Predicting and Explaining Malware Detection

[1]Tanay Tekön, tanaytekon@hacettepe.edu.tr
[1]Burak Kurt, burak_kurt@hacettepe.edu.tr
[1,2] Tugba Gurgen Erdogan, tugba@cs.hacettepe.edu.tr
[1]Fuat Akal, akal@cs.hacettepe.edu.tr
[1] Hacettepe University, Computer Engineering, Ankara, Turkey
[2] Hacettepe University, Computer Engineering, Software Engineering Research Group

Abstract

This report explores the development of a machine learning model for accurate and interpretable malware detection. Unlike previous approaches that performed feature selection, we leverage all available features from the dataset to potentially capture a wider range of informative insights. Additionally, we integrate Explainable AI (XAI) techniques to understand the relationships between data and the model's predictions, aiming for a more transparent and trustworthy model.

The project utilizes the Microsoft Malware Prediction Competition dataset and focuses on LightGBM and Multi-Layer Perceptron (MLP) models. LightGBM achieves an AUC score of 0.7138 on the training set and 0.7125 on the test set, demonstrating good generalizability. While the MLP model exhibits overfitting tendencies, it reaches an AUC score of 0.7146 on the test set after early stopping techniques are applied.

Furthermore, SHAP (SHapley Additive exPlanations) and LIME values are employed to interpret feature importance and gain insights into the model's decision-making process. This interpretability is valuable for understanding how the model identifies malware and can guide future feature engineering efforts.

The report concludes by acknowledging limitations, such as the overfitting observed in the MLP model, and proposes avenues for future work. These include exploring feature engineering techniques, ensemble learning approaches, deeper neural network architectures, and more advanced XAI methods. By addressing these limitations and pursuing these future directions, the accuracy, robustness, and interpretability of the malware detection model can be continuously enhanced.

1. INTRODUCTION

Malware detection plays a critical role in safeguarding computer systems from malicious software attacks. These attacks can steal sensitive data, disrupt operations, and cause significant financial damage. Machine learning (ML) models offer a promising approach to automatically identifying malware, but achieving both high accuracy and interpretability remains a challenge.

This report details the development of an ML model for accurate and interpretable malware detection. We deviate from prior studies that perform feature selection by leveraging all available features in the initial models. This comprehensive approach aims to capture a broader range of potentially informative data points that might contribute to malware identification. Additionally, we integrate Explainable AI (XAI) techniques to understand the relationships between the data and the model's predictions. This focus on interpretability fosters trust and transparency in the model's decision-making process.

The project employs the Microsoft Malware Prediction Competition dataset and investigates the performance of LightGBM and Multi-Layer Perceptron (MLP) models. We evaluate the models' accuracy and generalizability, while also incorporating XAI techniques to gain insights into feature importance. The report concludes by outlining limitations and proposing avenues for future work, paving the way for continuous improvement of the malware detection model's effectiveness.

2. BACKGROUND AND RELATED WORK

Table 1: Summary of the related work

Study	Features	Classifiers	Performance	Datasets
Analysis of Malware Prediction Based on Infection Rate Using Machine Learning Techniques [2]	All Features	LGBM / Decision Tree / Neural Network	AUC Score: 0.73232/ 0.63923/ 0.70284	Microsoft Malware Prediction[1]
Microsoft Malware Prediction Using LightGBM Model [3]	Feature Selection & Feature Engineering	Catboost/ Xgboost/ Lightgbm	AUC Score: 0.638/ 0.677/ 0.684	Microsoft Malware Prediction
Predicting Malware Attacks using Machine Learning and AutoAI [4]	All Features / Feature Selection (LGBM) / Feature Selection (Random Forest)	LGBM / Naive Bayes / Logistic Regression	Accuracy: 69.88 / 53.13 / 54.00	Microsoft Malware Prediction
The Application of LightGBM in Microsoft Malware Detection [5]	Feature Selection (Chi Square)	Logistic Regression / KNN / LGBM	AUC Score: 0.569 / 0.530 / 0.720687	Microsoft Malware Prediction
TabLSTMNet: enhancing android malware classification through integrated attention and explainable AI [6]	All Features	TabLSTMNet	F1 Score: 0.9681	NATICUSdroid, TUNADROMD

