Malware API Call-Based Multiclass-Classification Using Machine Learning and Deep Learning

Sarah Adair*, Cihan Varol†, Fan Liang‡ and Vung Pham§
Department of Computer Science, Sam Houston State University
Huntsville, Texas, United States
Emails: {*sadair, †cxv007, ‡fxl027, §vung.pham}@shsu.edu, §Corresponding author

2024 12th International Symposium on Digital Forensics and Security (ISDFS) | 979-8-3503-3036-6/24/$31.00 ©2024 IEEE | DOI: 10.1109/ISDFS60797.2024.10527239

***Abstract*—Malicious attacks have been on the rise with the growth of technological innovations. With the increase in malicious attacks, many current defense systems focus on preventive and reactive measures. However, the Intrusion prevention system and Intrusion Detection system may not be able to detect all suspicious processes and files that come through the network. Even if the security system detects suspicious activity, it may be unable to quickly classify the type of malware to implement effective and efficient protection. Thus, much research has focused on creating machine-learning models to classify suspicious programs as malicious or benign. However, the limitation of classifying a suspicious file as benign or malicious still does not aid the system in deploying effective security measures to counteract the damage of executed suspicious files. We proposed to train machine learning models and deep learning models to classify the eight most common malware classes. Trained models that can classify malware classes will boost a system's efficiency in identifying the type of malware, focusing on the affected area, and deploying effective countermeasures to minimize damages. Our experiment shows that the XGBoost model performs the best for tabular data types with an average accuracy of 65%, and the Transformer model achieves the highest accuracy score of 57% for sequential datasets.**

***Index Terms*—malware classification, machine learning, deep learning**

I. Introduction

The growth of technological innovation also saw the rise of malicious attacks. Adversaries constantly seek to find vulnerabilities within organizations' attack surfaces to gain unauthorized access to cause damage or steal sensitive information. According to SonicWall's Cyber Threat Report, 10.4 million malware attacks were reported in 2022. Moreover, the first half of 2022 shows an increase of 45 percent in new malware variants [1]. With the rise in malicious attacks, many current defense systems focus on preventive and reactive measures. There are intrusion prevention systems (IPS) designed to block unknown connections that appear malicious. However, an IPS may not be able to detect all possible malicious connections as some malware pretends to be legitimate packets (trojan) or attach itself to a legitimate packet (dropper, virus, worm, e.g.) in the form of attachments or links.

When a malicious program reaches its targeted destination, the targeted host system may not be aware of the infection before it causes significant damage. Some malware, such as Spyware, droppers, and downloaders, are intended to be downloaded and acted in the background without the user's awareness. With such malware on the target host system, damage to the system occurs before users can take proper procedures to contain the damages.

Analysts can analyze a process' behavior statically and dynamically to determine whether a process is behaving unconventionally. Static analysis method involves the examination of the malware's binary code and dynamic analysis engages in the execution of the suspected file within a controlled virtual environment to observe its behavior and functionalities. One common approach for gathering data for dynamic analysis is capturing the suspicious program's Application Programming Interface (API) call sequence.

Machine learning and deep learning models can analyze the captured API call sequences and classify the suspected program. Models trained to recognize patterns of API sequence calls with the associated malware class will reduce the response time in identifying the malicious process and ensure the effective execution of appropriate countermeasures to minimize damages to victim computers. Much malware classification research involves creating multiple models designed to conduct a binary classification of benign versus malicious. Then, various models are combined to detect multi-malware classes. However, these models can only classify whether the sample is malicious or benign. Without the ability to detect the type of malware, organizations cannot implement effective defense techniques to contain and minimize the damages. There is limited research in which researchers attempt to create models to distinguish classes of malware through analysis of input samples. Our contribution to this area of research is to design a combination of machine learning and deep learning multiclass classification models in classifying eight major malware classes. Our models will be compatible with tabular data and sequential data.

