


Article

# Malware Classification Using Dynamically Extracted API Call Embeddings

Sahil Aggarwal and Fabio Di Troia \* 

Department of Computer Science, San Jose State University, San Jose, CA 95192, USA

\* Correspondence: fabio.ditroia@sjsu.edu

**Abstract:** Malware classification stands as a crucial element in establishing robust computer security protocols, encompassing the segmentation of malware into discrete groupings. Recently, the emergence of machine learning has presented itself as an apt approach for addressing this challenge. Models can undergo training employing diverse malware attributes, such as opcodes and API calls, to distill valuable insights for effective classification. Within the realm of natural language processing, word embeddings assume a pivotal role by representing text in a manner that aligns closely with the proximity of similar words. These embeddings facilitate the quantification of word resemblances. This research embarks on a series of experiments that harness hybrid machine learning methodologies. We derive word vectors from dynamic API call logs associated with malware and integrate them as features in collaboration with diverse classifiers. Our methodology involves the utilization of Hidden Markov Models and Word2Vec to generate embeddings from API call logs. Additionally, we amalgamate renowned models like BERT and ELMo, noted for their capacity to yield contextualized embeddings. The resultant vectors are channeled into our classifiers, namely Support Vector Machines (SVMs), Random Forest (RF), k-Nearest Neighbors (kNNs), and Convolutional Neural Networks (CNNs). Through two distinct sets of experiments, our objective revolves around the classification of both malware families and categories. The outcomes achieved illuminate the efficacy of API call embeddings as a potent instrument in the domain of malware classification, particularly in the realm of identifying malware families. The best combination was RF and word embeddings generated by Word2Vec, ELMo, and BERT, achieving an accuracy between 0.91 and 0.93. This result underscores the potential of our approach in effectively classifying malware.



**Citation:** Aggarwal, S.; Di Troia, F. Malware Classification Using Dynamically Extracted API Call Embeddings. *Appl. Sci.* **2024**, *14*, 5731. <https://doi.org/10.3390/app14135731>

Academic Editor: Gianluca Lax

Received: 20 May 2024

Revised: 24 June 2024

Accepted: 26 June 2024

Published: 30 June 2024



**Copyright:** © 2024 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/>).

**Keywords:** word embeddings; dynamic analysis; API calls; Hmm2Vec; Word2Vec; ELMo; BERT; SVM; RF; kNN; CNN

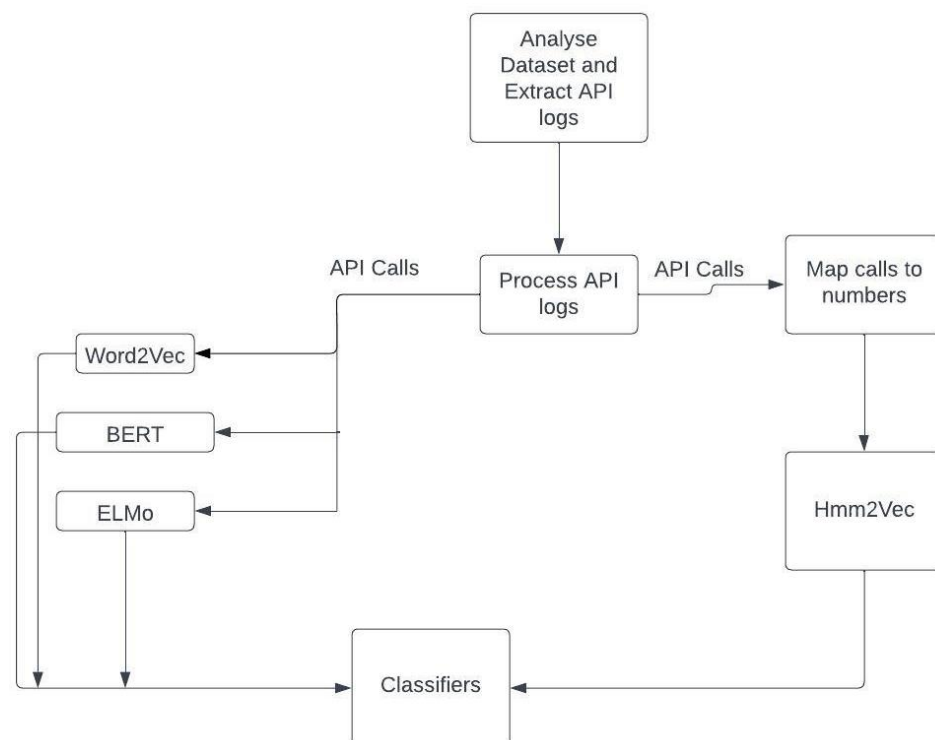
## 1. Introduction

Over the course of recent decades, technology has become an indispensable component of our daily lives, resulting in the pervasive presence of computers. With these remarkable advancements, it comes as no surprise that attackers have devised diverse methodologies to disrupt and impair computer systems for their own gain. These malevolent actors embed harmful code into ostensibly ordinary programs, leading to a range of detrimental outcomes. Their actions include pilfering funds and personal data, overwhelming computers with downloads to induce system failures, and even encrypting data while demanding cryptocurrency ransoms. This malicious code collectively goes by the term “malware”. Over time, malware has progressed to the extent that it can infiltrate virtually any system, including laptops, mobile devices, and internet servers. According to the 2023 SonicWall Cyber Threat Report, there has been a resurgence in malware attacks for the first time in over three years, reaching a substantial tally of 5.5 billion [1]. Given these circumstances, the need for malware detection and classification has intensified.

The classification of malware involves ascertaining the class or family to which a specific malware variant belongs. The underlying rationale is that malware within the same

families or categories exhibits akin behaviors during execution. This constitutes a foundational issue in malware analysis. Existing tools often rely on signature-based detection algorithms, maintaining a database of previously scrutinized malware signatures. However, this approach often falters when confronted with novel and unseen malware strains. Furthermore, malware creators persistently devise novel evasion strategies. To counter this, a substantial portion of recent research has centered around harnessing machine learning for the purposes of malware detection and classification. Malware can be classified into types like Worms and Trojans based on their interactions with systems, and these categories can be further broken down into families. Malware within the same family shares common attributes such as a shared code base or common authors. These shared traits can be exploited for effective malware classification.

In this study, we investigate the utilization of API calls as features for the classification of malware. We extract the API calls dynamically while the malware is running. The rationale here is to avoid techniques that may obfuscate API calls when statically extracted [2–4]. We deploy word embedding techniques to generate vector representations from these API calls, which are subsequently employed as inputs to a variety of multi-class classifiers. The word embedding methods scrutinized in this research are HMM2Vec, Word2Vec, Embeddings from Language Model (ELMo), and Bidirectional Encoder Representations from Transformers (BERT). These techniques play a pivotal role in our classification process, given that the quality of the generated embeddings substantially influences the capacity to encapsulate latent features within call sequences. Across the board, we experiment with a consistent ensemble of multi-class classifiers, specifically k-Nearest Neighbors (kNNs), Support Vector Machines (SVMs), Random Forest (RF) classifiers, and Convolutional Neural Networks (CNNs). These hybrid methodologies are implemented and evaluated across a diverse spectrum of malware samples. The findings derived from these experiments are amalgamated within this report. Figure 1 provides an overview of the experimental trajectory, commencing from malware analysis and log extraction, culminating in classification endeavors. The overall procedure remains largely uniform across methodologies, with the exception of HMM2Vec, which encompasses an additional step for mapping calls to numerical values.



**Figure 1.** General layout of the experiments.

The remainder of this paper is organized as follows. Section 2 discusses related work and the previous literature on malware classification, followed by Section 3, which describes the dataset used and provides an in-depth exploration of the background concepts necessary to understand the work performed. In Section 4, the proposed experiments and their results are presented in detail. Finally, Section 5 concludes the report and suggests potential ideas for further improvement.

## 2. Related Work

Malware classification is among some of the most researched problems because of its global impact, and as such, several machine learning techniques have been identified to try to solve this problem. Within them, a common methodology is to study how the malware interacts with a system and extract features from it which could be helpful in understanding its behavior. These features may include a number of things such as opcodes, register changes, file system changes, API calls made, and so on. These features can be extracted using static analysis or dynamic analysis of the malware executable.

Several studies have used API call logs as features to detect or classify malware, and they have achieved good results. In fact, malware under execution may cause the system to make different API calls, and these are good features for machine learning as they are usually used in a specific way. To extract these calls, we need to run the malware executable, which often requires a virtual environment [5]. The authors in [6] proposed a method for selecting a subset of API calls and achieved 0.98 accuracy in malware classification using algorithms like SVM and Random Forest. In a similar scheme, the authors in [7] proposed enhancing the API call information using Term Frequency and Inverse document frequency (TF-IDF) to select only the most important calls for malware detection, achieving near-perfect accuracy. These results provide enough evidence to argue for the effectiveness of API calls as a feature.

While algorithms like SVM and Random Forest generally perform well at classification tasks, recent works have shown that techniques using deep neural networks can be successfully used because of their ability to uncover and learn complex relationships. In [8], the authors proposed a CNN-based architecture where they converted malware binaries to images, recording an impressive 98% accuracy. In [9], the authors discuss data mining techniques to select features and classify them using models like kNN and Naive Bayes. The selection of appropriate techniques to enhance the relevance of information in the features can be crucial in classifying malware.

Word embeddings are learned representations of text that are used in natural language processing (NLP). Since their advent, they have been employed in a number of varied tasks with very promising results. Among these, malware classification has also emerged as a field where these techniques prove useful. This statement is supported by the work carried out in [10], where API call sequences are converted to numeric vectors using various methods. Another popular algorithm used to generate these representations is Word2Vec, developed at Google [11]. In [12], the author uses Word2Vec to extract vectors from machine code instructions and demonstrates the algorithm's ability in tasks of malware detection.

Hidden Markov Models (HMMs) have found an effective use case in malware analysis because of their ability to determine patterns in sequences that are generally not directly observable. Several research works have tried modeling HMMs on various malware features and have shown promising results. In [13], the authors conduct a comparison between the efficacy of opcode sequences and API call sequences by training HMMs on them and conclude that the latter performs better. In a similar study [14], the authors also use HMMs and reach a similar conclusion. They argue that while opcodes are generally good, they fail to predict some families due to obfuscation techniques that are likely to affect them more than API calls. In [15], the author uses HMM2Vec, a technique for generating vectors using HMMs opcodes as features. The work compares the performance of HMM2Vec with Word2Vec and delivers promising results with an accuracy of about 93% in both cases, thereby validating the HMM2Vec approach.

One drawback of the Word2Vec algorithm is the inability to produce vectors that have contextual information. Further developments in the field of NLP have led to the development of several other models for generating embeddings that could overcome this problem. One such model was introduced in [16] called Embeddings for Language Model (ELMo), which is based on the biLSTM model, and it achieved state-of-the-art results in several common NLP tasks. Another breakthrough came with the introduction of transformer architecture and the self-attention mechanism in [17]. With the increased interest in transformer-based architecture, J. Devlin et al. [18] introduced Bidirectional Encoder Representations from Transformers (BERT) which again achieved state-of-the-art results. In [19], the authors used malware opcode and compared the performance of these newer techniques with HMM2Vec and Word2Vec for the task of malware family classification. Both approaches performed relatively well and showed improved performance over the former two with results indicating that BERT slightly outperformed ELMo in the given task. More on how each of these techniques work is discussed in Section 3. This research is highly motivated by the work performed in [15,19]. The work performed here follows a similar structure to these works, but instead of opcode sequences, we rely on API call sequences for classification.

### 3. Background

This section provides an overview of the fundamental tools and technologies employed in this research. We commence with an exploration of malware analysis and the procedure for extracting API calls from a given executable. Subsequently, we offer a concise explanation of the functioning of the utilized word embedding models: HMM2Vec, Word2Vec, BERT, and ELMo. Finally, we examine the diverse multi-class classifiers employed in this study. Specifically, we discuss Support Vector Machines (SVMs), k-Nearest Neighbors (kNNs), Random Forest, and Convolutional Neural Networks (CNNs).

#### 3.1. Malware Analysis

Any program that is created with the intent to disrupt or harm a computer system can be defined as malware. As such, based on its behavior, malware can take many different forms and present as one among various categories such as Ransomware and Trojan. This broad categorization, dependent on malware's characteristics, behavior, and aim, is referred to as the category of malware. On the other hand, a malware family is a more specific characterization within a category wherein we group them based on functional similarities such as the code base or origin, i.e., a single category has multiple families. Due to this, the classification of malware by category is generally more complex than classification by family. In this paper, we focus on both of these categorizations, and our experiments try to classify malware in both groups.

We focus on performing a dynamic analysis of the malware sample by extracting the API calls at runtime. The term dynamic analysis can be explained as the analysis of software that is executed in a controlled environment. In contrast, static analysis refers to the analysis of software that is not under execution. Although dynamic analysis usually incurs a larger overhead, previous work has shown that they are suitable for deriving an accurate model of the malware [6,14]. Common information that can be derived from dynamic analysis of a malware executable is API calls, system calls, and register changes, and these can be useful as features in several malware tasks. In this research, we specifically focus on the API calls extracted. In the lifetime of its execution, malware may use various API calls for different purposes such as accessing a file system or connecting to a remote server. This behavior can be exploited to understand the malware better.

There are a lot of tools that can help us in extracting the required information. The tool that was used in this research is called Buster Sandbox Analyzer (BSA) version 1.92 [20]. BSA is a free, open-source dynamic analysis tool designed to detect if processes exhibit malicious activities. BSA works by executing the malware in a controlled virtual environment so the host system is not affected by any harmful activities. It does so by working in con-

junction with another software called Sandboxie, which provides a sandboxed environment to run programs in [21]. One of the best features of BSA is that it can capture an executable's workings and automatically generate reports based on it, including details such as API calls, network communications, file system changes, and various other information. Together, BSA and Sandboxie provide a powerful method of analyzing malware and, as such, are used in this research to analyze the malware samples.

### 3.2. Word Embedding Techniques

As mentioned previously, word embedding techniques are primarily used in natural language processing tasks to numerically quantify the distance between words in a vocabulary. In this research, we use these techniques to generate features for our classifiers. It is expected that, by using these techniques, we will be able to identify relationships that can be better understood by the classification algorithms. In this section, we discuss the embedding techniques used in this research starting with HMM2Vec, an embedding technique based on HMMs, followed by the well-known Word2Vec. Finally, we discuss some of the relatively newer techniques ELMo and BERT which can understand the context of the words in our vocabulary.