In [2], researchers tried LGBM, Decision Tree and Neural Network on the same dataset and LGBM performs best among others.In [3], researchers used LightGBM to obtain feature importances and select some features based on their importance. However, this feature selection technique results worse than the first study. 3 Different experimental setup is used in [4]. Three different model (LGBM, Naive Bayes and Logistic Regression) are tried on each experimental setup. For all models, best accuracies are obtained when first setup is used (All Features). Lastly, [5] uses Chi Square test to extract feature importance. Best score is obtained on LGBM model among Logistic Regression and KNN.

Building upon the valuable insights from previous research, our project aims to make the following contributions:

• **Focus on All Features:** Unlike studies like [3] that performed feature selection based on importance, we utilize all available features in our initial models. This comprehensive approach ensures that potentially informative features are not prematurely excluded, potentially leading to a more robust model that leverages the full dataset.

• **Explainable AI Integration:** Our project goes beyond achieving high accuracy. We plan to incorporate explainable AI techniques to understand the relationships between data and the model's predictions. This interpretability will be valuable for gaining insights into how the model identifies malware, potentially leading to improved detection strategies and feature engineering in future iterations.

3. METHODOLOGY

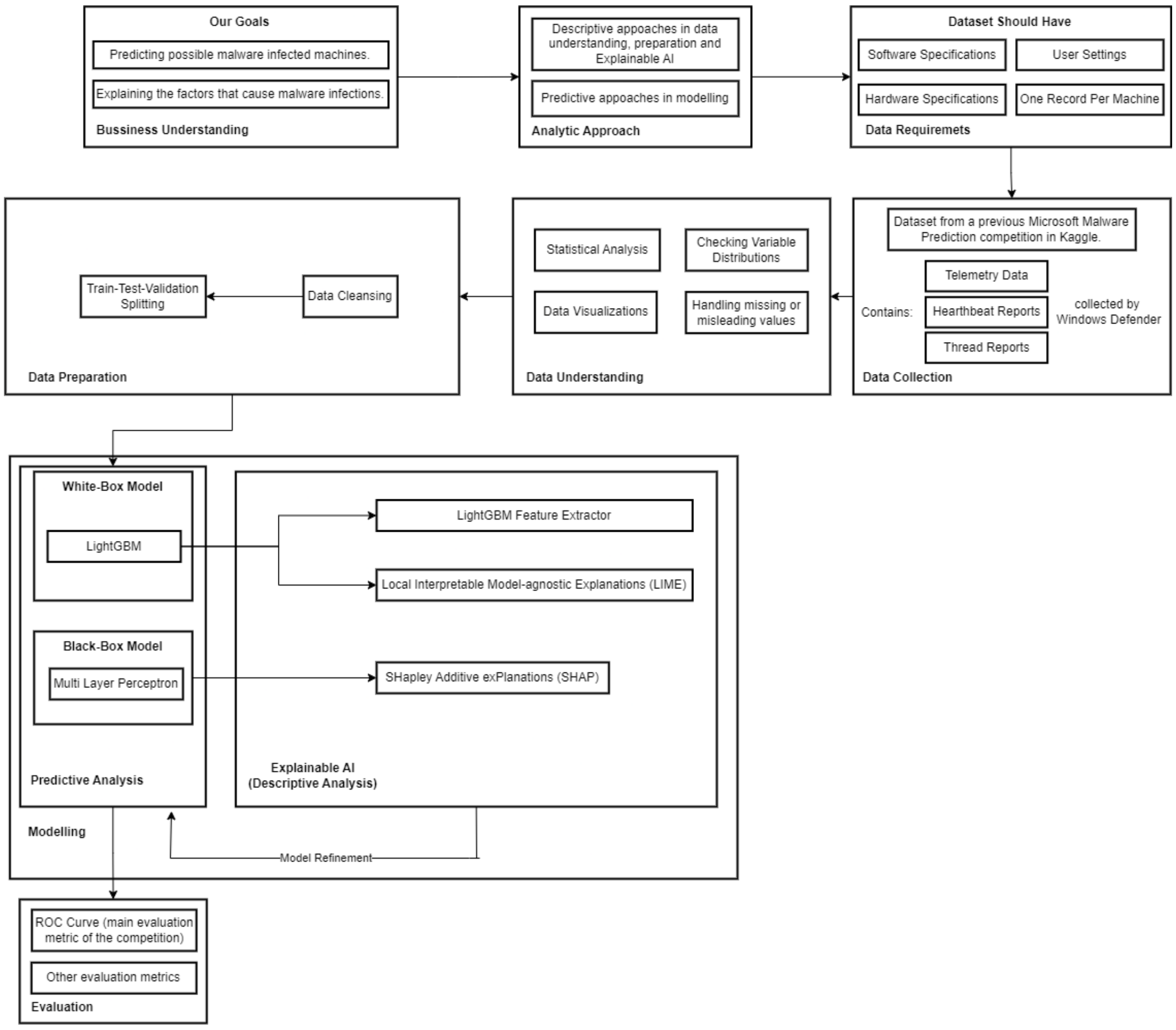

Our Goals
Predicting possible malware infected machines.
Explaining the factors that cause malware infections.
Bussiness Understanding
Descriptive appoaches in data understanding, preparation and Explainable AI
Predictive appoaches in modelling
Analytic Approach
Dataset Should Have
Software Specifications
User Settings
Hardware Specifications
One Record Per Machine
Data Requiremets
Dataset from a previous Microsoft Malware Prediction competition in Kaggle.
Contains:
Telemetry Data
Hearthbeat Reports
Thread Reports
collected by Windows Defender
Data Collection
Statistical Analysis
Checking Variable Distributions
Data Visualizations
Handling missing or misleading values