II. Related Work

A. Image Conversion

In their research, author Catak et al. [2] converted each malware sample into images and passed it into a convolutional neural network (CNN) model. Each sample was split into bytes, and each byte was converted to decimal representation to calculate the byte representation's entropy value. The authors then experimented with Gaussian, Poisson, and Laplace noises to enhance the input features. From their experiment, the Laplace/gaussian noise combination with a 0.01 noise ratio obtained the best accuracy of 100%.

Concomitantly, J. Luo et al. [3] research aimed to generate a behavior graph of each malware sample according to their respective API call sequence and API call set. The researcher used an algorithm to combine time (API call time sequences) and spatial information, such as adjacency and connectivity, to generate a behavior graph representing the malware's attack behavior. The research used a BiLSTM (Bidirectional Long Short-Term Memory) network and an attention mechanism to process the data and test its accuracy. The experiment revealed that the model achieved an accuracy score of 0.5249.

Similarly, F. Amer et al. (a multi-perspective malware detection) created generic behavior to classify the sample as malicious versus non-malicious. The authors extracted API-bigram features from the calling sequences and encapsulated them into clusters. Then, the author used MLIE to generate the transition probability to characterize the sequence of formulated transitions. The researchers trained the models on the dataset published by Catak et al. [2] and evaluated them on another Window-based dataset. Although the author did not test the accuracy of the dataset, it shows it is beneficial for training models, as the authors achieved an average accuracy of 98%.

Though this category of technique is common, our dataset is the sequence of API calls, and we do not explore this direction.

B. Binary Models Classification

Most research on the dataset explored in this work adopted the approach of creating multiple binary models to classify each malware class. K.B. Sundharakumar et al. [4] compared the results of seven different malware classes using Decision Tree, kNN (k-nearest neighbors), SVM (support vector machine), LSTM (long short-term memory), CNN-LSTM, and two Conv1D (convolutional 1D) models, in which Virus classification has the highest accuracy prediction, and the kNN and SVM model achieved the best score. However, in A.Y. Daeef et al.'s [5] research, they found that RF (random forest) has provided the best classification in comparison to kNN and SVM models.

In C. Avci et al.'s [6] experiment, the researchers created variations of binary LSTM models and compared their performance using the Mal-API 2019 dataset [7]. Namely, the authors compared Vanilla LSTM, stacked LSTM, Bidirectional LSTM, and CNN-LSTM models. According to their experiment results, the Vanilla LSTM binary model performed the best, with an accuracy score of 92.8%.

W.R. Aditya et al. [8] similarly created binary classification models and used 342 sequences as input for predictions. Their experiment revealed that LSTM with RMSProp (Root Mean Squared Propagation) optimizer achieved the best score of 97.3%. Similarly, F.O. Catak [7] experimented with binary classification with a 2-layer LSTM model for each malware class. Their model achieves a 90% accuracy score. B. Panda et al. [9] utilized eight 1D-CNN models trained as binary classifiers. Then, all trained models are combined using Ensemble techniques to classify individual types and use the modified soft-voting algorithm to determine the best-predicted class. The authors' experiment obtained a 90% accuracy score for all malware classes. Y. Belkhouche's [10] trained several multi-class neural network models that use features extracted from a convolutional auto-encoder and then establish a decision-level fusion to combine all models into a final decision. The combined accuracy was 66.8%.

These research papers utilized many techniques that our research has experimented with. An overview of these papers and their experiment results helps us understand how well other models perform using the same dataset. However, our research goal was to determine the best model to utilize for multiclass classification; therefore, we did not create multiple models for binary classification.

C. Multiclass Classification Models

J. Jackson and K. Ghosh [11] created four subsets of datasets that extracted two, three, four, and five families of malware samples from the original Catak dataset. The authors used the Cosine Louvain (CL), Jaccard Louvain (JL), Cosine Girvan-Newman (CGN), and Jaccard Girvan-Newman (JGN) methods to determine the API call sequences' similarity measures within each malware family. The research compared several unsupervised models' accuracy - Naïve Bayes, Decision Trees, and Random Forests - with each subset of the dataset. The Random Forest model achieved the highest accuracy score of 73.3% on the five-malware family subset dataset. Similarly, Aditya et al. [8] also created an LSTM multiclass classification model and GRU (Gated Recurrent Unit) architecture to analyze 342 API sequences. Their experiment found that the LSTM model with Adam optimizer achieved the best score of 56.05%.