#### 3.2.1. HMM2Vec

The Hidden Markov Model is a statistical model in which a system is assumed to be a Markov process of order one with a number of hidden states that are not directly observable. The aim of an HMM is to figure out the most probable sequence of states that could have generated the data. Usually, HMM can be represented by three matrices  $A$ ,  $B$ , and  $\Pi$  which denote the hidden state transition probability, the observation probability matrix, and the initial state probabilities, respectively [22]. The number of hidden states that the HMM may infer is denoted by  $N$  and is usually selected by the user. Other notations that are used are  $T$ , to define the length of the observation sequence, and  $M$ , which represents the number of unique symbols in the vocabulary. The values of  $M$  and  $T$  are most commonly derived on the basis of the training dataset.

HMM2Vec is a technique in which we calculate the cosine similarity between any two vectors. We consider the row of the converged  $B$ -transpose matrix as the vector representation of that particular letter. This means that for each individual letter, we have a vector of length 2. The length of the vector depends on the number of hidden states.

More detailed information on HMMs and how they work can be accessed at [22]. More details on HMM2Vec can be found at [15,23].

#### 3.2.2. Word2Vec

Word2Vec is a word embedding technique introduced by Tomas Mikolov at Google in 2013 [11]. The algorithm is based on a shallow neural network that can be used to embed features in a high-dimensional space. The trained extracted embeddings can be used in several tasks as the words that are similar in context are represented closely together.

Word2Vec consists of two algorithms that can be trained to generate the embeddings for words, which are the Common Bag of Words (CBOW) and Skip N-gram. In CBOW, the embedding for a given word is generated by combining the distributed representations of the other words that appear in its context. The context is usually defined as a window where the word to be represented is the middle word, and all others are the context. In Skip N-gram, the model tries to model a given word by trying to predict the neighboring words that would appear in its context, i.e., given an input word, the model tries to predict what the words are that either appear immediately before or after.

While both of these models are quite effective for generating embeddings, the Skip-gram model is more powerful as it models every single word in the vocabulary. However, due to this, it also incurs a high overhead compared to the CBOW. In our experiments, we use the default model available which is based on the CBOW architecture.



Since the Word2Vec similarity is also based on the cosine similarity discussed in the previous section, the experiments are somewhat analogous to the HMM2Vec experiments. However, Word2Vec offers some advantages over the HMM2Vec. The main one is that HMM2Vec is based on a Markov model of order one, which means it is limited in the context information it may capture compared to Word2Vec. Another drawback with HMM is the training time, being slower to train for long sequences as in our case.

### 3.2.3. ELMo

ELMo was introduced by Matthew E. Peters et al. in [16] in 2018. ELMo is a bidirectional language model capable of generating contextualized vectors for a particular word by capturing both the semantics as well as the syntax for said word [19]. This means that similar words represented in different contexts are represented differently, which allows us to capture subtle differences in the language. The architecture of ELMo consists of two LSTM networks called the Forward layer and Backward layer. These layers are trained on the tasks of next-word prediction and previous-word prediction, respectively, and work in conjunction, thereby processing input in both directions and capturing meaningful relationships in both directions. The final word embedding is generated by concatenating these contextualized embeddings and scaling them with the help of a normalizing factor. These bidirectional layers are also supported by a task-specific layer on top which helps transform the embeddings from our layers into a suitable form for the task at hand, by usually projecting them into a lower dimension.

Apart from generating contextualized vectors, ELMo offers another advantage over the previously mentioned techniques, wherein we do not necessarily have to handle out-of-vocabulary words as ELMo can create embeddings for these using character-level representations. This makes ELMo quite powerful in tasks that are highly domain-specific and may contain lots of out-of-vocabulary words. More information on ELMo can be found here [16].

### 3.2.4. BERT

BERT, standing for Bidirectional Encoder Representations from Transformers, was introduced in [18] at Google. It is based on the transformer architecture that was introduced in 2017 by Vaswani et al. [17]. The transformer architecture introduced the concept of self-attention, in which each input embedding is represented as three vectors, namely the query, key, and value vectors. These vectors are then used to calculate attention scores among the different units and combined to produce a final sum. This was a key improvement over other models as it meant that we could now capture long-range dependencies without having fixed-size windows. Like ELMo, BERT uses self-attention and also produces contextualized word embeddings.

The architecture of BERT consists of just a stack of encoder models, twelve in number to be precise. The BERT model produces embeddings by following a two-step procedure of pre-training and fine-tuning. During pre-training, the model is trained on a very huge amount of text data and also undergoes training on a couple of specific tasks which are next-sentence prediction (NSP) and masked language modeling (MLM). In MLM, some of the words in the input are masked, and the model is tasked with predicting the original masked word which helps the model learn robustly. In NSP, the model is tasked with deciding which among two of the input sentences precedes the other in a text. This helps the model learn relationships between sentences. Following all this, fine-tuning of these pre-trained parameters is carried out. This fine-tuning is based on the task to be performed and incorporates training on a labeled dataset to optimize the parameters generated in pre-training.

## 3.3. Classifiers

Given a labeled dataset, classification can be defined as the process of predicting the label for a given input. This prediction relies on input features that are analyzed to uncover similarities between the input and previously scrutinized inputs. Numerous

models have been developed and demonstrated to be highly valuable in classification tasks, with SVM being one such example. In our research, we harness the vectors produced by word embedding techniques as features to attempt the classification of malware into their respective classes or families. We employ a variety of classifiers, and the forthcoming section elucidates the underlying principles upon which these classifiers operate. All the classifiers employed herein have demonstrated efficacy in the domain of malware classification using diverse features, as indicated by the works conducted in [8,13,15,19].

### 3.3.1. Support Vector Machine

Support Vector Machines (SVMs) are a set of supervised machine learning algorithms that are hugely popular in classification tasks. In order to maximize the distance between classes, SVM tries to construct a hyperplane that can act as a separator. In [19], the authors used SVM for the classification of malware features generated using word embedding techniques and achieved promising results. SVM can help us identify subtle changes in malware samples as the hyperplane is capable of working in a higher dimensional space with the help of a kernel trick in cases where the data are not linearly separable. There are a number of kernel functions that we can use to achieve this high-dimensional mapping. Another important part of the algorithm is the cost regularization parameter  $C$  which helps avoid overfitting. It does so by allowing some classes into the boundary of the hyperplane. Once the hyperplane is set, the algorithm predicts classes by mapping data points to the high-dimension space and judging the relative position with the hyperplane. More information on SVM and its mathematical proof has been provided in the following sources [24–26].

### 3.3.2. Random Forest

Random Forest (RF) is an ensemble learning technique that uses multiple decision trees to predict a class label, introduced by Breiman in [27]. A decision tree is a supervised machine learning algorithm that works by recursively splitting up the input into smaller similar subsets. While it has its advantages, a single decision tree tends to overfit as the depth and complexity of the tree increase. Random Forest tries to solve this problem by combining the decisions from several decision trees. This is because, since the algorithm bags features along with the observations, the trees tend to protect each other from individual errors and avoid overfitting. The algorithm works by assigning samples at random to different decision trees and then averaging the results. The class label predicted is the one that receives the most votes from the individual trees. A good explanation of how to use these algorithms can be found at [28].

### 3.3.3. k-Nearest Neighbors

k-Nearest Neighbors is a simple supervised machine learning algorithm that makes use of a sample's neighboring data points to predict where it belongs. First, the  $k$  nearest samples are taken, and then, the distance between them and the input sample to be predicted is calculated. The distance measure is a parameter that we can decide based on the use case such as Euclidean distance or Minkowski distance, and this value is what the prediction is based on. The algorithm does not include a training phase which is why is referred to as a lazy classifier. When implementing kNN, the value of  $k$  is of paramount concern, as too small values can lead to overfitting, while very large values can lead to performance and accuracy degradation [19].

### 3.3.4. Convolutional Neural Networks

Neural networks are algorithms modeled to work like the human brain. A Convolutional Neural Network or a CNN is a type of feedforward deep neural network introduced in [29] that has found success in several image classification tasks. A CNN consists of several layers including an input layer, several hidden layers, and an output layer. Among these,

the convolutional layer is the core building block. It contains a set of filters that slide over an input to extract relevant features by exploiting any spatial patterns.

### 3.4. Dataset

The dataset employed in this project encompasses a collection of Windows executable files. These executables underwent analysis within the Buster Sandbox Analyzer, leading to the extraction of API logs that the malware invoked during execution. In total, 782 samples were extracted from the initial raw dataset, subsequently divided into two distinct sets. One set consisted of approximately 583 samples categorized by types, while the other, comprising about 492 samples, was categorized by families. These datasets serve as the foundation for the experiments detailed in the forthcoming sections. For the category classification, the 583 samples were segregated into 11 distinct malware categories. The distribution of sample counts for each of these categories within the dataset is provided in Table 1. Additionally, in the case of family classification, the samples were classified into seven malware families. The corresponding sample counts for each of these families within the dataset can be found in Table 2.

**Table 1.** Number of samples for each malware category.

Malware Category	Sample Count
Adware	50
Backdoor	53
Modifier	52
PWS	50
Rogue	53
Tool	60
Trojan	53
Trojan Downloader	52
Trojan Monitoring Service	53
Virus	55
Worm	52

**Table 2.** Number of samples for each malware family.

Malware Family	Sample Count
Adload	70
Bancos	71
Onlinegames	70
VBinject	70
Vundo	71
Winwebsec	70
Zwangi	70

Each entry in the dataset comprises an API call log sequence corresponding to a specific malware instance. These logs generally contain the API name alongside supplementary details like file paths. To streamline the information, we preprocessed these files to retain solely the API call names. Consequently, for each sample, we processed a sequence of API calls executed during its runtime. The aggregate count of unique API calls encountered across the datasets tallies up to 94. Although employing the complete array of calls could yield valuable insights, it would entail extensive processing time when generating embeddings. Thus, it becomes pivotal to cherry-pick the most pertinent or crucial calls. The importance of an individual API call is determined by how frequently it appears in the dataset. In our experiments, we emphasized the most commonly occurring calls—the top 20 and top 40—which created two distinct groups of features. The percentage of total calls covered by these feature sets in relation to the entire range of calls is detailed in Table 3. Remarkably, even when focusing exclusively on the top 40 calls, we managed to capture



over 99% of the entire set of calls. Conducting experiments with the complete API call values did not result in enhanced performance. As a result, we chose to present only the experiments involving these two sets of percentage distributions.

**Table 3.** Percentage distribution of top calls.

Calls	Count (Percentage)
Total (Category)	233,539 (100%)
Top 40 (Category)	232,080 (99.37%)
Top 20 (Category)	223,698 (95.78%)
Total (Family)	160,813 (100%)
Top 40 (Family)	159,987 (99.48%)
Top 20 (Family)	154,868 (96.3%)

In both the cases of families and categories, we observed that the same API call (VirtualAllocEx) had the highest frequency.

#### 4. Results

Presented here are the outcomes of both experiments, that is, the classification of samples into their respective malware categories, as well as the classification into their corresponding malware families.

You can find supplementary details in the Appendix A, including confusion matrices, loss metrics, and training accuracy data from all our experiments.

##### 4.1. Malware Category Classification

In this section, we present a series of experiments involving the classification of malware types. These experiments were conducted using hybrid classification methodologies.

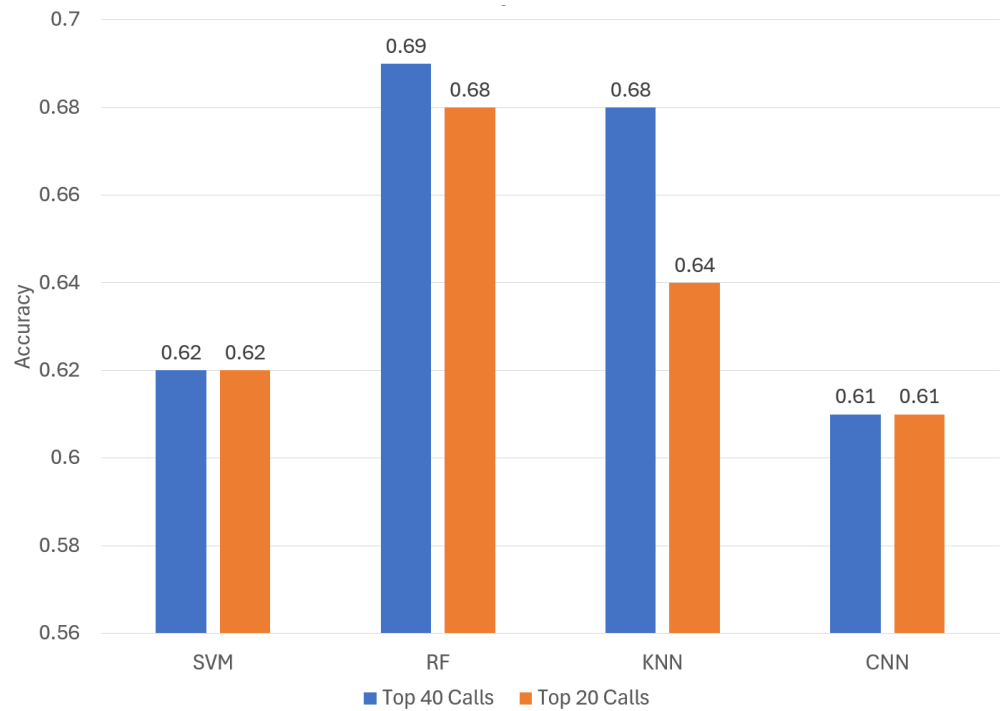
###### 4.1.1. HMM2Vec

For the HMM2Vec experiments, the best accuracy we achieved was with the Random Forest classifier at 0.69. kNN performed very closely, reaching 0.68, followed by SVM at 0.62. The worst performance was from CNN with 0.61 accuracy.

From these results, we observed a common pattern where the Worm category was continually misidentified with all classifiers. This may be because of the high amount of variance within the samples in this class. With respect to the calls retained, for HMM2Vec, we saw that the Top 40 calls performed better than the Top 20 calls, although not by too big of a margin. Figure 2 shows the accuracy achieved in both cases. Table 4 shows additional metrics for the individual categories.