Data Understanding
Train-Test-Validation Splitting
Data Cleansing
Data Preparation
White-Box Model
LightGBM
Black-Box Model
Multi Layer Perceptron
Predictive Analysis
LightGBM Feature Extractor
Local Interpretable Model-agnostic Explanations (LIME)
SHapley Additive exPlanations (SHAP)
Explainable AI (Descriptive Analysis)
Modelling
Model Refinement
ROC Curve (main evaluation metric of the competition)
Other evaluation metrics
Evaluation

Fig 1 Methodology Diagram

1) Business Requirements:

- Improved Malware Detection: Improved Malware Detection allows businesses to proactively identify and prioritize security measures for vulnerable systems, minimizing potential damage and downtime

- Transparency and Trust: Our model needs to produce reliable results and the model building process should be as transparent as possible to the company.

- Security: We need to apply necessary measures to ensure data security and privacy, considering the potentially sensitive nature of user information.

2) **Data Requirements:**
   - The data must have relevant features for our problem.
   - A sufficient volume of data is necessary to train the model effectively.
   - The data should also be relatively complete, with minimal missing values or inconsistencies.

3) **Model Requirements:**
   Our goal in this project is to create a machine learning model that, using a dataset that has been provided, can identify whether malware is present on a computer system. The following summarizes the main choices taken at the model requirements stage:

   Feature Selection:

   - **Importance:** We considered all features in the dataset as potentially relevant for malware detection (except for the redundant and missing ones).
   - **Data Type:** Since the majority of the features are categorical, we needed a strategy to handle them effectively for machine learning algorithms.

   Feature Encoding:

   - **Categorical Encoding:** To prepare the categorical data for modeling, we employed frequency encoding or one-hot encoding depending on the specific characteristics of each column. Frequency encoding is suitable for situations where the categories have too many unique values, while one-hot encoding is preferred when categories are independent and have no inherent order.

