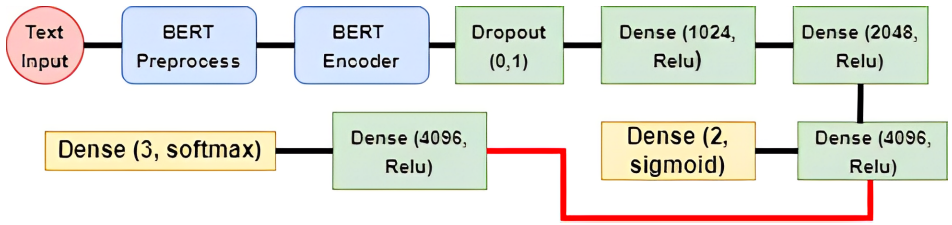


**Figure 3: Structure of *Detection* NN and *Categorization* NN.**

fewer instances compared to others. This can lead to biased model performance and by generating new sequences specifically for the Spyware or SMS Trojan classes, the dataset becomes more balanced, allowing the BERT model to learn effectively from each class and improve its capability to detect these types of malware.

## 4 EXPERIMENTS

The Dataset for SMS-Trojan, Rootkit and Spyware APKs has been retrieved from Drebin [1][14], collecting more than 2000 applications. In order to associate each application to a precise category, we have leveraged the association made by the categorization proposed in [12]; in the first step, we have linked each malware family to the correspondent malware category and then we have linked each malware hash value to the corresponding category passing through the family associated to the chosen hash value. As you can see from the Table 4, most number of malwares belong to three different classes: **SMS Trojan**, **Rootkit**, **Spyware**. They cover more than 90% of the whole dataset.

Regarding Benign APKs, we have retrieved the complete package from Maldroid [8] [7] and we have also crawled Google Play store in order to reach a balanced dataset combining the same number of malicious and benign applications.

It is worth describing the behavior of the analyzed malwares proposed in [12].

*Rootkit*: an Android rootkit is a form of malware that can gain administrative access to an Android device, allowing it to execute tasks that ordinary apps cannot.

*SMS Trojan*: An Android SMS Trojan is a type of malware that infects an Android device and uses it to send text messages (SMS) to premium-rate numbers, incurring charges for the victim without the victim's knowledge or consent.

*Spyware*: Android Spyware is a type of malicious software that is designed to gather information from an infected Android device without the user's knowledge or consent.

### 4.1 Training Phase Testbed

A split function is used to split the input data into training and testing sets; the function is applied two times in order to extract from the first training set the partition used as validation set. The stratify parameter ensures that the proportion of each class is balanced between the training and testing sets. A *Metrics* list contains three evaluation metrics for the model: categorical and binary accuracy, respectively for categorization and detection, precision, and recall. Moreover, three callback functions are defined to be used during model training:

- *EarlyStopping* is a callback function that will stop the training process if the validation loss does not improve for 5 consecutive epochs.
- *ModelCheckpoint* is a callback function that saves the weights of the model at every epoch.
- *Callback* is a parent class for user-defined callback functions.

The optimizer parameter specifies the algorithm to use for optimizing the model parameters, and *Adam* is a popular choice for deep learning. The loss parameter specifies the loss function to use during training, and *CategoricalCrossentropy* is appropriate for multi-class classification problems.

### 4.2 Results

Table 5 is a categorization report that provides precision, recall, and F1-score for each class in the dataset, as well as macro and weighted averages and the number of instances for each class. The last row of Table 5 shows the results of categorizing the test dataset, with metrics such as loss, accuracy, precision, and recall. The loss is 0.3295, which is relatively low, indicating good performance. The accuracy is 0.9423, meaning that the model classified 94.23% of the test instances correctly. The precision is 0.9433, which suggests that the model has a low false positive rate, and the recall is 0.9387, indicating a relatively low false negative rate. The figure 6 shows the Categorization ROC cuve. An AUC value of 1.0 represents a perfect classifier, while a value of 0.5 represents a random classifier. The values provided are all close to 1.0, with Spyware having an AUC of 0.9899, SMS-Trojan having an AUC of 0.9902, and Rootkit having an AUC of 0.9871. These high AUC values suggest that the model is able to distinguish between positive and negative instances with high accuracy for all three classes.