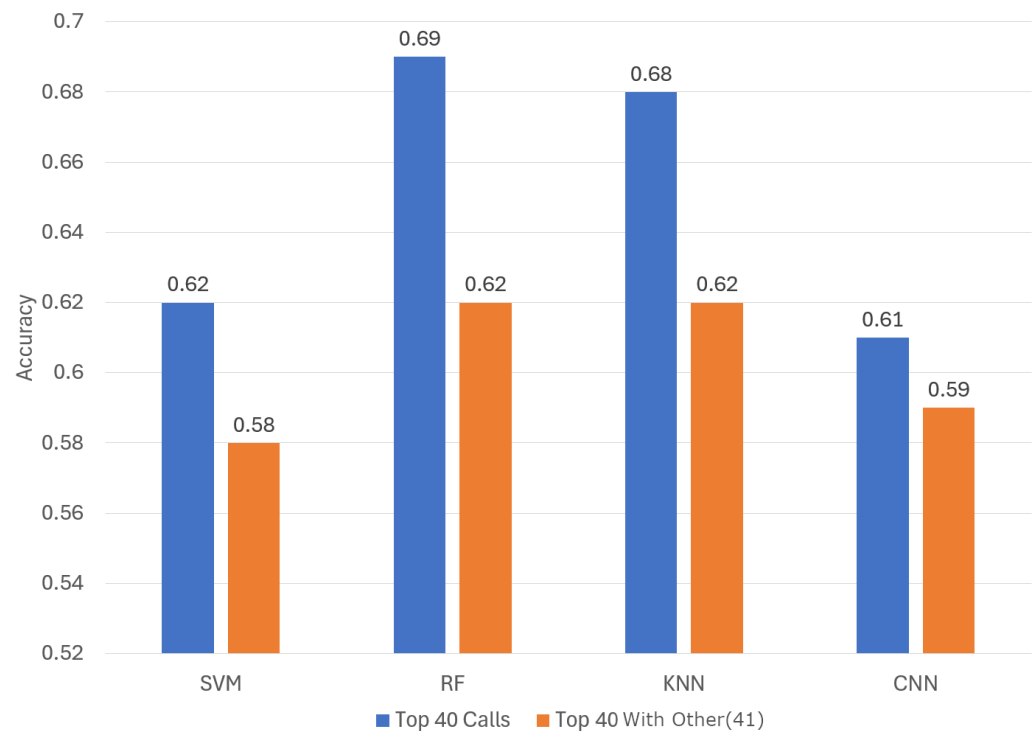
**Table 4.** Classification report for HMM2Vec category classification.

Category	Precision	Recall	F1-Score
Adware	0.64	0.90	0.75
Backdoor	0.50	0.27	0.35
PWS	0.82	0.90	0.86
Modifier	0.82	0.90	0.86
Rogue	0.89	0.73	0.80
Tool	0.67	0.67	0.67
Trojan	0.73	0.73	0.73
Trojan Downloader	0.78	0.70	0.74
Trojan Spy	0.58	0.64	0.61
Worm	0.50	0.36	0.42
Virus	0.64	0.90	0.75



**Figure 2.** Top 20 calls vs. Top 40 calls for HMM2Vec category classification.

For experiments performed where we replaced all calls outside of the selected ones with a constant symbol, the performance decreased. This may be because the number of these calls that lie outside our selected set may add up to a higher number masking what the sequence actually represents in the output matrices. For these experiments, the best accuracy we achieved was with the Random Forest classifier at 0.62. Figure 3 shows the accuracy of this approach compared to the one where we deleted the calls.



**Figure 3.** Top 40 calls vs. Top 40 calls for HMM2Vec category classification.

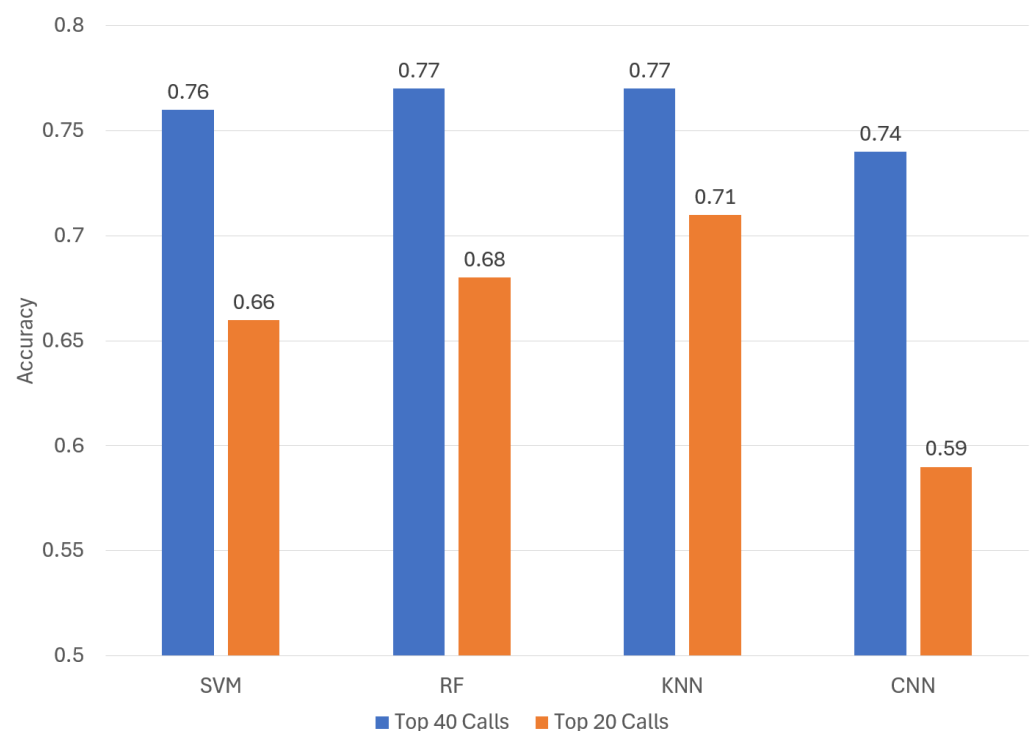
#### 4.1.2. Word2Vec

For the Word2Vec experiments, the best accuracy we achieved was with the Random Forest and kNN classifiers at 0.77. Overall, this was the highest accuracy achieved in the task of malware category classification. Word2Vec-SVM performed slightly worse at 0.76, whereas CNN was last with about 0.74.

Once more, we observed that the Worm class exhibited the lowest level of accuracy. Nevertheless, a comparison with the outcomes of the HMM2Vec experiments reveals a noteworthy trend: Word2Vec exhibited improved accuracy in predicting the Backdoor category. This improvement likely contributed to the overall boost in accuracy observed. In terms of the retained calls, our findings indicate an interesting difference between Word2Vec and HMM2Vec. Specifically, in the case of Word2Vec, the Top 40 calls displayed markedly superior performance compared to the Top 20 calls, a contrast to the behavior observed in HMM2Vec experiments. The accuracy achieved for both scenarios is illustrated in Figure 4. Table 5 shows additional metrics for the individual categories.

**Table 5.** Classification report for Word2Vec category classification.

Category	Precision	Recall	F1-Score
Adware	0.80	0.80	0.80
Backdoor	0.58	0.64	0.61
PWS	0.91	1.00	0.95
Modifier	0.80	0.80	0.80
Rogue	0.82	0.82	0.82
Tool	0.53	0.67	0.59
Trojan	0.83	0.91	0.87
Trojan Downloader	0.89	0.70	0.84
Trojan Spy	0.89	0.73	0.80
Worm	0.83	0.45	0.59
Virus	0.75	0.90	0.82



**Figure 4.** Top 20 calls vs. Top 40 calls for Word2Vec category classification.

#### 4.1.3. ELMo

In the context of the ELMo experiments, the RF model once again emerged as the top performer, attaining an accuracy of 0.77. However, it is worth noting that kNN did not fare as well, lagging behind at 0.71. On a similar note, the performance of SVM and CNN remained somewhat less satisfactory, recording accuracies of approximately 0.70 and 0.67, respectively.

As observed with Word2Vec, ELMo also demonstrated improved performance when utilizing the Top 40 calls, compared to its performance with the Top 20 calls. The additional metrics for individual categories are instead shown in Table 6.

**Table 6.** Classification report for ELMo-RF category classification.

Category	Precision	Recall	F1-Score
Adware	0.80	0.80	0.80
Backdoor	0.80	0.36	0.50
PWS	1.00	1.00	1.00
Modifier	0.82	0.90	0.86
Rogue	0.80	0.73	0.76
Tool	0.58	0.58	0.58
Trojan	0.75	0.82	0.78
Trojan Downloader	0.75	0.90	0.81
Trojan Spy	0.80	0.72	0.75
Worm	0.50	0.58	0.53
Virus	0.63	0.91	0.75

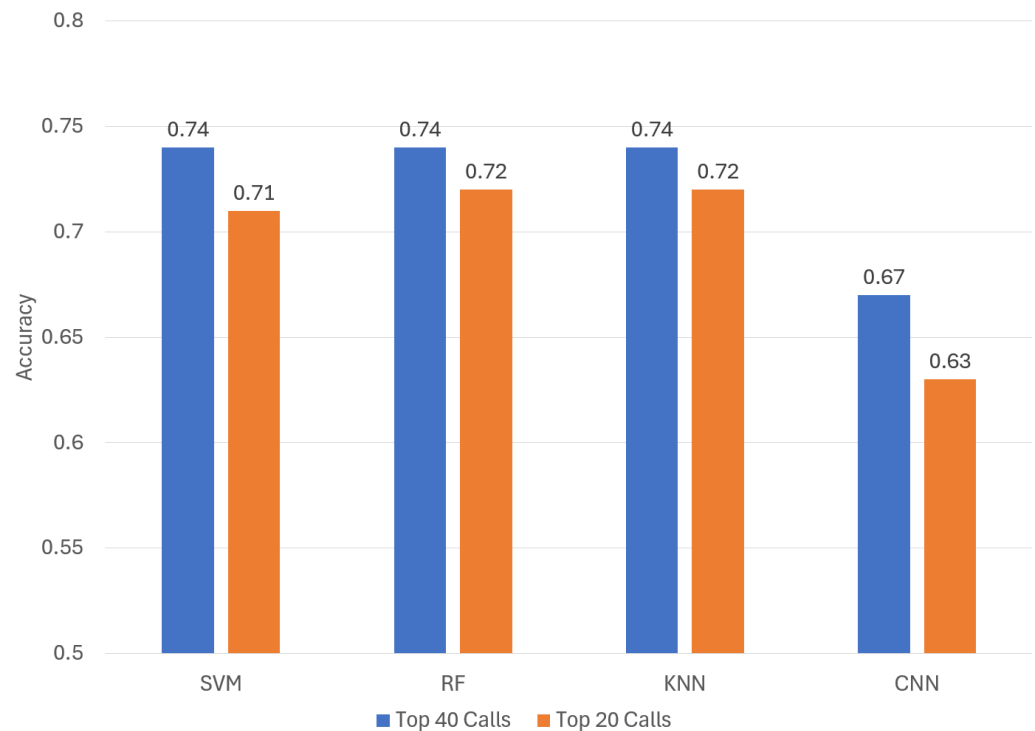
#### 4.1.4. BERT

The methodology applied to BERT closely mirrors that of ELMo, yielding similar outcomes. In a manner reminiscent of earlier findings, both kNN and RF achieved the highest accuracy, standing at 0.74. Although this accuracy fell short of ELMo's performance, an intriguing twist was the superior performance of SVM in the context of BERT, where it attained an accuracy of 0.74. Conversely, CNN lagged behind the other three techniques, registering an accuracy of 0.67.

The accuracy results for the retained calls are depicted in Figure 5. Although the Top 40 calls exhibited slightly superior performance once more, the margin is not substantial. For a more detailed breakdown, please refer to Table 7, which presents additional metrics for individual categories.

**Table 7.** Classification report for BERT-RF category classification.

Category	Precision	Recall	F1-Score
Adware	0.82	0.90	0.86
Backdoor	0.57	0.36	0.44
PWS	0.91	1.00	0.95
Modifier	0.77	1.00	0.87
Rogue	1.00	0.73	0.84
Tool	0.54	0.58	0.56
Trojan	0.58	0.64	0.61
Trojan Downloader	0.89	0.80	0.84
Trojan Spy	0.67	0.91	0.77
Worm	0.62	0.73	0.67
Virus	1.00	0.50	0.67

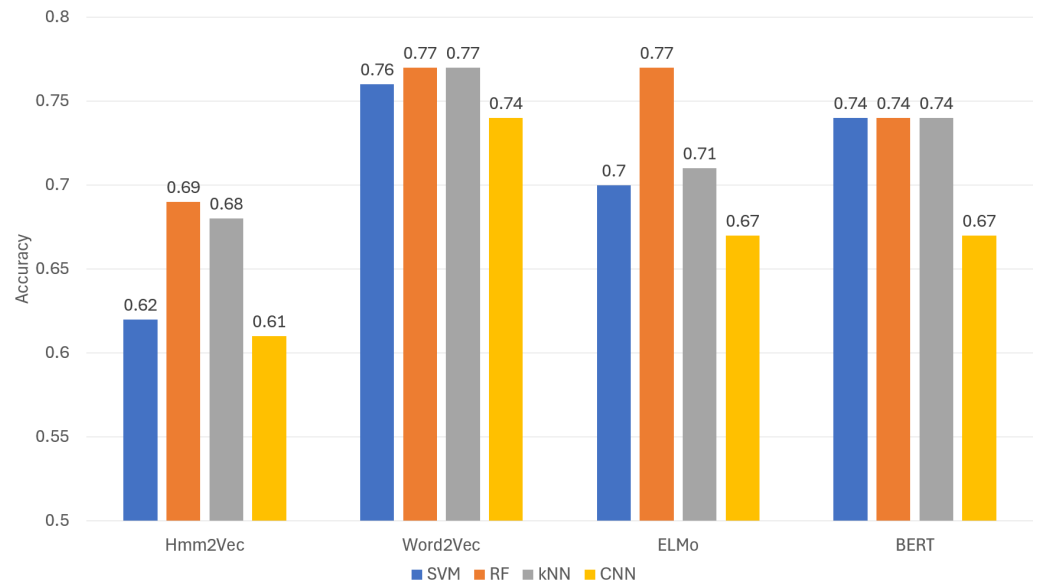


**Figure 5.** Top 20 calls vs. Top 40 calls for BERT category classification.

#### 4.1.5. Discussion on Malware Category Classification

On the whole, the classification of malware categories falls short in comparison to the family-level classification. This outcome is expected, given the increased complexity of predicting malware categories, where individual malware instances within a category can belong to diverse families and exhibit distinct behaviors. Our dataset underscores this challenge—across 11 categories, there are approximately 90 distinct families, a factor that likely contributes to the relatively lower accuracy scores observed. Figure 6 visually represents the achieved accuracies for the various hybrid techniques. Notably, instances of misclassification are prevalent in the Worm and Backdoor categories. The highest accuracy attained stands at around 0.77, achieved by the Word2Vec-RF combination. ELMo paired with RF also yields commendable results. Among the embedding techniques, both Word2Vec and ELMo perform remarkably well, producing comparable outcomes. Additionally, a significant observation emerges, that is, agreement among the embedding techniques in designating RF as the most effective classifier. Furthermore, a trend emerges where RF and kNN generally perform on par with each other, while SVM and CNN trail slightly behind. SVM showcases a comparable performance to RF and kNN in conjunction with BERT. This could be attributed to SVM's capability to handle high-dimensional data, which aligns well with BERT's complex embeddings.