   Model Selection:

   - **LightGBM:** Due to the presence of frequency-encoded categorical features, LightGBM was chosen as the first model. This gradient boosting decision tree algorithm is well-suited for handling encoded categorical data while offering high efficiency and accuracy.
   - **Multi-Layer Perceptrons (MLPs):** To potentially capture more complex non-linear relationships within the data, a Multi-Layer Perceptron (MLP) model was also selected. MLPs are artificial neural networks known for their ability to learn intricate patterns in data, potentially leading to improved performance in malware detection.

   These decisions ensured that our models could effectively leverage the provided data, including the significant portion of categorical features, to deliver accurate predictions for malware presence.

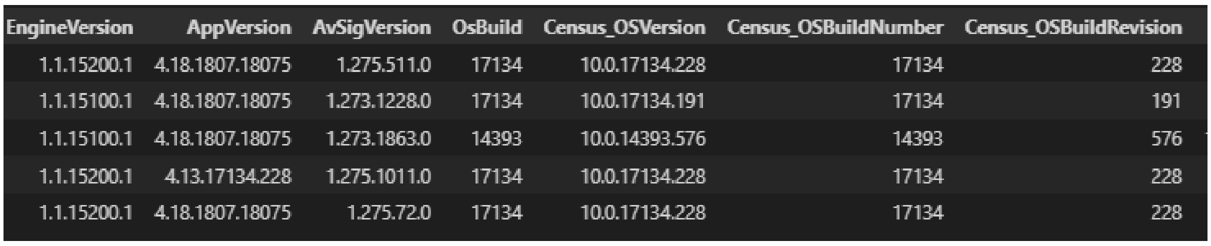
4) **Data Exploration and Understanding.:** In this project, only one open source dataset is used and it comes from Microsoft Malware Prediction Competition from Kaggle. Dataset contains 8921483 rows along with 83 columns. 1 column indicates the ‘‘MachineIdentifier’’ (unique for every row), and 1 column is the target column (‘‘HasDetections’’). When these two columns are excluded, there are 81 initial features in the dataset.

In the dataset, most of the columns are categorical and have many unique values. Even some of the columns are indicated with “uint” or “float” as their datatype, every number in that columns represent another category. For example, “CityIdentifier” column has “float32” datatype and includes numbers from 0 to 101118 but every number corresponds to different city where computer is located. To prevent such confusion, each column is analyzed using histograms (numerical) or bar plots (categorical). After analyzing every column, they are divided into five group: Categoric Variables, Categoric Identifiers, Version Identifiers, Binary Variables and Numerical Variables.

Columns in the Categoric Variables are true categorical columns where values are string. Some of them too many unique values which could create the curse of dimensionality problem.

Categoric Identifier columns have numerical type values but represent categorical information. “CityIdentifier” column is in this group.

Version Identifier columns indicates version of the softwares. Some of the column have numerical type and some of them have categorical type. Sample of version values are provided in Fig1.



EngineVersion	AppVersion	AvSigVersion	OsBuild	Census_OSVersion	Census_OSBuildNumber	Census_OSBuildRevision
1.1.15200.1	4.18.1807.18075	1.275.511.0	17134	10.0.17134.228	17134	228
1.1.15100.1	4.18.1807.18075	1.273.1228.0	17134	10.0.17134.191	17134	191
1.1.15100.1	4.18.1807.18075	1.273.1863.0	14393	10.0.14393.576	14393	576
1.1.15200.1	4.13.17134.228	1.275.1011.0	17134	10.0.17134.228	17134	228
1.1.15200.1	4.18.1807.18075	1.275.72.0	17134	10.0.17134.228	17134	228

Fig 2 Sample Values from Version Identifier Columns

Binary Variable columns are categorical columns where only 2 unique values in it. After handling missing values, they are ready to use in model.

Numerical Variable columns are true numeric columns where values are integer or float. Distribution of the data is analyzed to locate potential outliers in the columns.

All these categories have their own encoding technique due to their differences. Encoding techniques are discussed in data preprocessing part.

5) Data preprocessing:

From the beginning of the project to model development, we have encountered several issues related to characteristics of the data. Here is the issues and solutions we provided:

Memory Consumption: Dataset has nearly 9 million rows in it and there are too many columns in it. It is not feasible to do some operations on this dataset due to the size of it. To overcome this problem, we converted type of categorical columns from “object” to “categorical”. Also, we take maximum and minimum values of numerical columns and converted them into appropriate numerical type to reduce memory consumption.