Our research goal was to determine a model that can achieve the highest score of multiclass classification. In our experiment, we utilized some of the techniques used by the above papers, namely Random Forest and LSTM, to experiment with various dataset types that we have derived from the original dataset. More information is in Section III.

III. Datasets and Pre-processing

A. Original Dataset

Our research used Catak's [12] tabular Windows malware Dataset with PE API Calls. The dataset was obtained from Kaggle, and it is open source. The dataset consists of 342 unique API calls. The authors analyzed each sample's dynamic behavior in the Cuckoo sandbox and recorded the API sequences each malware sample performs. The authors only included malware samples that called at least ten API calls. In total, the dataset contains 7,107 samples. The dataset consists of eight malware classes: Trojan (1001 samples), Backdoor (1001 samples), Downloader (1001 samples), Worms (1001 samples), Virus (1001 samples), Dropper (891 samples), Spyware (832 samples), and Adware (379 samples). The Label/Malware Class was of string values in the original dataset. All samples within the dataset belong to one type of the mentioned malware class, and there are no benign samples.

B. Frequency Count

As previously mentioned, all malware samples contained at least ten API sequence calls. Most of the samples consist of repetitive API calls within the sequence series. As such, samples range from 10 to 1,764,421 API calls. We used a Python program to analyze the original dataset and counted the total frequency of each API call for each malware sample called during dynamic analysis (*RegOutput.csv*). After analysis, we found 61 API calls were not used by any of the malware samples.

A total of 342 unique API calls were identified during the malware's dynamic analysis, and the author may not include malware samples that could have used the 61 API calls within the dataset, which explains the non-usage of these API calls. We derive two new datasets from the original frequency count dataset, one without features with non-usage (*output-nonzero.csv*) and one without features with less than ten frequency records (*output-nonsingle.csv*). The main reason is that the feature with zero frequency count does not have value in training and evaluating our models, and the feature with less than ten counts did not have much weight in classifying the malware class.

C. Normalization

After obtaining the frequency count of the 342 API calls, we applied two normalization techniques for preprocessing the dataset. The first normalization technique involved summing up the total number of API calls the specific sample calls (e.g. sample 1 called a total of 541 API calls). Then, we divide the frequency count of each feature (e.g. feature 1 have a total of 28 calls) by the total number of API calls. This normalization technique aimed to find correlations for these features. We created a *Normalization1.csv* dataset containing the computation results.

The second normalization technique involved finding the maximum value of each API call feature (e.g. feature 2 has a max value of 65536). Then, each sample's corresponding value (e.g. sample 1's feature 2) was divided by the max value. This technique aims to identify the relative relationship among each class of malware. Similarly, a new dataset*Normalization2.csv* containing the result was created.

D. Term Frequency Inverse Document Frequency

Term Frequency Inverse Document Frequency (TF-IDF) [13] values determine the weighted value of each word within a document. Term frequency is the number of specific terms in a document, represented as $tf(t, d)$. Inverse Document Frequency calculates the relevancy of a specific word in documents, represented by $idf(t, D)$. The TF-IDF values are calculated by multiplying the two values of TF and IDF, and the terms with higher scores are more significant. For our experiment, we defined the term (t) as the specific API call (feature) and the document (d) as the malware sample.

Equation 1 shows the formula for computing the term frequency for a given term t in a document d. It is defined as the total number of times the term t appears in document d divided by the total number of terms in the document. Concomitantly, there are several variants for computing inverse term frequency. However, Equation 2 is the base formula for computing inverse document frequency for a term t in a corpus D. Specifically, it is defined as the $\log$ of the number of documents in the corpus divided by the number of documents in which the term t appears.

$$tf(t, d) = \frac{f_{t,d}}{\sum_{t' \in d} f_{t',d}} \tag{1}$$

$$idf(t, D) = \log(\frac{|D|}{|d \in D : t \in d|}) \tag{2}$$

After computing the TF-IDF values for all 342 unique API data, we further refined and created new datasets by removing features based on specific criteria established through analyzing the TF-IDF values.