**Figure 6.** Comparison of hybrid machine learning approaches for malware category.

#### 4.2. Malware Family Classification

The conducted experiments revolve around seven malware families as outlined in Section 3.4. Due to the consistent superiority of the Top 40 calls over the Top 20 calls across all our hybrid techniques, we have excluded the features associated with the Top 20 Calls for this specific segment of our research.

##### 4.2.1. HMM2Vec

In the context of the HMM2Vec experiments, the highest accuracy was attained using Random Forest, reaching 0.85. The remaining classifiers displayed closely aligned performance, achieving accuracies of 0.84, 0.84, and 0.82 for SVM, kNN, and CNN, respectively. The additional metrics for individual families are shown in Table 8.

We observed a recurring pattern of misclassification, particularly involving samples from various malware families, which were wrongly categorized as belonging to the Bancos and Onlinegames families. This trend was evident across multiple experiments and might signify the necessity for a more robust feature selection technique to enhance classification accuracy.

**Table 8.** Classification report for best HMM2Vec family classification.

Category	Precision	Recall	F1-Score
Adload	0.93	0.93	0.93
Bancos	0.69	0.73	0.71
Onlinegames	0.81	0.93	0.87
Vbinject	0.92	0.79	0.85
Vundo	0.75	0.86	0.80
Winwebsec	1.00	0.79	0.88
Zwangi	0.93	0.93	0.93

#### 4.2.2. Word2Vec

Out of all the word embedding techniques, the Word2Vec-RF combination emerged as the top performer for family classification, achieving an impressive accuracy of 0.93. This result is particularly noteworthy because this approach demonstrates its ability to predict a substantial number of samples effectively. The other classifiers also demonstrated commendable performance, with SVM, kNN, and CNN achieving accuracies of 0.90, 0.92, and 0.89, respectively. The additional metrics for individual families are shown in Table 9.

**Table 9.** Classification report for best Word2Vec family classification.

Category	Precision	Recall	F1-Score
Adload	0.93	1.00	0.97
Bancos	0.93	0.93	0.93
Onlinegames	0.93	0.93	0.93
Vbinject	0.87	0.93	0.90
Vundo	0.93	0.87	0.90
Winwebsec	0.92	0.86	0.89
Zwangi	1.00	0.93	0.97

#### 4.2.3. ELMo

In the context of the ELMo experiments, the achieved accuracies, ranked in ascending order, were as follows: 0.86 for CNN, 0.9 for both SVM and k-Nearest Neighbors (kNNs), and finally, 0.91 for Random Forest. The additional metrics for individual families are shown in Table 10.

**Table 10.** Classification report for best ELMo family classification.

Category	Precision	Recall	F1-Score
Adload	1.00	1.00	1.00
Bancos	0.82	0.93	0.87
Onlinegames	0.83	0.71	0.77
Vbinject	0.87	0.93	0.90
Vundo	1.00	1.00	1.00
Winwebsec	0.97	0.93	0.90
Zwangi	1.00	0.86	0.92

#### 4.2.4. BERT

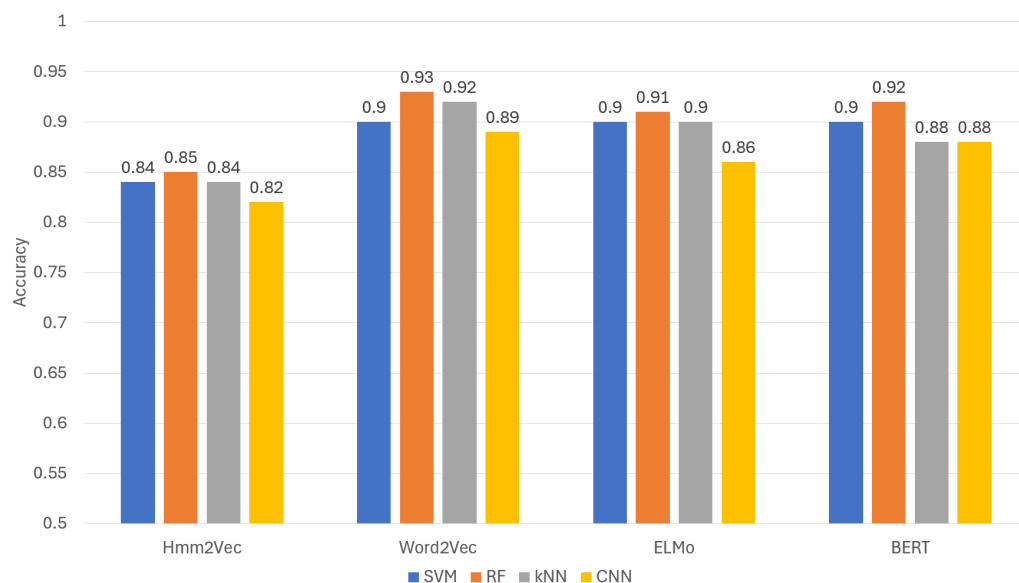
The BERT model attained its highest accuracy of 0.92 when paired with RF. In line with the earlier category experiments, it is noteworthy that SVM outperformed kNN, achieving an accuracy of 0.90 compared to kNN's 0.88. Lastly, CNN concluded with an accuracy of approximately 0.88. The additional metrics for individual families are shown in Table 11.

**Table 11.** Classification report for best BERT family classification.

Category	Precision	Recall	F1-Score
Adload	1.00	0.93	0.97
Bancos	0.87	0.87	0.87
Onlinegames	0.93	1.00	0.97
Vbinject	0.93	1.00	0.97
Vundo	0.93	0.93	0.93
Winwebsec	0.85	0.79	0.82
Zwangi	0.93	0.93	0.93

#### 4.2.5. Discussion on Malware Family Classification

By employing hybrid techniques, we achieved improved accuracy levels for the malware family classification when compared with the malware category classification results. The pinnacle was reached at 0.93 accuracy when utilizing Word2Vec-RF. Figure 7 visually portrays the attained accuracies across the various techniques. In the context of HMM2Vec, the highest accuracy, around 0.85, was achieved in combination with RF. This accuracy figure stands as the lowest among all our employed word embedding techniques. We observed that in instances where the count of distinct API calls was relatively low, the resultant feature vectors contained comparatively less information than those obtained from other techniques. This could possibly explain the lower scores observed. Furthermore, a prevalent misclassification pattern involved samples initially labeled as Winwebsec, which were often predicted as belonging to the Bancos or Onlinegames families.



**Figure 7.** Comparison of hybrid machine learning approaches for malware family.

## 5. Conclusions

In this study, we conducted an array of experiments to assess the efficacy of utilizing API call information as a feature for both malware category and family classification. Our approach involved testing hybrid machine learning techniques, wherein diverse word embedding methods were combined to engineer features. We evaluated the performance of four distinct embedding techniques—HMM2Vec, Word2Vec, ELMo, and BERT—paired with four classifiers, namely SVM, RF, kNN, and CNN. The results of our experiments unequivocally demonstrate Word2Vec's superior performance, achieving the highest accuracy of 0.93 for family classification and 0.77 for category classification. However, Word2Vec's training time was relatively shorter compared to other techniques. Interestingly, BERT's performance did not surpass Word2Vec, which was somewhat anticipated. This might be attributed to the dataset size. BERT necessitates a substantial training dataset to grasp meaningful relationships, implying that an increase in sample size could potentially lead to improved performance. Another potential factor is the input length limitation of 512 tokens in BERT, where we employ the initial 512 tokens in our sequences. Given the length of some sequences, we could be losing crucial information. Utilizing alternate token selections, such as the last 512 tokens or a custom approach, might yield better outcomes. Across classifiers, a consistent trend emerged: Random Forest consistently outperformed other classifiers. Our documented outcomes underscore the valuable potential of API calls as a robust feature for effective malware classification.

For future work, the scope of the experiments conducted herein can be extended to encompass a broader range of malware categories and families. Exploring analogous experiments with alternate features, such as byte n-grams or combinations thereof, could yield promising results. In addition, ensemble learning could be tested to understand its benefit with these specific types of data. Also, additional study on the effect of obfuscation techniques on the embeddings could be an interesting follow-up to this work. While this research relied on frequency-based feature selection, techniques like TF-IDF [7] and Fisher score [30] have shown efficacy in similar tasks, and employing them could enhance scores, especially for HMM2Vec, by eliminating irrelevant features. Furthermore, an exploration into other embedding methods is warranted. GPT-based models, which have gained substantial traction, merit investigation to gauge their performance against the techniques assessed in this study.

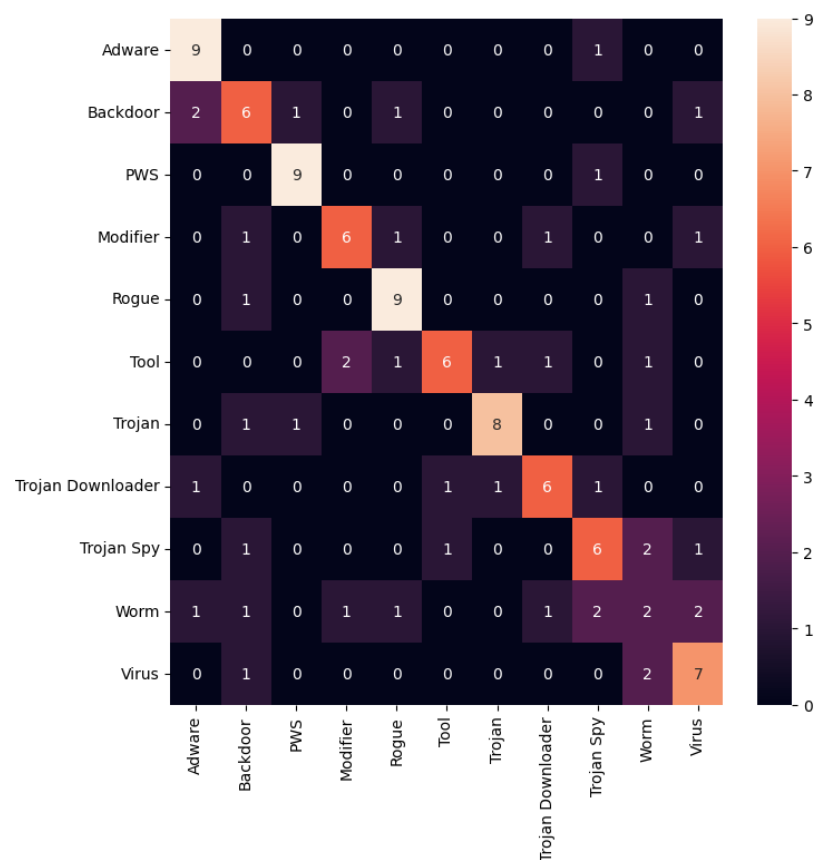
**Author Contributions:** Conceptualization, F.D.T.; methodology, F.D.T.; software, S.A.; validation, F.D.T.; formal analysis, F.D.T.; investigation, F.D.T.; resources, S.A.; data curation, F.D.T.; writing—original draft preparation, S.A.; writing—review and editing, F.D.T.; visualization, F.D.T.; supervision, F.D.T. All authors have read and agreed to the published version of the manuscript.

**Funding:** This research received no external funding.

**Data Availability Statement:** The dataset used in this article can be found at <https://drive.google.com/drive/folders/1eDv1gSO-RUjLiZobprOyoflNMMy65Oe4?usp=sharing> (accessed on 25 June 2024).

**Conflicts of Interest:** The authors declare no conflicts of interest.

### Appendix A. Additional Results



**Figure A1.** Confusion matrix for HMM2Vec-SVM category classification.

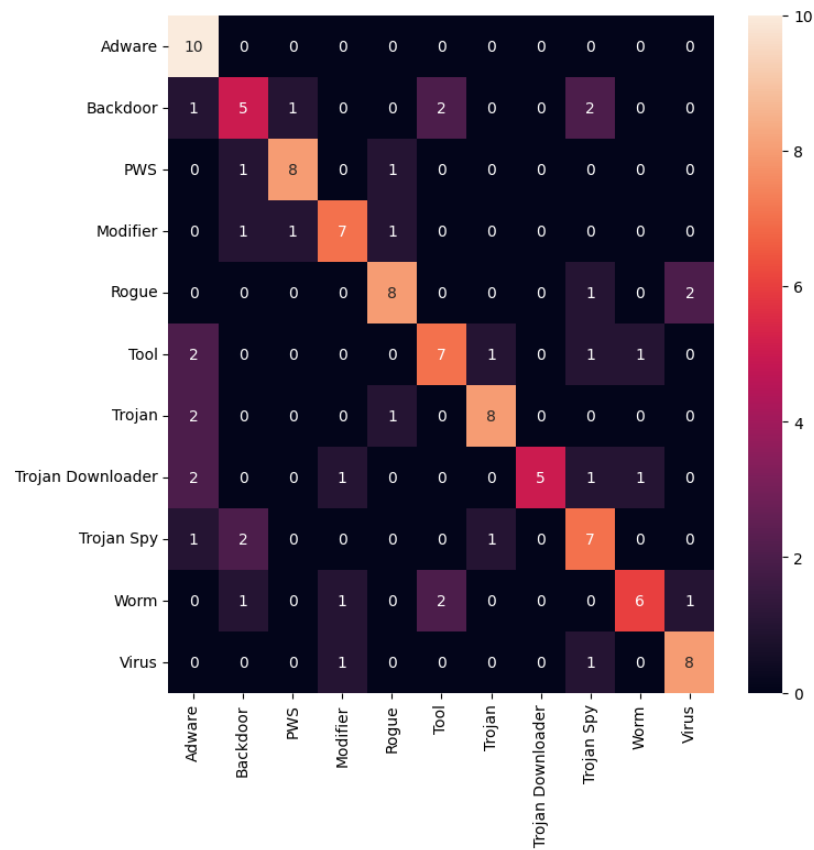


Figure A2. Confusion matrix for HMM2Vec-KNN category classification.