Dominant Values: In some columns, there are “dominant values” where majority of the column belongs to that category or number. These dominant columns could be harmful for model’s performance. We set dominance threshold as 0.95 (If one category/number in a column takes 95% percent of the values) and dropped columns which has values higher than this threshold.

Redundant Columns: After analyzing every column, we realized that some of the columns holds the same information even their names are different. In order to remove repeated information, we removed one of the redundant columns from the dataset.

Many Unique Values: Most of the categoric columns have too many unique values, so applying one hot encoding to them become impossible. To reduce the number of categories in columns, first we analyzed value counts of each category, and we combined categories which have small counts as “other”. Also, in some columns, we combined some sub-categories to one category to reduce the dimensionality.

Missing Values: We analyzed each columns missing value percentage and decided to drop that column or impute missing values in it. If %50 or more values in the column are missing, then we dropped it. For other columns, we applied mode imputation to fill missing values.

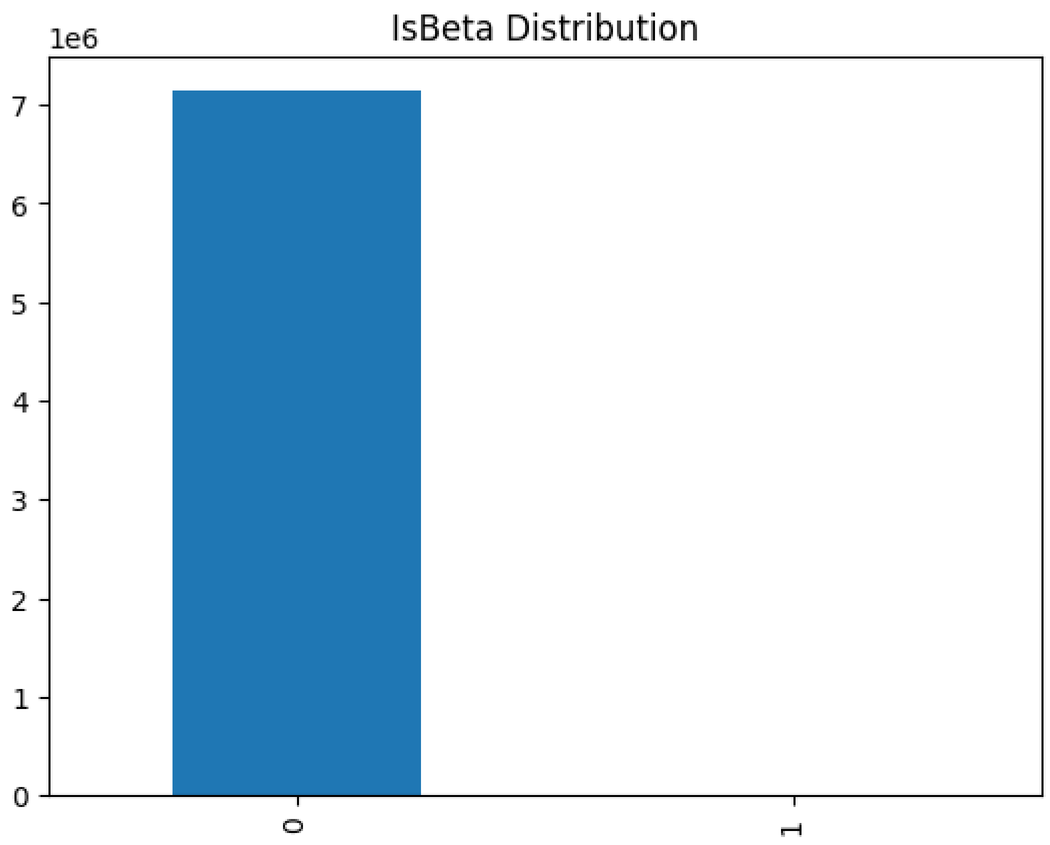


Fig 3 Example of column which contains "Dominant Value"