Table 7 shows the detection report in which the precision, recall, and F1-score for both classes are very high, with both classes having a precision and recall of 0.98 and an F1-score of 0.98. In the last row Table 7 the loss is 0.1254, hence negligible. The precision is 0.9805, which means that the model has a low false positive rate, and the recall is 0.9718, indicating that the model has a low false negative rate.

## 5 RELATED WORK

In the last years, several attempts have been made in order to detect malwares by the use of DNNs (Deep Neural Networks): MAPAS [3] uses convolution neural networks to assess the behaviors of malicious programs based on their API call graphs (CNN) while Sachith Seneviratne et al. [13] developed SHERLOCK, a self-supervised deep learning model based on the Vision Transformer (ViT) architecture that can find malware. Compared to convolutional neural

**Figure 4: Category distribution**

| Category | Counts |
|---|---|
| *SMS Trojan + Installer* | 0.141% |
| ***SMS Trojan*** | **0.2879%** |
| ***Rootkit*** | **0.2542%** |
| *Installer* | 0.0342% |
| ***Spyware*** | **0.3737%** |
| *Trojan* | 0.0038% |
| *SMS Trojan + Botnet* | 0.0016% |
| *SMS Trojan + Spyware* | 0,0261% |
| *Botnet* | 0.0033% |
| *Ransomware* | 0.0011% |
| *Total No. of APKs* | 903 |

**Figure 5: Categorization Report**

| Class | Precision | Recall | F1-Score | Support |
|---|---|---|---|---|
| *Spyware* | 0.913 | **0.971** | 0.941 | 272 |
| *Rootkit* | **0.946** | 0.911 | 0.929 | 271 |
| *SMS-trojan* | **0.970** | 0.945 | 0.957 | 272 |
| **Accuracy** | - | - | 0.942 | 815 |
| **Macro avg** | 0.943 | 0.942 | 0.942 | 815 |
| **Weighted avg** | 0.943 | 0.942 | 0.942 | 815 |
| | **Loss** | **Accuracy** | **Precision** | **Recall** |
| **Overall Values** | *0.3295* | 0.9423 | **0.9433** | 0.9387 |

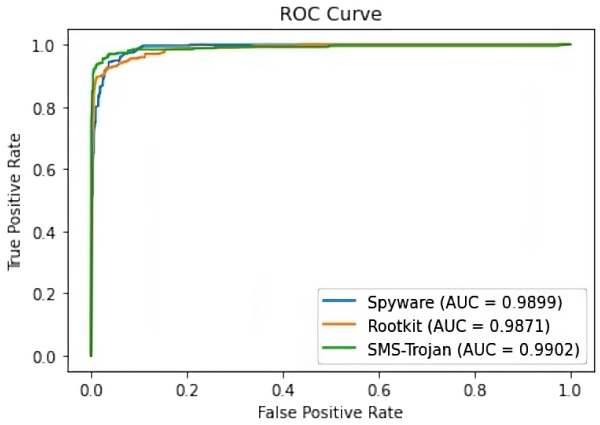


**Figure 6: AUC metric for Malware Categorization**

**Figure 7: Detection Report**

| Class | Precision | Recall | F1-Score | Support |
|---|---|---|---|---|
| *Benign APK* | 0.97 | **0.98** | **0.98** | 568 |
| *Malicious APK* | **0.98** | 0.97 | **0.98** | 568 |
| **Accuracy** | - | - | 0.98 | 1136 |
| **Macro avg** | 0.98 | 0.98 | 0.98 | 1136 |
| **Weighted avg** | 0.98 | 0.98 | 0.98 | 1136 |
| | **Loss** | **Accuracy** | **Precision** | **Recall** |
| **Overall Values** | *0.1254* | 0.9762 | **0.9805** | 0.9718 |