- *TF-IDF1 dataset*: removed all features with TF-IDF value of zero (e.g. Feature 1) and values with less than five decimal place values (e.g. Feature 1 with value of 0.000001485). After refinement, the dataset has 224 features.
- *TF-IDF2 dataset*: removed all features with TF-IDF values less than six-digit place values (e.g. Feature 10 with value of 0.000035923). The dataset has 168 features.
- *TF-IDF3 dataset*: removed all features with a six-digit place value or less (e.g. Feature 38 with value of 0.000152858). The dataset has 115 features.

After refining the frequency count datasets with the above feature extraction, we applied the same technique with our normalization dataset, in which we removed the same features as with the frequency count dataset. After manipulation, we created six new datasets: *TF-IDF1NOM*, *TF-IDF2NOM*, *TF-IDF3NOM*, *TF-IDF1NOM2*, *TF-IDF2NOM2*, and *TF-IDF3NOM2*.

E. Sequential

The Mal-API dataset contains a series of sequential API calls that were observed during the execution of the samples, which may help experiment with sequential models such as 1D CNN and LSTM deep learning models. By inputting a series of sequential API calls, deep learning models can analyze the pattern of the API calls and predict appropriate malware classes while using the information brought by the sequence of API calls.

We used a Python program to extracted the "X" number of sequential calls and created multiple datasets from the raw data. The program processed the raw data, selected the samples that met the extraction requirements, and skipped samples that did not meet the requirements. We extracted the first, middle, and last "X" number of sequential calls and created the following datasets:

- First, Middle, Last – 25 features (6,335 samples)
- First, Middle, Last – 30 features (6,163 samples)
- First, Middle, Last – 35 features (6,021 samples)
- First, Middle, Last – 40 features (5,916 samples)

- First, Middle, Last – 45 features (5,827 samples)

We use one-hot encoding to encode the API call sequences and the label values. Therefore, these values are represented individually instead of scaling the value between 0 and 1.

IV. Experiments

This research aims to create machine learning models that can perform multiclass classification. We created combinations of machine learning and deep learning models. Then, we experimented with the various datasets we have created from the raw data. Specifically, we have experimented with Random Forests (RF), Extreme Gradient Boosting (XGBoost), Support Vector Machine (SVM), Multilayer Perceptron (MLP), 1D CNN, Long Short-Term Memory (LSTM) (single-layer, 2-layer, and Bidirectional), and Transformer. We also attempted to experiment with the Partial Least Squares Regression (PLSR) model because it often works well with regression problems with many input features [14] as in this case. However, PLSR did not obtain satisfactory results; therefore, we dropped the model from the research.

A. Random Forest

Random Forest (RF) [15] constructs multiple decision trees to analyze the input data during the training phase and select the best results as the output. As this technique is known for achieving a good accurate classification score for tabular datasets, we selected this model to experiment with our tabular dataset. For our model, we manipulated the n-estimators and random state values to determine which pair performs the best in classification. Furthermore, we included cross-validation scores and the *K-fold* technique to train the models on subsets of datasets and evaluate them on other datasets. Moreover, the Cross-Validation technique helps prevent overfitting of the model training.

B. Extreme Gradient Boosting

The eXtreme Gradient Boosting (XGBoost) technique [15] is part of the gradient boosting family of implementations and is commonly used for classification and regression tasks. XGBoost model builds a series of decision trees, each designed to correct the mistakes of the precedent decision trees. In theory, the model will improve throughout the training phase and provide a better accuracy score. In our XGBoost Classifier, we manipulated the seed value and left other parameters as default. Like the RF model, we implemented a Cross-Validation technique to train and evaluate the model on a subset of datasets.

C. Support Vector Machine

Support Vector Machine (SVM) [15] is another supervised machine learning model used for classification and regression tasks. The SVM model aims to find the best hyperplane that can best place data points (support vector) and determine the class these data points belong to. SVM models are well suited for classification, which fits our tabular dataset and purposes.