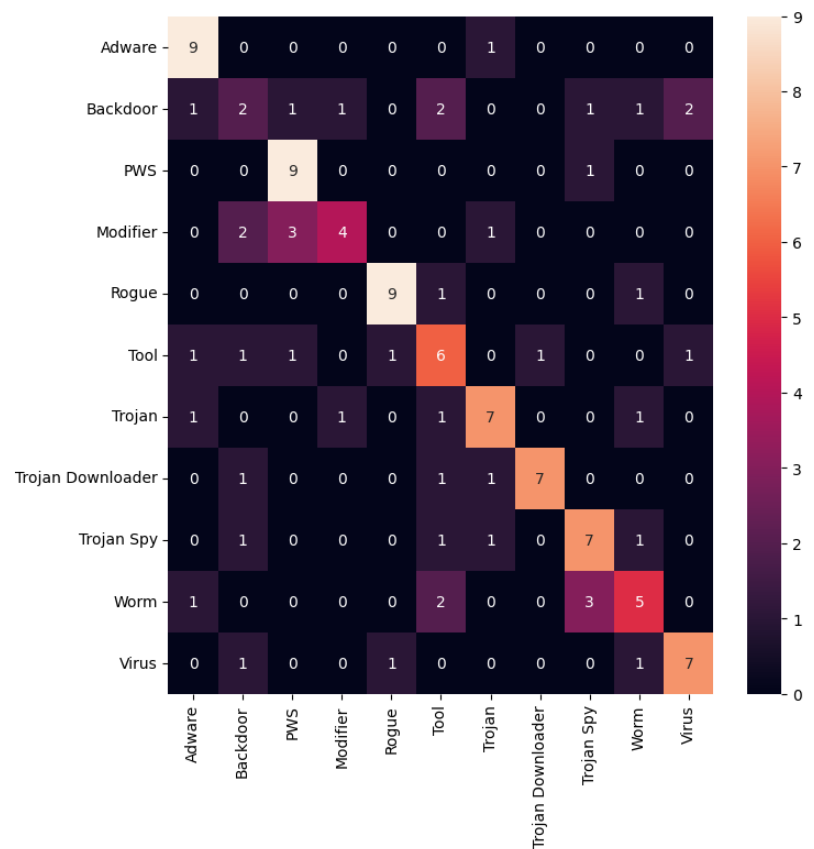


Figure A3. Confusion matrix for HMM2Vec-CNN category classification.





(a) Accuracy

(b) Loss

Figure A4. Training model accuracy vs. loss for Hmm2Vec-CNN for category.

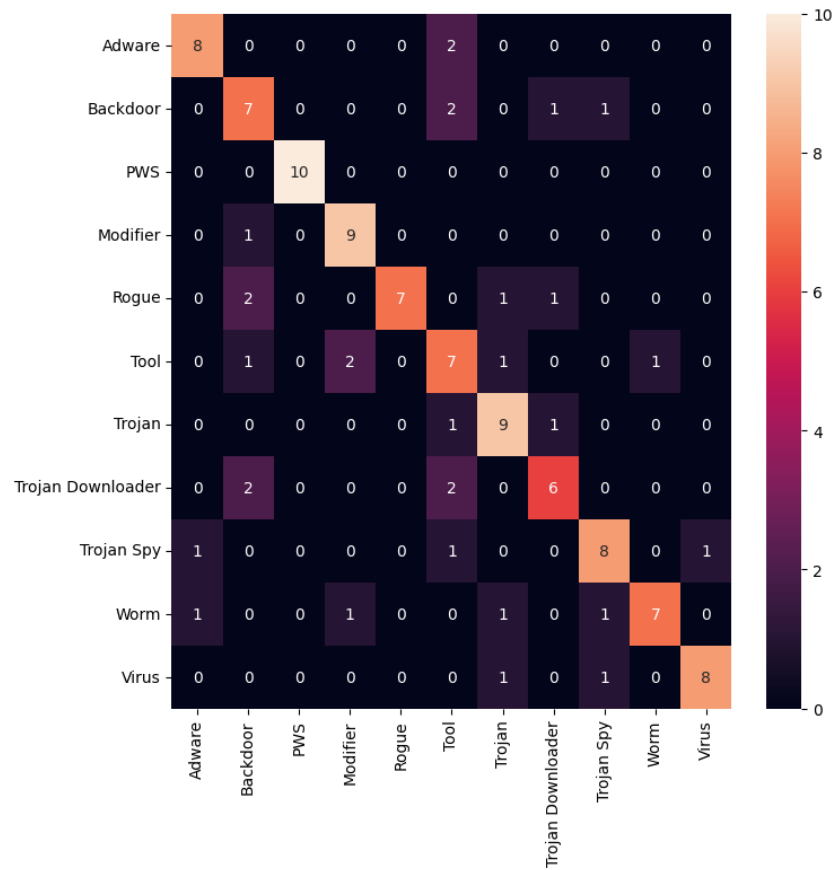


Figure A5. Confusion matrix for Word2Vec-SVM category classification.

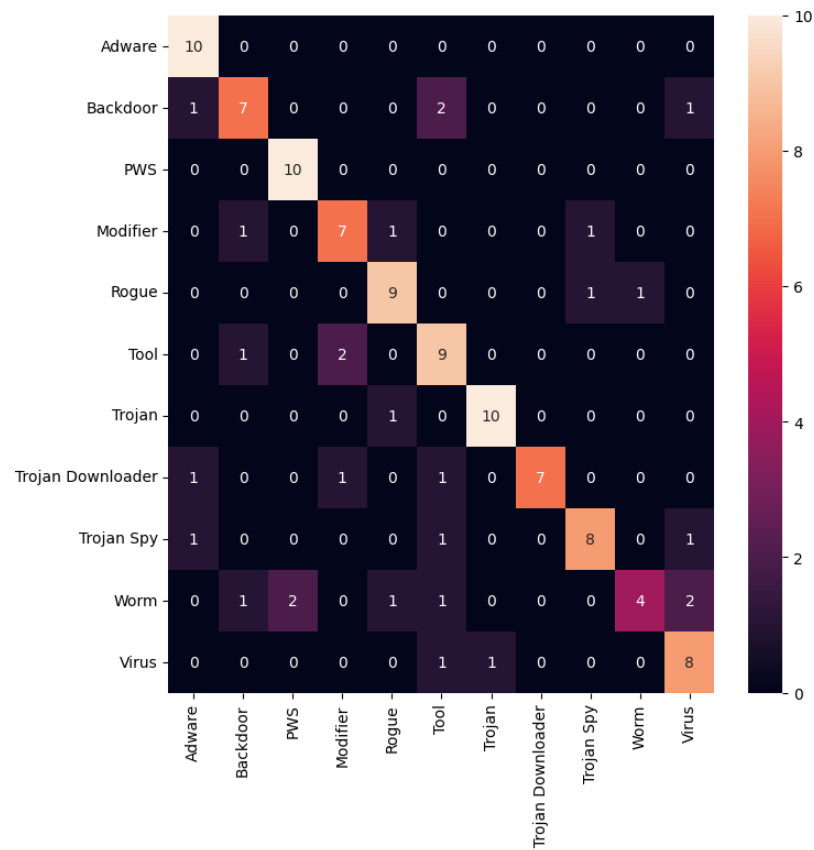


Figure A6. Confusion matrix for Word2Vec-KNN category classification.

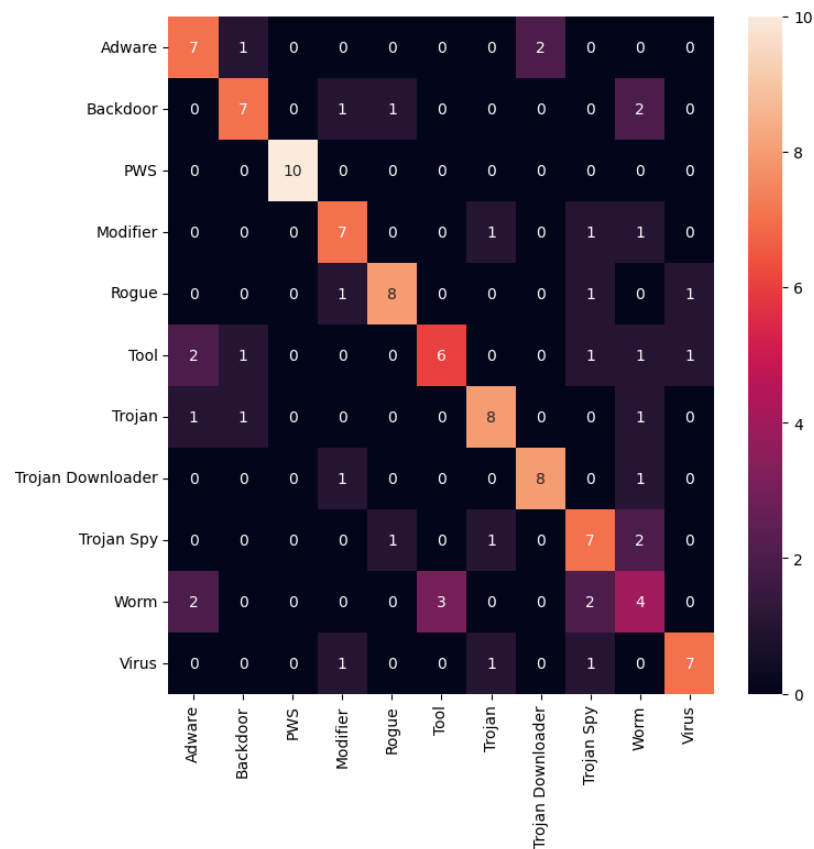
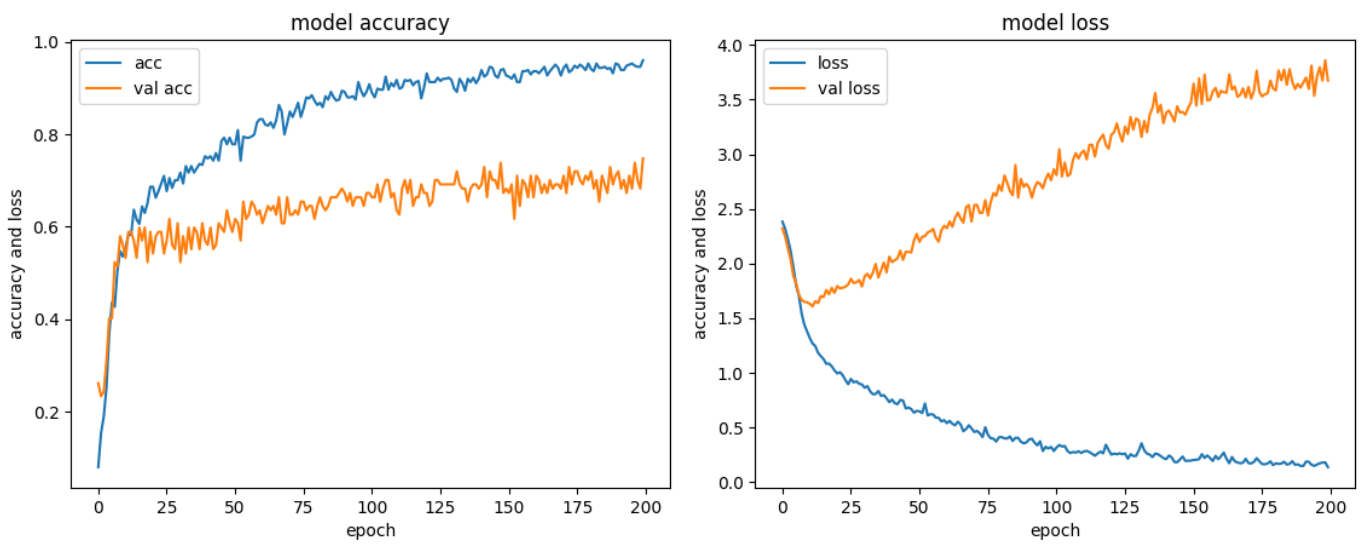


Figure A7. Confusion matrix for Word2Vec-CNN category classification.



(a) Accuracy (b) Loss

Figure A8. Training model accuracy vs. loss for Word2Vec-CNN for category.