networks (CNNs), our approach has the advantage of being able to capture the relationships between API calls in the sequence, which is crucial for detecting and categorizing complex Android malware. Minghui Cai et al.[15] created a vector representation of E-FCGs using an approach based on Graph Convolutional Networks (GCN). GCNs are a powerful tool for analyzing the structure and behavior of complex graphs, including API call graphs that represent the behavior of Android apps. However, developing effective GCN models can require a large amount of domain knowledge and feature engineering and GCNs are often limited by the quality and completeness of the graph representation of the app's behavior. BERT, on the other hand, can learn from any sequence of API calls, not just those included in the graph. This makes it more robust to variations in the representation of the app's behavior and less likely to miss important signals of malicious behavior. MALBERT [11] is an automated detection of dangerous software using a Transformers-style architecture while Ullah, Farhan, et al. [16] developed a method that makes use of both textual and visual aspects to discover and assess network malware. Malceiver [9] is a multi-modal hierarchical Perceiver model for detecting malware on Android. In comparison to other approaches, our methodology for detecting and categorizing malware in Android APKs has several advantages. Firstly, our approach uses BERT, which is capable of capturing the relationships between API calls in a sequence, making it more effective in detecting and categorizing complex malware than other approaches such as CNNs. Additionally, BERT can learn from any sequence of API calls, not just those included in the graph, making it more robust to variations in the representation of the app's behavior and less likely to miss important signals of malicious behavior. Furthermore, our approach is based on preprocessed API sequences, which makes it less dependent on domain knowledge and feature engineering than approaches such as GCNs. Finally, our approach does not require any visual aspects of the APK or network traffic, making it more scalable and efficient.

## 6 CONCLUSIONS AND FUTURE WORK

In this work we have presented an approach for Android malware analysis using BERT to classify API call sequences. The proposed approach has shown promising results for both malware detection and categorization. For malware categorization, our approach achieved a relatively low loss, high accuracy, and precision, indicating a low false positive rate and a relatively low false negative rate. This suggests that our approach can effectively categorize Android malware

and help mitigate the risks posed by malicious applications. For malware detection, our approach achieved an extremely low loss, high accuracy, and precision, indicating good performance with a low false positive rate and a low false negative rate. These results suggest that our proposed approach can effectively detect Android malware and help prevent its spread. In future works, we plan to expand the categories of Android malware that our approach can detect and classify. This could involve exploring new techniques for feature engineering or incorporating additional sources of information, such as metadata or network traffic. Additionally, we plan to extend our approach to other types of systems, such as ELF systems commonly used in Linux.

## ACKNOWLEDGMENTS

This work has been partially supported by the H2020 project SIFIS-Home (grant agreement 952652).

## REFERENCES

[1] Daniel Arp, Michael Spreitzenbarth, Malte Hubner, Hugo Gascon, and Konrad Rieck. 2014. DREBIN: Effective and Explainable Detection of Android Malware in Your Pocket. In *NDSS 2014, San Diego, California, USA, February 23-26, 2014*. The Internet Society. https://www.ndss-symposium.org/ndss2014/drebin-effective-and-explainable-detection-android-malware-your-pocket

[2] Jacob Devlin et al. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers)*, Jill Burstein, Christy Doran, and Thamar Solorio (Eds.). Association for Computational Linguistics, 4171–4186. https://doi.org/10.18653/v1/n19-1423

[3] Jinsung Kim et al. 2022. MAPAS: a practical deep learning-based android malware detection system. *Int. J. Inf. Sec.* 21, 4 (2022), 725–738. https://doi.org/10.1007/s10207-022-00579-6

[4] Martín Abadi et al. 2016. TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems. *CoRR* abs/1603.04467 (2016). arXiv:1603.04467 http://arxiv.org/abs/1603.04467

[5] Monisha Kanakaraj and Ram Mohana Reddy Guddeti. 2015. Performance analysis of Ensemble methods on Twitter sentiment analysis using NLP techniques. In *IEEE ICSC 2015, Anaheim, CA, USA, February 7-9, 2015*, Mohan S. Kankanhalli, Tao Li, and Wei Wang (Eds.). IEEE Computer Society, 169–170. https://doi.org/10.1109/ICOSC.2015.7050801

[6] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. 2017. ImageNet classification with deep convolutional neural networks. *Commun. ACM* 60, 6 (2017). https://doi.org/10.1145/3065386

[7] Samaneh Mahdavifar, Dima Alhadidi, and Ali A. Ghorbani. 2022. Effective and Efficient Hybrid Android Malware Classification Using Pseudo-Label Stacked Auto-Encoder. *J. Netw. Syst. Manag.* 30, 1 (2022), 22. https://doi.org/10.1007/s10922-021-09634-4