In our context, the data point for the frequency count and normalization dataset will be the values of the frequency count and the normalization score of the malware sample's API calls. For sequential datasets, the data points are the various sequential values extracted from the original dataset. The SVM model can classify the different malware classes based on pattern analysis. The following variable was tested within the param-grid, *C = 10, 100*, *Kernel = rbf*, and *gamma = 0.1, 1*. Furthermore, cross-validation parameters were also included when running the param-grid variable.

D. Multilayer Perceptron

Multilayer Perceptron (MLP) is a form of Artificial Neural Network (ANN) consists of multiple fully connected layers that consist of neurons. The basic architecture of an MLP model is an input layer, one or more hidden layers, and an output layer. The input layer takes in the features and processes them through the established set of neurons. Then, the data are passed and processed in the hidden layers. Once the data has been trained within these hidden layers, the model will produce an output through the output layer. Early stopping techniques were also experimented with to prevent overfitting of the model.

For our MLP model, we experiment with the following hyperparameter to get the best result: number of nodes, epochs, batch size, dropout rate, and learning rate. Regarding the architecture, after several experiments, the best model of this type has two blocks of dense layers with one batch-normalization layer in between. The first block contains four fully connected layers, while the second contains three. The batch-normalization helps smoothen the loss function and improves the training speed of the model.

E. Convolutional Neural Network

Convolutional Neural Network (CNN) [16] is a deep learning technique designed to process image data and predict classification outcomes. Because our dataset has univariate features, we used an alternative version of CNN, 1D CNN. The 1D CNN model best fits sequential data; the model captures the sequential data pattern to make the classification prediction.

Similar to the MLP model, early stopping techniques were also experimented with to prevent overfitting of the model. For our 1D CNN model, we experiment with the following hyperparameter to get the best result: number of nodes, epoch size, dropout rate, and learning rate. After several experiments, we found the following architecture achieves the best result: alternating three 1D CNN layers and max pooling layers, followed by a flattened layer and a dense layer as the output.

F. Long Short-Term Memory

The LSTM model [17] is a subcategory of recurrent neural networks (RNNs) designed to work with and learn from data sequences. Our research experiments with two variations of LSTM models: single-layer LSTM and two-layer LSTM. Experimenting with different variations aims to determine which models work best for the classification of malware.

Similarly, we utilized early stopping techniques to prevent overfitting of the training model.

We experiment with the following hyperparameters to get the best result: number of nodes, epoch size, dropout rate, and learning rate. The best LSTM architecture consists of either one LSTM or two LSTM layers, respective to the specified models, and two blocks of dense layers separated by a flattened layer.

G. BiDirectional LSTM

The BiDirectional LSTM model [17] consists of two LSTMs - in which the first takes the input features in a forward direction, and the second takes the input in a backward order. The BiDirectional LSTM model captures the dataset's more complex patterns. Because the BiDirection LSTM model is designed for sequential data, we only train and evaluate the model with sequential datasets. The best architecture for our BiDirection LSTM model consists of a bidirectional layer, two dense layers, a batch normalization layer, a flattened layer, a dropout layer, and three dense layers.

H. Transformer

We experimented with a Transformer deep learning model [18] in processing sequential datasets. The transformer model handles sequential data through the use of a self-attention mechanism. Each API call in our dataset is represented as a vector and processed through the input embedding. Then, the self-attention mechanism weights each word and determines its importance score relative to other words. The weighted words are passed through the neural network consisting of fully connected nodes.

For the same reason as BiDirectional LSTM, we only experiment with sequential datasets for our Transformer model. Specifically, we implemented a learning schedule in which we increased the exponential decay of the learning rate throughout the model training to find the best learning rate for the model. After experimenting, the best architecture for the Transformer model consists of an embedding layer, a transformer-block layer, a global average pooling (1D) layer, two dense layers, and a flattened layer, followed by three dense layers. The global average pooling layer is one-dimensional because our data are univariate features.

V. Results and Discussions

This work utilizes commonly used evaluation metrics for classification problems: precision, recall, F1-score, and accuracy as evaluation metrics. However, the dataset is relatively balanced (described in Section III), and the measured scores for precision and recall are relatively similar. Therefore, this section only reports the accuracy scores, as shown in Table I.