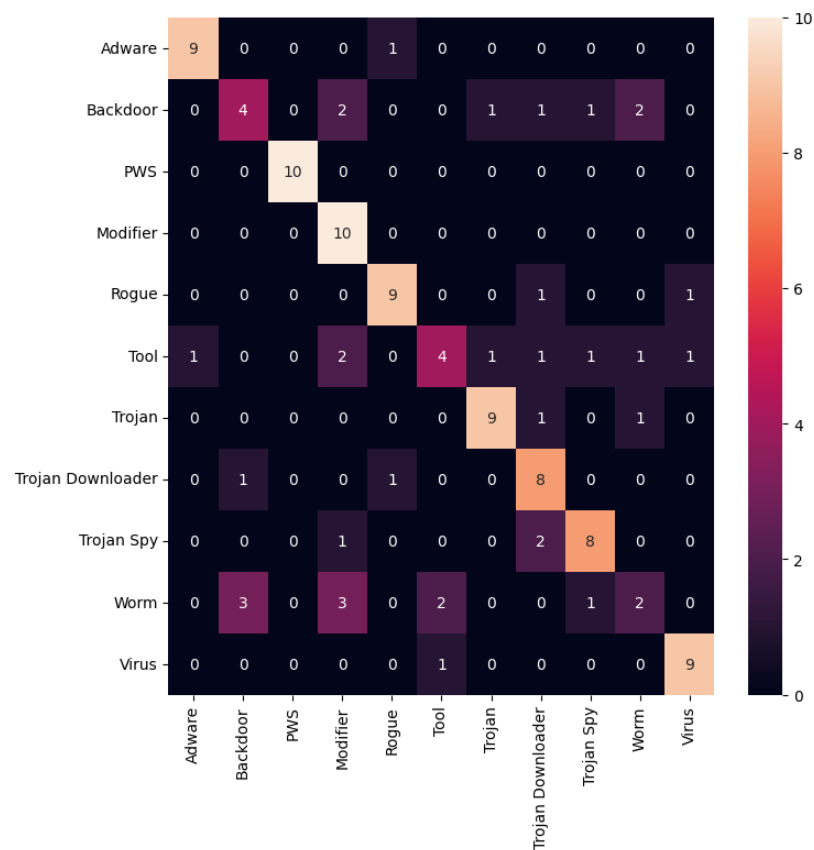


Figure A9. Confusion matrix for ELMo-SVM category classification.

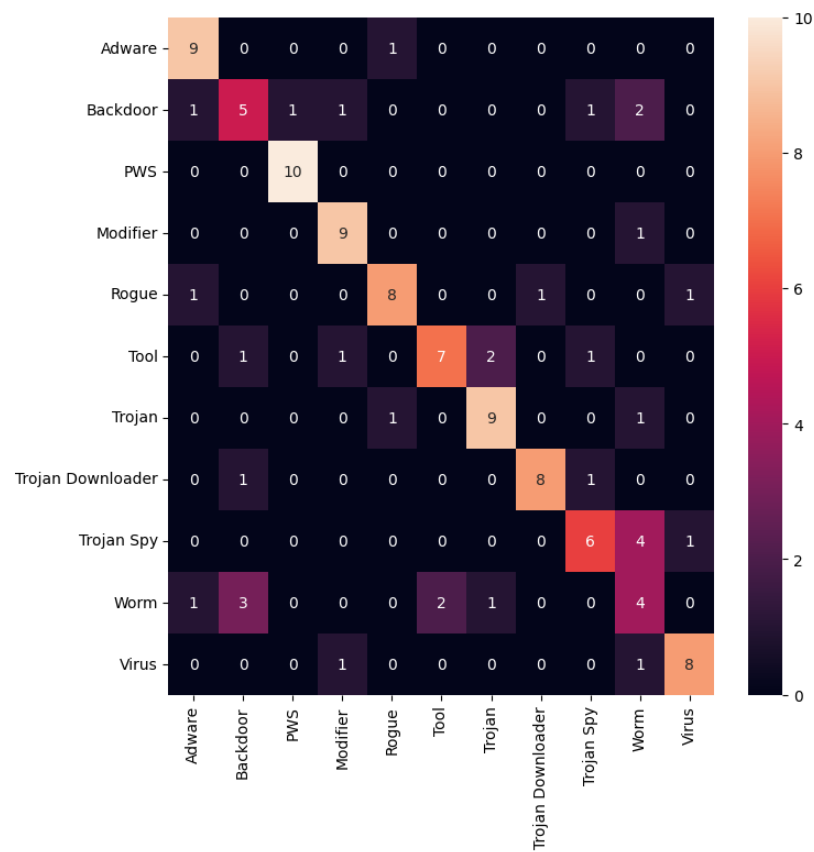


Figure A10. Confusion matrix for ELMo-KNN category classification.

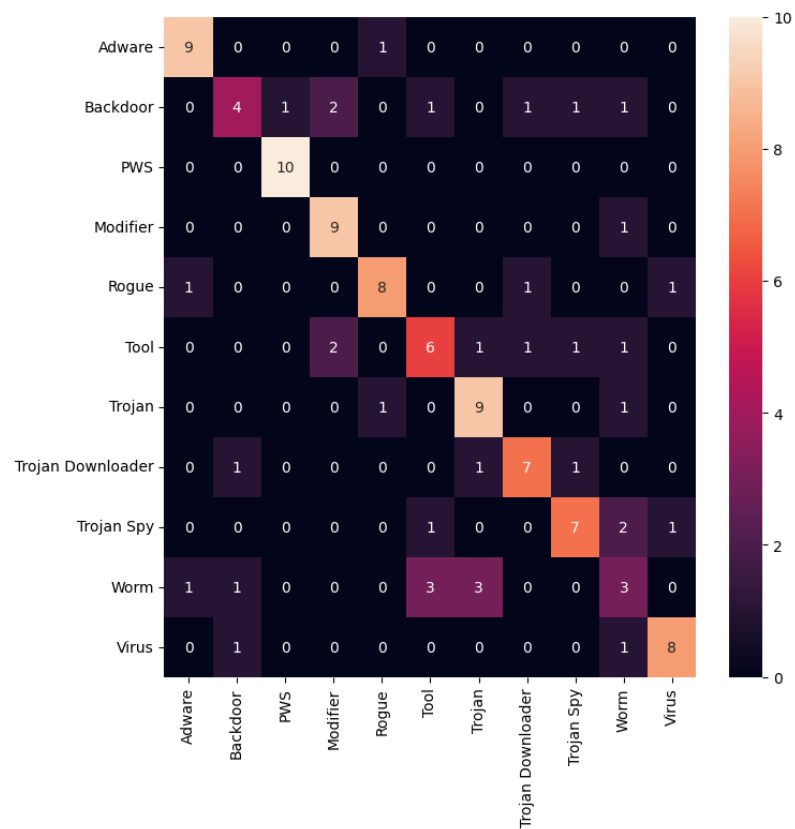


Figure A11. Confusion matrix for ELMo-CNN category classification.

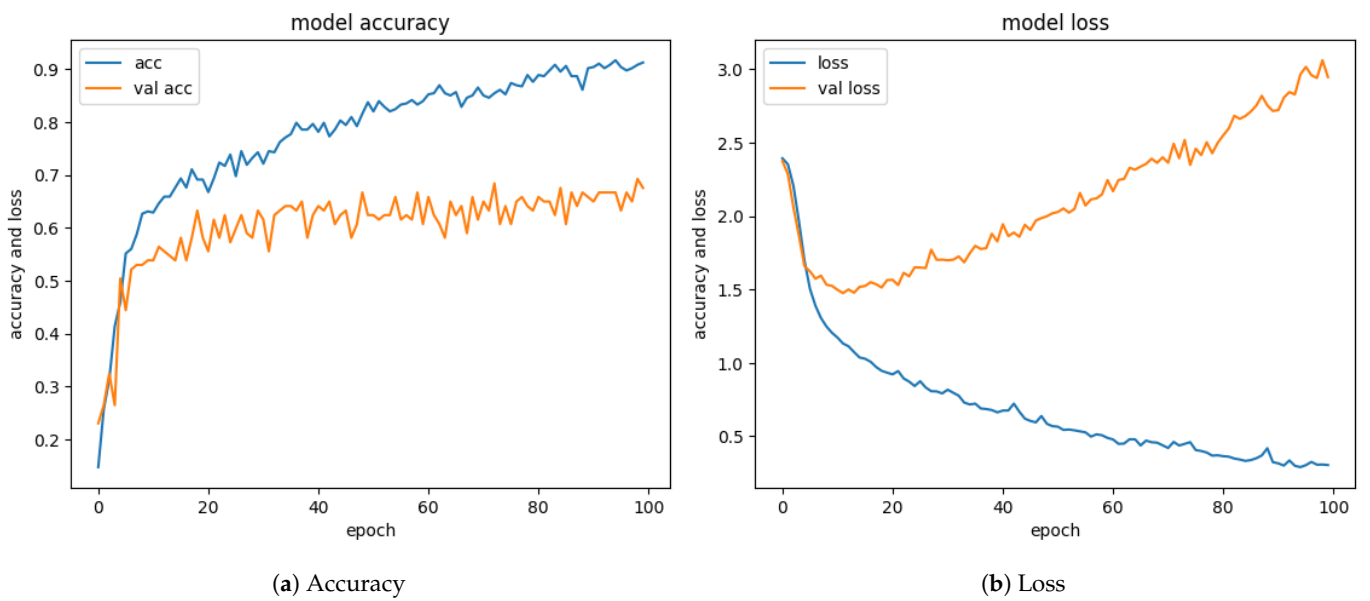


Figure A12. Training model accuracy vs. loss for ELMo-CNN for category.

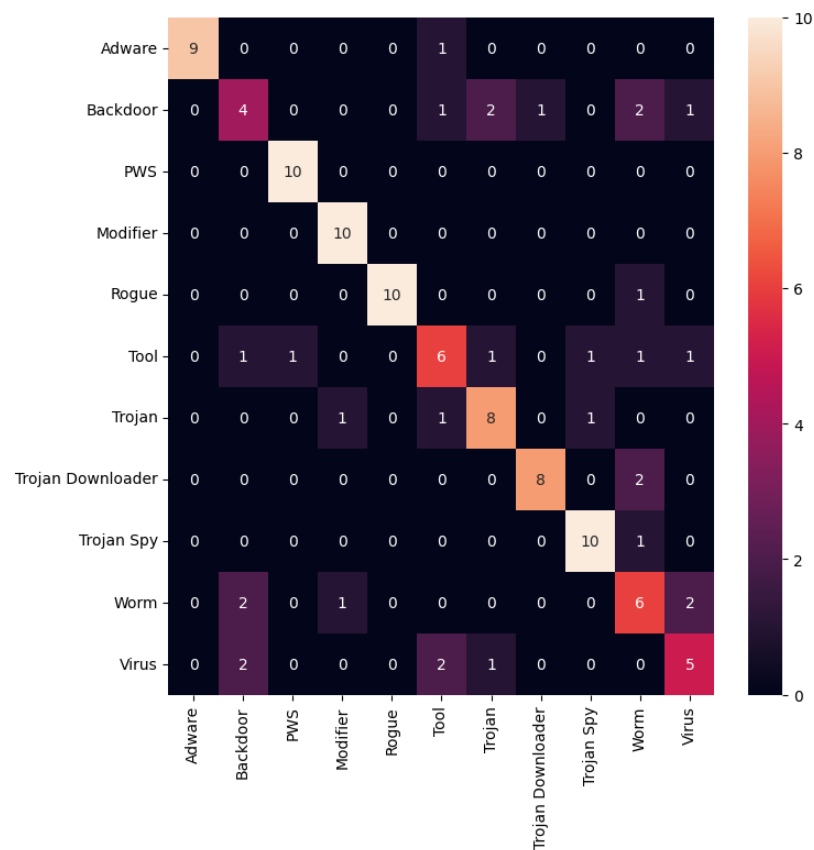


Figure A13. Confusion matrix for BERT-SVM category classification.



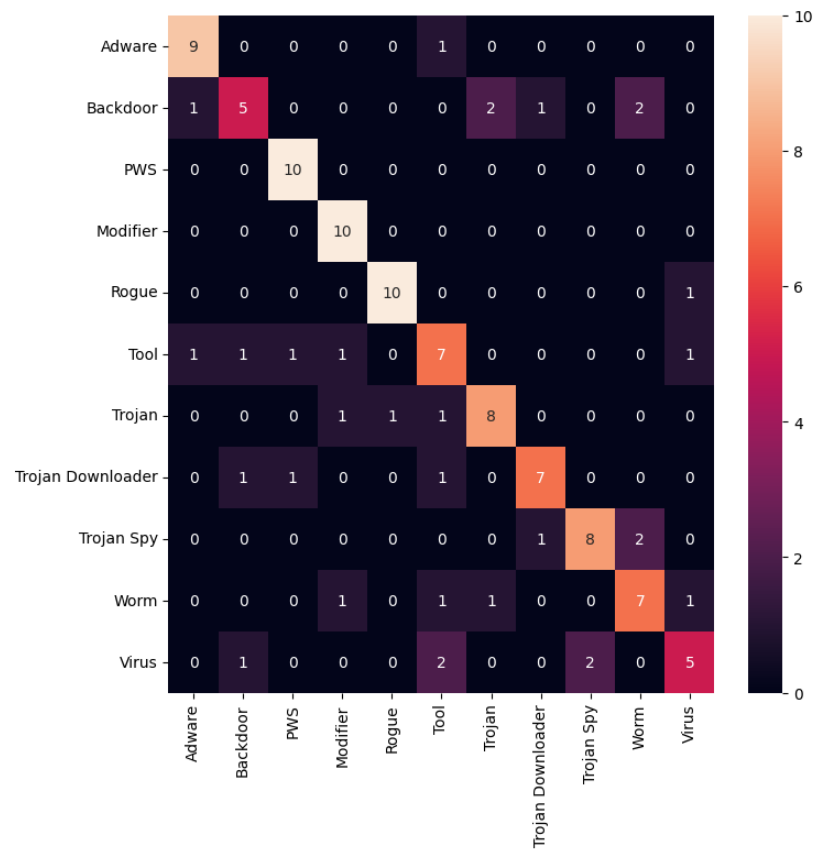


Figure A14. Confusion matrix for BERT-KNN category classification.