[8] Samaneh Mahdavifar, Andi Fitriah Abdul Kadir, Rasool Fatemi, Dima Alhadidi, and Ali A. Ghorbani. 2020. Dynamic Android Malware Category Classification using Semi-Supervised Deep Learning. In *IEEE, DASC/PiCom/CBDCom/CyberSciTech 2020, Calgary, AB, Canada, August 17-22, 2020*. IEEE, 515–522. https://doi.org/10.1109/DASC-PICom-CBDCom-CyberSciTech49142.2020.00094

[9] Niall McLaughlin. 2022. Malceiver: Perceiver with Hierarchical and Multi-modal Features for Android Malware Detection. *CoRR* abs/2204.05994 (2022). https://doi.org/10.48550/arXiv.2204.05994 arXiv:2204.05994

[10] Fabio Pierazzi, Ghita Mezzour, Qian Han, Michele Colajanni, and V. S. Subrahmanian. 2020. A Data-driven Characterization of Modern Android Spyware. *ACM Trans. Manag. Inf. Syst.* 11, 1 (2020), 4:1–4:38. https://doi.org/10.1145/3382158

[11] Abir Rahali and Moulay A. Akhloufi. 2021. MalBERT: Malware Detection Using Bidirectional Encoder Representations from Transformers. In *2021 IEEE (SMC)* (Melbourne, Australia). IEEE Press, 3226–3231. https://doi.org/10.1109/SMC52423.2021.9659287

[12] Andrea Saracino, Daniele Sgandurra, Gianluca Dini, and Fabio Martinelli. 2018. MADAM: Effective and Efficient Behavior-based Android Malware Detection and Prevention. *IEEE Trans. Dependable Secur. Comput.* 15, 1 (2018), 83–97. https://doi.org/10.1109/TDSC.2016.2536605

[13] Sachith Seneviratne, Ridwan Shariffdeen, Sanka Rasnayaka, and Nuran Kasthuriarachchi. 2022. Self-Supervised Vision Transformers for Malware Detection. *IEEE Access* 10 (2022), 103121–103135. https://doi.org/10.1109/ACCESS.2022.3206445

[14] Michael Spreitzenbarth, Felix C. Freiling, Florian Echtler, Thomas Schreck, and Johannes Hoffmann. 2013. Mobile-sandbox: having a deeper look into android applications. In *SAC '13, Coimbra, Portugal, March 18-22, 2013*, Sung Y. Shin and José Carlos Maldonado (Eds.). ACM, 1808–1815. https://doi.org/10.1145/2480362.2480701

[15] Chi Sun, Xipeng Qiu, Yige Xu, and Xuanjing Huang. 2019. How to Fine-Tune BERT for Text Classification?. In *Chinese Computational Linguistics - 18th China National Conference, CCL 2019, Kunming, China, October 18-20, 2019, Proceedings (Lecture Notes in Computer Science, Vol. 11856)*, Maosong Sun, Xuanjing Huang, Heng Ji, Zhiyuan Liu, and Yang Liu (Eds.). Springer, 194–206. https://doi.org/10.1007/978-3-030-32381-3_16

[16] Farhan Ullah, Amjad Alsirhani, Mohammed Mujib Alshahrani, Abdullah Alomari, Hamad Naeem, and Syed Aziz Shah. 2022. Explainable Malware Detection System Using Transformers-Based Transfer Learning and Multi-Model Visual Representation. *Sensors* 22, 18 (2022), 6766. https://doi.org/10.3390/s22186766

[17] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is All you Need. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, Isabelle Guyon, Ulrike von Luxburg, Samy Bengio, Hanna M. Wallach, Rob Fergus, S. V. N. Vishwanathan, and Roman Garnett (Eds.). 5998–6008. https://proceedings.neurips.cc/paper/2017/hash/3f5ee243547dee91fbd053c1c4a845aa-Abstract.html

[18] Dong-Hoon You and Bong-Nam Noh. 2011. Android platform based linux kernel rootkit. In *6th International Conference on Malicious and Unwanted Software, MALWARE 2011, Fajardo, Puerto Rico, USA, October 18-19, 2011*. IEEE Computer Society, 79–87. https://doi.org/10.1109/MALWARE.2011.6112330