A. Random Forest

Based on our experiment, the set of hyperparameters produces the best accuracy, including *n-estimator = 20*, *random-state = 40*, and *kFold =15*. For the frequency count data, the *TF-IDF1* dataset with 254 features performs the best, with an accuracy score of 0.6512. For the *normalization1* data, the *TF-IDF3* dataset with 115 features achieves the highest score of 0.6231. The *TF-IDF1* dataset performed the best when applying the Normalization preprocessing technique, which achieved a score of 0.6512. For our sequential data, the dataset of the last 30 API calls of each malware sample obtained the best score of 0.5069. As Table I shows, the Sequential data type achieved the lowest score, whereas the frequency count and *normalization 2* data type performed with the same accuracy score with frequency count.

B. Extreme Gradient Boosting

For our XGBoost model, the parameters include *Objective = multi: softmax*, *seed = 55*, and *kFold = 15* have performed the best. For the frequency count data, the *TF-IDF1* dataset performs the best, with an accuracy score of 0.6666. For the *normalization1* data, the *TF-IDF2* dataset with 168 feature extraction or manipulation achieves the highest score of 0.5113. The *TF-IDF1* dataset performed the best when applying the *Normalization2* preprocessing technique, which achieved a 0.3045 accuracy score. For the sequential data, the last 45 API calls dataset obtained the best score of 0.4826. For this model, the *normalization1* data type performed the best, with the sequential data type gaining second best.

C. Support Vector Machine

The SVM's best parameter varies slightly depending on the data type with the C-variable remains constant as *C=100*. For the frequency count data type, the *TF-IDF1* dataset again achieves the best score of 0.3629 with hypermeters of *gamma = 0.1* and *CV = 5*. For the **normalization1** data type, the **TF-IDF2NOM** dataset performs the best with hypermeters of **gamma = 1** and *CV =5*, achieving an accuracy score of 0.5056. For *normalization2* datatype, the model achieved a score of 0.4030 with *gamma = 1* and *CV = 5*. With the sequential dataset, the last 45 feature dataset obtained the best score of 0.3473 with a hyperparameter combination of *gamma = 0.1*.

D. Multilayer Perceptron

Based on our experiment, the combination of 128 nodes, dropout rate 0.5, learning rate 0.09, and batch size 128 performed the best for our frequency count and *normalization1* dataset. For the *normalization2* technique, batch size 64 performed the best. For our experiment with sequential data, we tested different ranges of hypermeter combinations. Based on our experiment, a combination of 32 nodes, 0.5 dropout probability, 0.07 learning rate, and 32 batch size achieved the highest accuracy score.

E. Convolutional Neural Network

For 1D CNN, we have experimented with two models, one with one-layer 1D CNN and the second with two-layer 1D CNN. For the one-layer 1D CNN model, a hyperparameter combination of 256 nodes, dropout rate 0.5, learning rate 0.09,

TABLE I
SUMMARY OF THE RESULTS

Data Preprocessing	RF	XGBoost	SVM	MLP	CNN1	CNN2	LSTM1	LSTM2	BiLSTM	Transformer
Frequency Count	0.6512	**0.6666**	0.4677	0.4501	0.4719	0.4744	0.4233	0.1589	x	x
Normalization1	0.6231	0.6350	0.5056	0.5682	0.5893	0.5633	0.4437	0.2045	x	x
Normalization2	0.6512	**0.6666**	0.4030	0.4156	0.4480	0.4838	0.4261	0.1976	x	x
Sequential	0.5069	0.4826	0.3473	0.5132	0.5329	0.5131	0.5026	0.4820	0.5163	**0.5717**

RF: Random Forest, XGBoost: Extreme Gradient Boosting, MLP: Multilayer Perceptron, CNN1: One-Layer Convolutional Neural Network (CNN), CNN2: Two-Layer CNN, LSTM1: One-Layer Long Short-Term Memory (LSTM), LSTM2: Two-Layer LSTM, BiLSTM: Bidirectional LSTM

and batch size 64 achieved the best result for the frequency count dataset and *normalization2* datasets, achieving scores of 0.4719 and 0.4480, respectively. For the *normalization1* datatype, a combination of 128 nodes, a dropout rate of 0.5, a learning rate of 0.09, and a batch size of 128 achieve an accuracy score of 0.5893. For the sequential dataset, the Last 45 API sequence call dataset performs the best with the highest accuracy score of 0.5329, with a combination of 64 nodes, 0.5 dropout rate, 0.09 learning rate, and batch size 32.