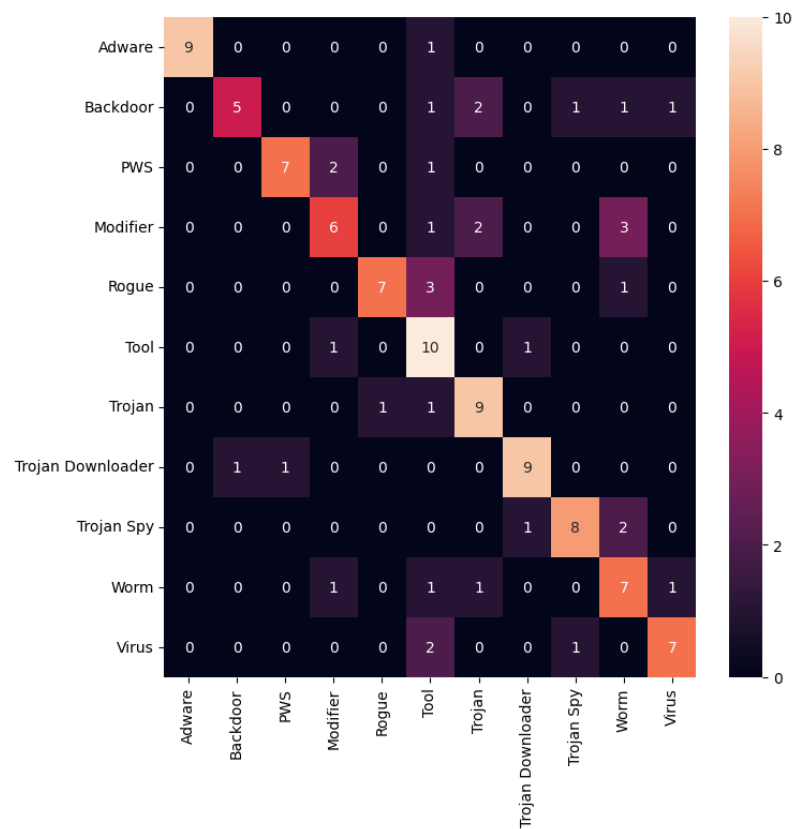


Figure A15. Confusion matrix for BERT-CNN category classification.

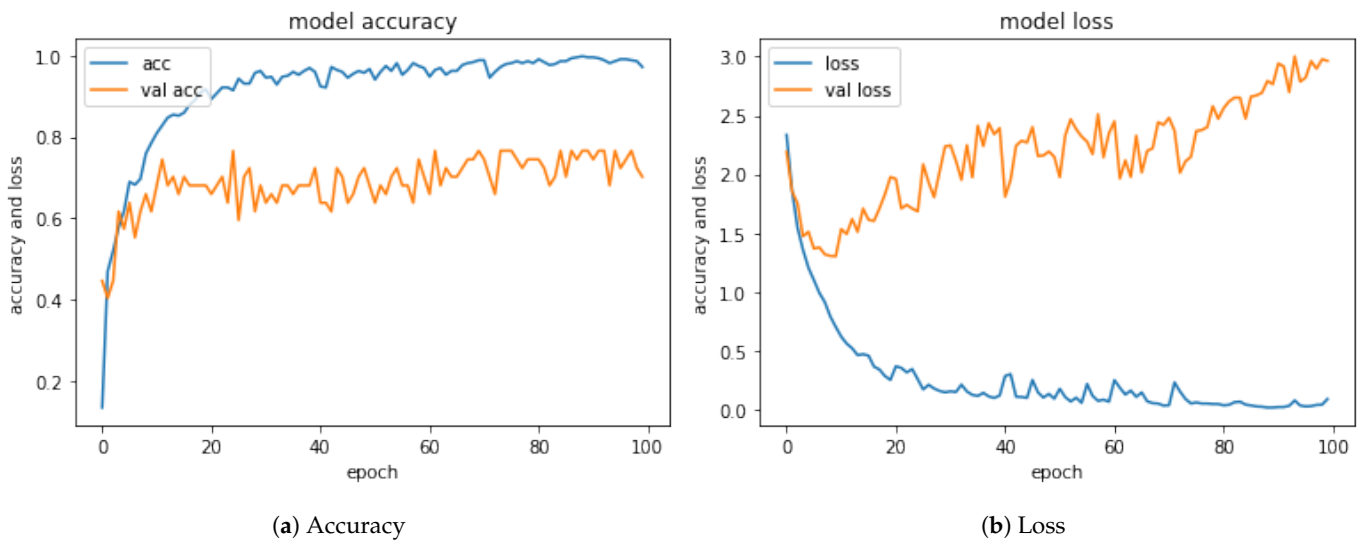


Figure A16. Training model accuracy vs. loss for BERT-CNN for category.

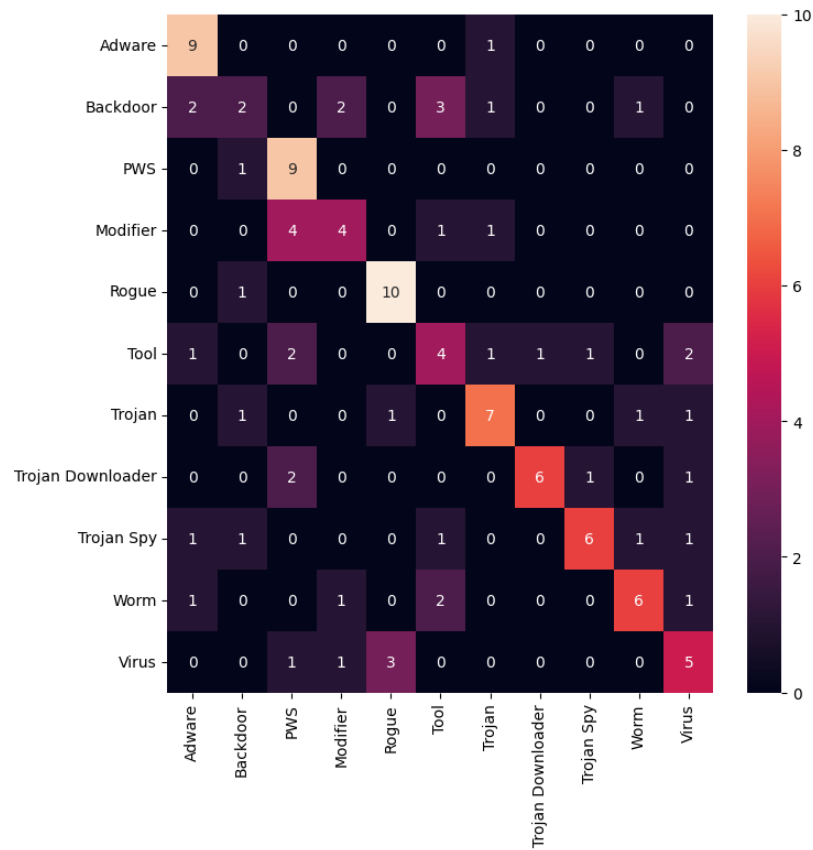


Figure A17. Confusion matrix for HMM2Vec-SVM category classification with other symbol (41 calls).

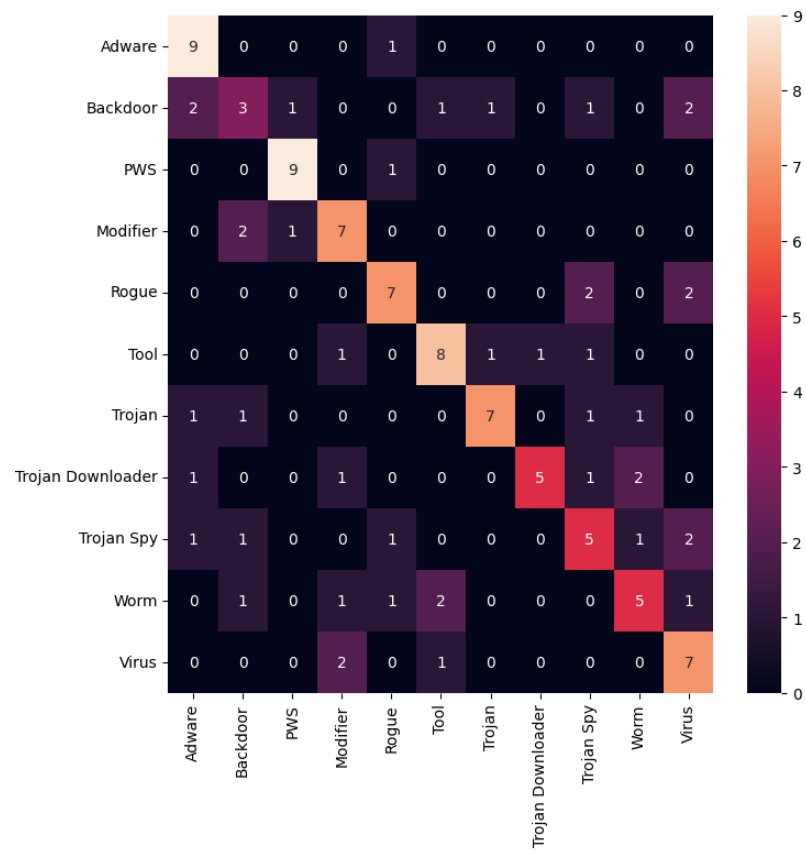


Figure A18. Confusion matrix for HMM2Vec-KNN category classification with other symbol (41 calls).

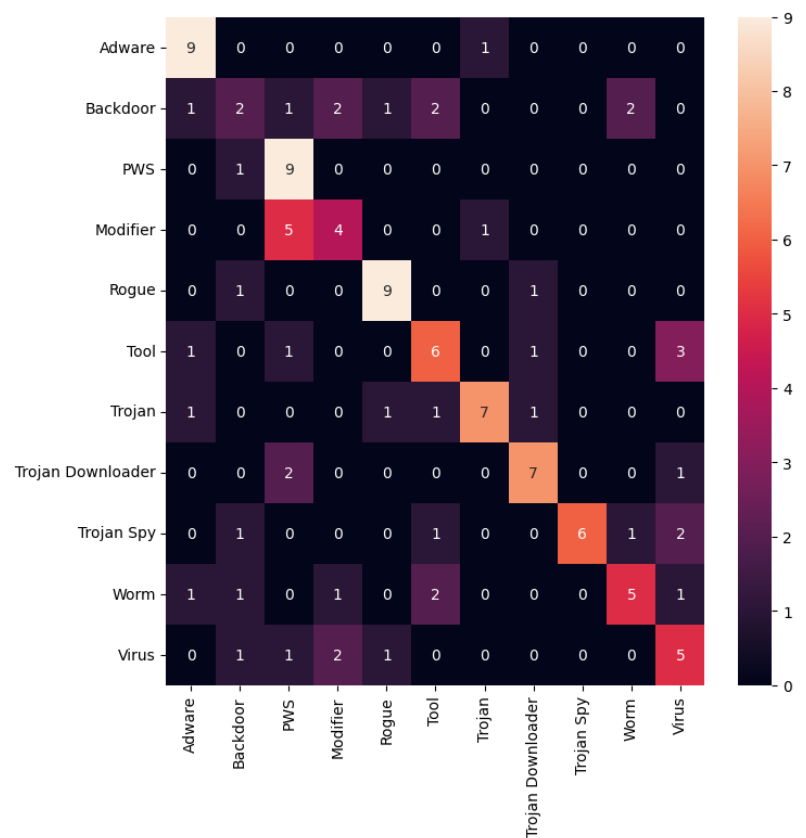


Figure A19. Confusion matrix for HMM2Vec-CNN category classification with other symbol (41 calls).

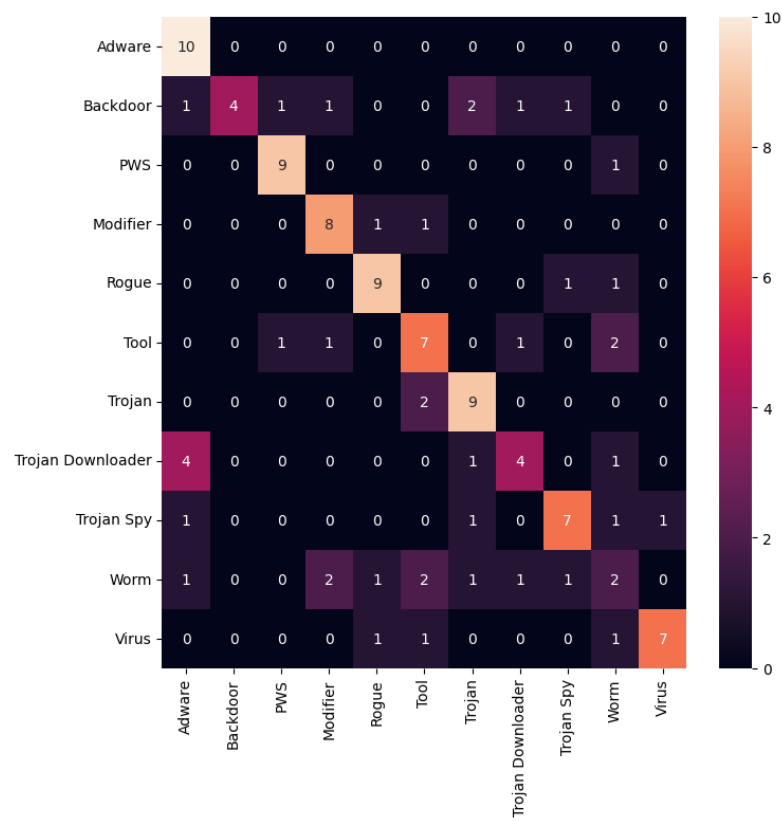


Figure A20. Confusion matrix for HMM2Vec-KNN category classification with calls in all categories.

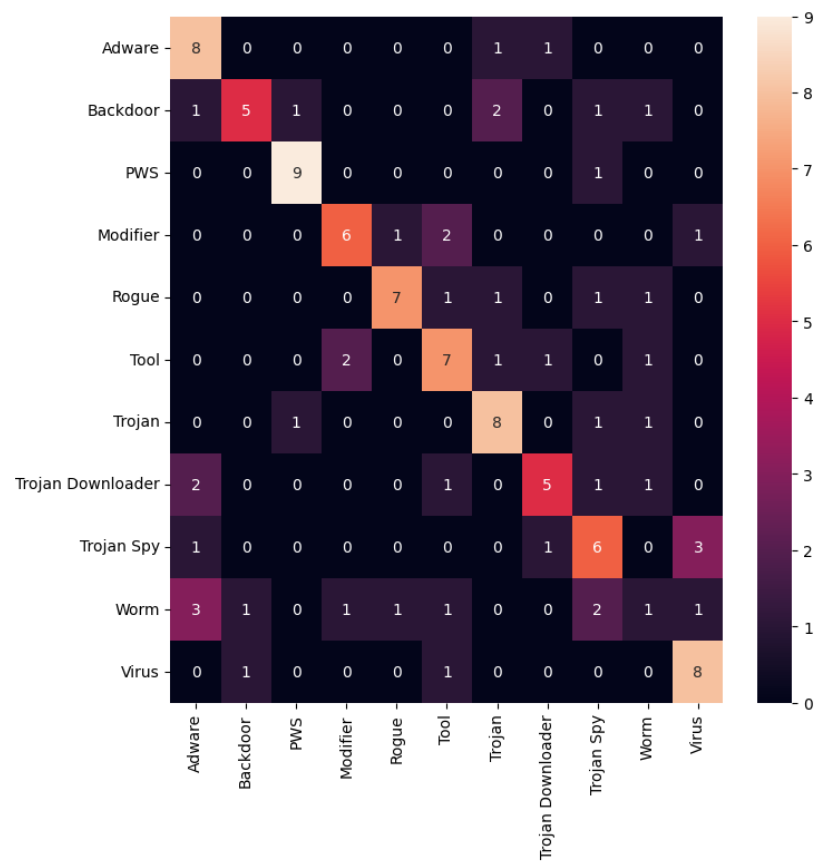


Figure A21. Confusion matrix for HMM2Vec-CNN category classification with calls in all categories.