Encoding Categorical Columns: Since a significant portion of our features are categorical, a crucial step in data preprocessing involves transforming them into a format suitable for machine learning algorithms. We employed two primary encoding techniques based on the characteristics of each categorical column:

1. Frequency Encoding (for High Cardinality):
This technique is particularly effective for columns with a very high number of unique categories. Here's why we used it:

- **Curse of Dimensionality:** Applying one-hot encoding (which creates a separate binary column for each category) to high-cardinality features can significantly increase the dimensionality of the data. This phenomenon, known as the "curse of dimensionality," can negatively impact model performance and efficiency.
- **Informative Frequency:** Frequency encoding addresses this issue by replacing each category with its frequency (count) within the dataset. This approach captures the relative importance of each category based on its prevalence, potentially providing informative insights for the model.

2. One-Hot Encoding (for Low Cardinality):
For categorical features with a manageable number of unique categories, we utilized one-hot encoding. Here's when it's beneficial:

- **Efficient Representation:** When the number of categories is reasonable, one-hot encoding efficiently represents the data. It creates a new binary column for each category, with a value of 1 indicating the presence of that category and 0 indicating its absence. This allows the model to learn the relationships between specific categories and the target variable.
- **Clear Distinction:** One-hot encoding provides a clear distinction between each category, which can be advantageous for models that rely on understanding these relationships. It avoids potential biases that might arise from assigning numerical values (as in label encoding) to categories with no inherent order.

By strategically selecting between frequency encoding and one-hot encoding based on the number of unique categories in each feature, we ensured efficient data representation while preserving valuable information for our machine learning models.

6) Development Iterations

The development process of the malware detection model involved several iterations, each aimed at improving the model's accuracy, robustness, and interpretability. The first iteration focused on developing an initial model using all available features from the dataset. During this phase, we encountered high cardinality in categorical features, which necessitated the use of specific encoding strategies. We applied frequency encoding for high-cardinality categorical features to manage the dimensionality effectively, while one-hot encoding was used for low-cardinality features to maintain clear distinctions between categories. LightGBM was selected as the primary model due to its efficiency in handling large datasets with numerous categorical features. Additionally, we incorporated Multi-Layer Perceptrons (MLPs) to capture complex non-linear relationships in the data. This initial model provided a baseline for future improvements, and visualizations of feature importances and model predictions were created to interpret the results.

In the second iteration, the focus shifted to addressing data imbalances and refining feature selection to improve model performance. We identified and addressed dominant values in certain columns by setting a dominance threshold and dropping highly dominant features. Missing values were handled by dropping columns with over 50% missing data and applying mode imputation for others. We also reduced the number of unique values in categorical columns by combining less frequent categories, thus mitigating the curse of dimensionality. These preprocessing enhancements, along with refined encoding techniques, ensured optimal feature representation. This iteration led to an enhancement in model accuracy and provided deeper insights into the importance of different features.

The third iteration introduced explainable AI techniques to make the model's predictions more interpretable. We utilized SHAP (SHapley Additive exPlanations) and LIME values to interpret the impact of each feature on the model's predictions. Visualizations were developed to demonstrate how different features influenced the model's decisions, thereby providing transparency. Based on insights from SHAP and LIME values, we further refined the feature set and adjusted the model to focus on the most impactful features. This iteration achieved a balance between model accuracy and interpretability, enhancing trust in the model's predictions by offering clear explanations of how decisions were made.

Our LightGBM model achieved 0.7138 AUC score on training test and 0.7125 on test set. MLP model achieved 0.8093 AUC score on training set and 0.6776 on validation set after 30 epoch.