For our two-layer 1D CNN model, the following hyperparameter performs the best. For the frequency count and *normalization2* datatype, parameter 256 nodes, dropout rate 0.5, learning rate 0.09, and bath size 64 attain the highest score of 0.4744 and 0.5633, respectively. The *normalization2* dataset, parameter 128 nodes, dropout 0.5, learning rate 0.09, and batch size 64 attain the highest score of 0.4838. For our sequential dataset, a hyperparameter combination of 64 nodes, 0.5 dropout rate, 0.09 learning rate, and batch size 32 obtained a score of 0.5131.

F. Long Short-Term Memory

For our one-layer LSTM model, our frequency count and *normalization 1* dataset achieve the best score using the same hyperparameter of 128 nodes, dropout rate 0.5, learning rate 0.09, and batch size 64, achieving an accuracy score of 0.4233 and 0.4437, respectively. For the *normalization2* data type, a hyperparameter combination of 256 nodes and batch size 128 has performed the best accuracy score of 0.4261. For this model, the sequential dataset achieved the highest accuracy score of 0.5026, with hyperparameter 256 nodes, 0.5 dropout rate, 0.09 learning rate, and 32 batch size.

Based on our experiments, the two-layer LSTM model did not perform well with frequency count and normalization data types. A possible reason for such a low score is that the LSTM model is designed for sequential data and does not work well with datasets with standardized values. This can be seen with our sequential dataset experiment, which obtained an accuracy score of 0.4820 with hyperparameter 256 nodes, 0.5 dropout rate, 0.09 learning rate, and 32 batch size.

As previously stated, the BiDirectional LSTM model works best with sequential data; therefore, our experiment only included sequential datasets. Based on our experiment, the Last 45 features of API call sequences achieved the best accuracy score of 0.5163, with a hyperparameter combination of 256 number nodes, 0.5 dropout rate, 0.09 learning rate, and 32 batch size.

G. Transformer Model

Similarly, the Transformer model expects three-dimensional data input. Therefore, we only experimented with sequential datasets. From our experiment, the Last 400 features of API call sequences achieve the highest accuracy score of 0.5717 with a hyperparameter learning rate of 0.001 and batch size of 64.

VI. CONCLUSIONS

This research aims to create machine learning and deep learning models that classify multiclass malware. We utilized the Malware Window API sequence dataset. From the original dataset, we preprocessed the data by counting the frequency count, normalizing the frequency count, and extracting a specified amount of API sequential calls. We experimented with conventional machine learning techniques, including Random Forest, Extreme Gradient Boosting, and Support Vector Machine. We also experimented with deep learning techniques, including Multilayer Perceptron, one-directional CNN, Long Short-Term Memory, and Transformer.

From our experiments, the XGBoost model achieved the highest score for the frequency count, *normalization1*, and *normalization2* data type. Unlike Random Forest's parallel decision tree, XGBoost utilizes ensemble decision trees in which sequential trees aim to correct the wrong predictions of previous trees to reduce loss and achieve higher accuracy. The Transformer model achieved the highest accuracy score for sequential datasets. The Transformer model performed the best because it utilizes positional encoding and self-attention mechanism to understand the sequential value of each inputted API call and weighted the importance of those values to capture in-depth patterns to classify each malware class. Based on these results, we can conclude that machine learning models fit best with tabular API data, while deep learning models perform best with sequential data.

The future direction of this work should be to incorporate explainable artificial components to find out how the experimented models make their decisions. Finally, our models' source code, architecture, summary, and test experiment's confusion matrix are on our GitHub repository at https://github.com/Sarah-Adair/Malware-API-Call-based-Multiclass-Classification.