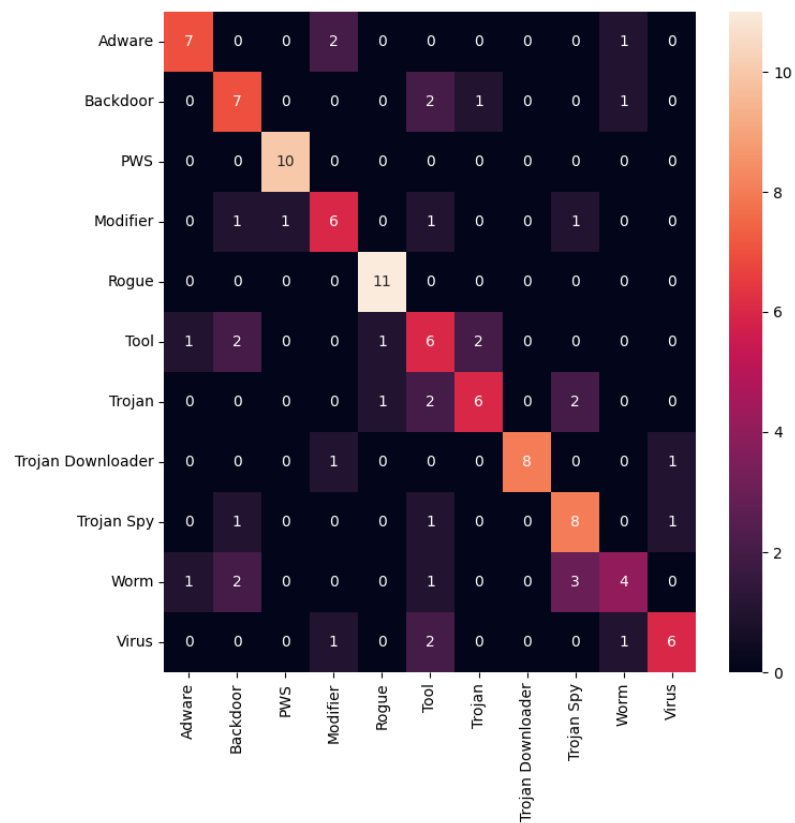


Figure A22. Confusion matrix for Word2Vec-SVM category classification with calls in all categories.

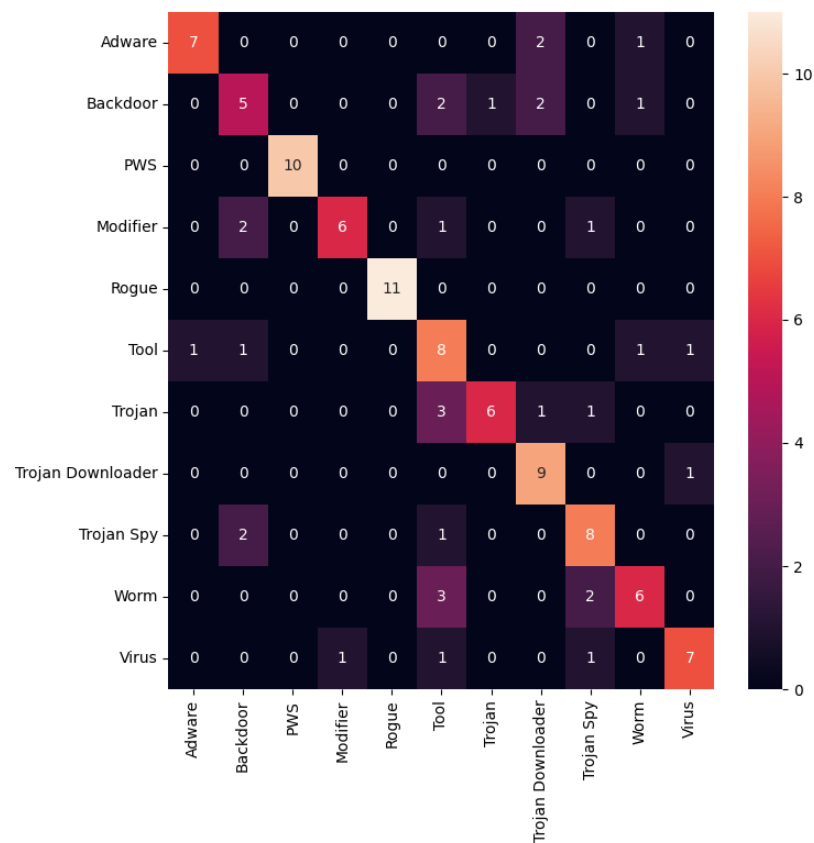


Figure A23. Confusion matrix for Word2Vec-RF category classification with calls in all categories.

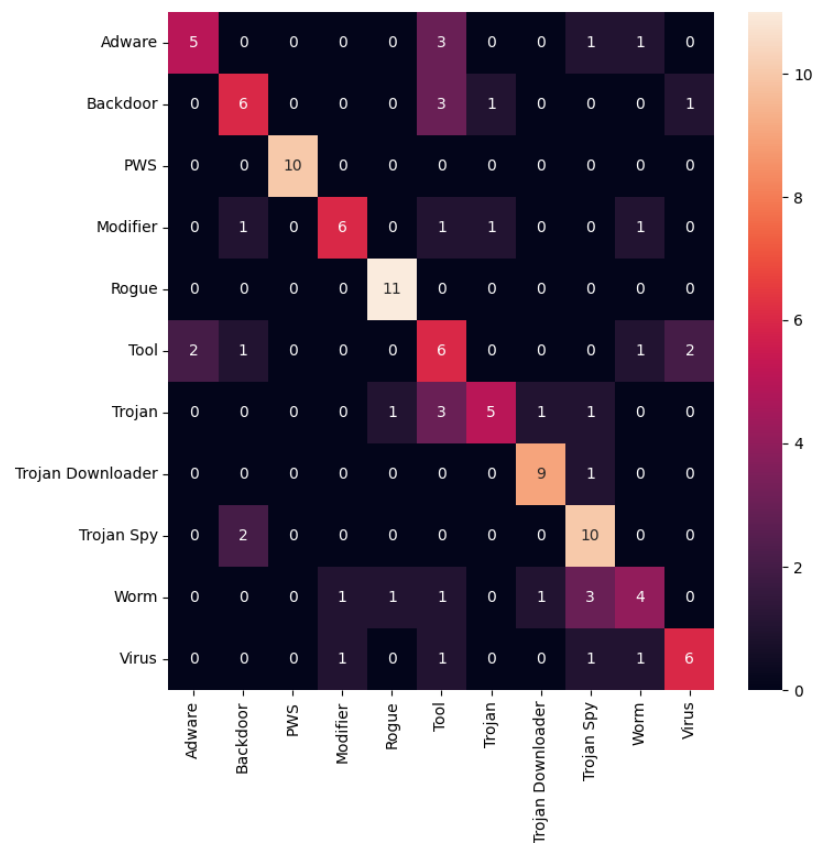


Figure A24. Confusion matrix for Word2Vec-CNN category classification with calls in all categories.

## References

1. Sonicwall. *SonicWall Cyber Threat Report*; Sonicwall: Milpitas, CA, USA, 2023.
2. Srivastava, A.; Lanzi, A.; Giffin, J. System call API obfuscation. In Proceedings of the Recent Advances in Intrusion Detection: 11th International Symposium, RAID 2008, Cambridge, MA, USA, 15–17 September 2008; Proceedings 11; Springer: Berlin/Heidelberg, Germany, 2008; pp. 421–422.
3. Kotov, V.; Wojnowicz, M. Towards generic deobfuscation of windows API calls. *arXiv* **2018**, arXiv:1802.04466.
4. Ali, M.; Hamid, M.; Jasser, J.; Lerman, J.; Shetty, S.; Di Troia, F. Profile Hidden Markov Model Malware Detection and API Call Obfuscation. In Proceedings of the ICISSP, Online, 9–11 February 2022; pp. 688–695.
5. Egele, M.; Scholte, T.; Kirda, E.; Kruegel, C. A Survey on Automated Dynamic Malware-Analysis Techniques and Tools. *ACM Comput. Surv.* **2008**, *44*, 1–42. [\[CrossRef\]](#)
6. Uppal, D.; Sinha, R.; Mehra, V.; Jain, V. Exploring Behavioral Aspects of API Calls for Malware Identification and Categorization. In Proceedings of the 2014 International Conference on Computational Intelligence and Communication Networks, Bhopal, India, 14–16 November 2014; pp. 824–828. [\[CrossRef\]](#)
7. Namita; Prachi; Sharma, P. Windows Malware Detection using Machine Learning and TF-IDF Enriched API Calls Information. In Proceedings of the 2022 Second International Conference on Computer Science, Engineering and Applications (ICCSEA), Gunupur, India, 8 September 2022; pp. 1–6. [\[CrossRef\]](#)
8. Kalash, M.; Rochan, M.; Mohammed, N.; Bruce, N.D.B.; Wang, Y.; Iqbal, F. Malware Classification with Deep Convolutional Neural Networks. In Proceedings of the 2018 9th IFIP International Conference on New Technologies, Mobility and Security (NTMS), Paris, France, 26–28 February 2018; pp. 1–5. [\[CrossRef\]](#)
9. Siddiqui, M.; Wang, M.C.; Lee, J. A Survey of Data Mining Techniques for Malware Detection Using File Features. In Proceedings of the 46th Annual Southeast Regional Conference on XX, New York, NY, USA, 28–29 March 2008; pp. 509–510. [\[CrossRef\]](#)
10. Tran, T.K.; Sato, H. NLP-based approaches for malware classification from API sequences. In Proceedings of the 2017 21st Asia Pacific Symposium on Intelligent and Evolutionary Systems (IES), Hanoi, Vietnam, 15–17 November 2017; pp. 101–105. [\[CrossRef\]](#)
11. Mikolov, T.; Chen, K.; Corrado, G.; Dean, J. Efficient Estimation of Word Representations in Vector Space. *arXiv* **2013**, arXiv:1301.3781.
12. Popov, I. Malware detection using machine learning based on word2vec embeddings of machine code instructions. In Proceedings of the 2017 Siberian Symposium on Data Science and Engineering (SSDSE), Novosibirsk, Russia, 12–13 April 2017; pp. 1–4. [\[CrossRef\]](#)

13. Alqurashi, S.; Batarfi, O. A comparison between API call sequences and opcode sequences as reflectors of malware behavior. In Proceedings of the 2017 12th International Conference for Internet Technology and Secured Transactions (ICITST), Cambridge, UK, 11–14 December 2017; pp. 105–110. [[CrossRef](#)]
14. 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.* **2015**, *13*, 1–12. [[CrossRef](#)]
15. Chandak, A.; Lee, W.; Stamp, M. A Comparison of Word2Vec, HMM2Vec, and PCA2Vec for Malware Classification. *arXiv* **2021**, arXiv:2103.05763.
16. Peters, M.E.; Neumann, M.; Iyyer, M.; Gardner, M.; Clark, C.; Lee, K.; Zettlemoyer, L. Deep contextualized word representations. *arXiv* **2018**, arXiv:1802.05365.
17. Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A.N.; Kaiser, L.; Polosukhin, I. Attention Is All You Need. *arXiv* **2017**, arXiv:1706.03762.
18. Devlin, J.; Chang, M.W.; Lee, K.; Toutanova, K. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. *arXiv* **2019**, arXiv:1810.04805.
19. Kale, A.; Pandya, V.; Di Troia, F.; Stamp, M. Malware classification with Word2Vec, HMM2Vec, BERT, and ELMo. *J. Comput. Virol. Hacking Tech.* **2022**, *19*, 1–16. [[CrossRef](#)]
20. Buster Sandbox Analyzer. 2021. Available online: <https://bsa.isoftware.nl/> (accessed on 25 June 2024).
21. Sandboxie. 2023. Available online: <https://sandboxie-plus.com/downloads/> (accessed on 25 June 2024).
22. Stamp, M. *Introduction to Machine Learning with Applications in Information Security*; CRC Press: Boca Raton, FL, USA, 2022.
23. Kale, A.S.; Di Troia, F.; Stamp, M. Malware classification with word embedding features. *arXiv* **2021**, arXiv:2103.02711.
24. Noble, W.S. What is a support vector machine? *Nat. Biotechnol.* **2006**, *24*, 1565–1567. [[CrossRef](#)] [[PubMed](#)]
25. Russell, S.; Norvig, P. *Artificial Intelligence: A Modern Approach*; Pearson: London, UK, 2016.
26. Stamp, M. A Reassuring Introduction to Support Vector Machines. In *Introduction to Machine Learning with Applications in Information Security*; Chapman and Hall/CRC: Boca Raton, FL, USA, 2017; pp. 95–132. [[CrossRef](#)]
27. Breiman, L. Random Forests. *Mach. Learn.* **2001**, *45*, 5–32. [[CrossRef](#)]
28. Liaw, A.; Wiener, M. Classification and Regression by RandomForest. *Forest* **2001**, *23*, 18–22.
29. O’Shea, K.; Nash, R. An Introduction to Convolutional Neural Networks. *arXiv* **2015**, arXiv:1511.08458.
30. Sharma, S.; Krishna, C.R.; Sahay, S.K. Detection of Advanced Malware by Machine Learning Techniques. *arXiv* **2019**, arXiv:1903.02966.

**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.