REFERENCES

[1] Sam Cook, "Malware statistics and facts for 2024," https://www.comparitech.com/antivirus/malware-statistics-facts/, accessed: 2024-02-24.

[2] F. O. Catak, J. Ahmed, K. Sahinbas, and Z. H. Khand, "Data augmentation based malware detection using convolutional neural networks," *PeerJ Computer Science*, vol. 7, p. e346, Jan. 2021. [Online]. Available: https://doi.org/10.7717/peerj-cs.346

[3] J. Luo, Z. Zhang, J. Luo, P. Yang, and R. Jing, "Sequence-based malware detection using a single-bidirectional graph embedding and multi-task learning framework," *Journal of Computer Security*, no. Preprint, pp. 1–23, 2023.

[4] K. Sundharakumar, N. Bhalaji *et al.*, "Malware classification using deep learning methods," in *2023 3rd International Conference on Smart Data Intelligence (ICSMDI)*. IEEE, 2023, pp. 278–281.

[5] A. Y. Daeef, A. Al-Naji, and J. Chahl, "Features engineering for malware family classification based api call," *Computers*, vol. 11, no. 11, p. 160, 2022.

[6] C. Avci, B. Tekinerdogan, and C. Catal, "Analyzing the performance of long short-term memory architectures for malware detection models," *Concurrency and Computation: Practice and Experience*, vol. 35, no. 6, pp. 1–1, 2023.

[7] F. O. Catak, A. F. Yazı, O. Elezaj, and J. Ahmed, "Deep learning based sequential model for malware analysis using windows exe api calls," *PeerJ Computer Science*, vol. 6, p. e285, 2020.

[8] W. R. Aditya, R. B. Hadiprakoso, A. Waluyo *et al.*, "Deep learning for malware classification platform using windows api call sequence," in *2021 International Conference on Informatics, Multimedia, Cyber and Information System (ICIMCIS*. IEEE, 2021, pp. 25–29.

[9] B. Panda, S. S. Bisoyi, and S. Panigrahy, "An ensemble approach for imbalanced multiclass malware classification using 1d-cnn," *PeerJ Computer Science*, vol. 9, p. e1677, 2023.

[10] Y. Belkhouche, "Api-based features representation fusion for malware classification," in *2023 IEEE 47th Annual Computers, Software, and Applications Conference (COMPSAC)*. IEEE, 2023, pp. 1658–1662.

[11] A. J. Jackson and K. Ghosh, "Unsupervised learning approaches for construction of malware families," in *2022 IEEE International Conference on Big Data (Big Data)*. IEEE, 2022, pp. 2989–2996.

[12] Ferhat Ozgur Catak, "API Call based Malware Dataset," https://www.kaggle.com/datasets/focatak/malapi2019, accessed: 2024-02-24.

[13] G. Salton and C. Buckley, "Term-weighting approaches in automatic text retrieval," *Information processing & management*, vol. 24, no. 5, pp. 513–523, 1988.

[14] V. Pham, D. C. Weindorf, and T. Dang, "Soil profile analysis using interactive visualizations, machine learning, and deep learning," *Computers and Electronics in Agriculture*, vol. 191, p. 106539, 2021.

[15] C. LeDoux and A. Lakhotia, "Malware and machine learning," in *Intelligent Methods for Cyber Warfare*. Springer, 2014, pp. 1–42.

[16] C. Hasegawa and H. Iyatomi, "One-dimensional convolutional neural networks for Android malware detection," in *2018 IEEE 14th International Colloquium on Signal Processing & Its Applications (CSPA)*. IEEE, 2018, pp. 99–102.

[17] E. d. O. Andrade, J. Viterbo, C. N. Vasconcelos, J. Guérin, and F. C. Bernardini, "A model based on LSTM neural networks to identify five different types of malware," *Procedia Computer Science*, vol. 159, pp. 182–191, 2019.

[18] T. Lin, Y. Wang, X. Liu, and X. Qiu, "A survey of transformers," *AI Open*, 2022.