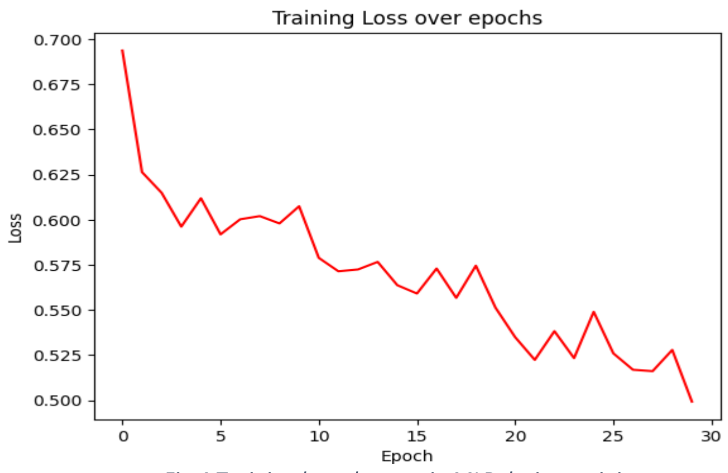


Fig 4 Training loss changes in MLP during training

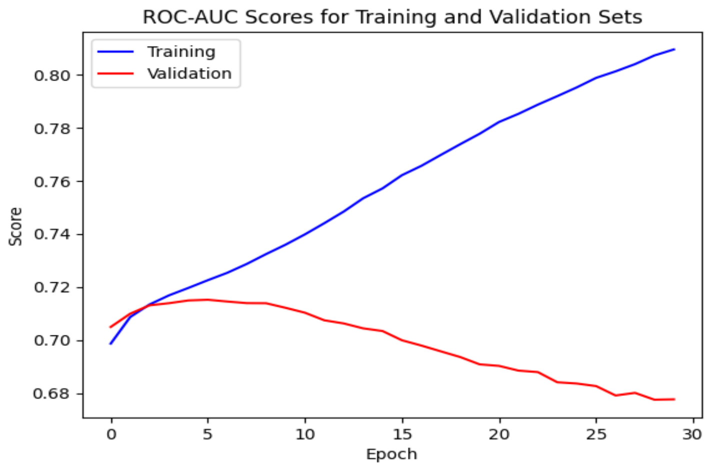


Fig 5 ROC Score changes in MLP during training on both train and validation sets

7) Model Evaluation

The initial LightGBM model achieved an AUC score of 0.7138 on the training set and 0.7125 on the test set. This indicated that the model performed consistently across both sets, suggesting good generalization capabilities. For the MLP model, after 30 epochs, it achieved an AUC score of 0.8093 on the training set but a lower AUC of 0.6776 on the validation set. The discrepancy between the training and validation scores suggested that the MLP model might be overfitting, capturing patterns specific to the training data that did not generalize well to new data. To avoid overfitting, we used early stopping and used the model weights where it achieved the highest validation score.

4. EXPERIMENTS

Experimental Setup

Data is splitted into train and test parts before preprocessing because some of the features are encoded using frequency encoding technique and test data shouldn't affect the frequencies in the training data. Another reason for doing that before is category concatenation and converting categories with low frequency as "other". If we didn't split them before, then frequencies of test data would have impact on training data, but test data should be unseen and independent from training data.

After the preprocessing of train and test data, validation split is created from training data to evaluate model's performance during training. For the MLP model, Binary Crossentropy Loss (BCELoss) is used as a loss function and model is trained with optimizer Adam (learning rate = 0.001) and batch size of 1024.

For the LightGBM model, we carefully selected a set of parameters to optimize performance. The parameter num_leaves were set to 2048 to control the complexity of the model by determining the maximum number of leaves in each tree, which helps in balancing accuracy and overfitting. The min_data_in_leaf parameter was set to 42, ensuring that each leaf has a minimum number of data points to prevent overfitting. The objective was set to binary as the task is a binary classification problem. A max_depth of 5 was chosen to limit the depth of each tree, helping in reducing overfitting while maintaining model complexity.

The learning_rate was set to 0.05, balancing the speed of learning and the ability to find the optimal solution. The boosting type was specified as gbdt (Gradient Boosting Decision Tree), which is effective for this type of classification task. The feature_fraction and bagging_fraction was both set to 0.8, and bagging_freq to 5, to enable random feature and data point sampling, enhancing the model's robustness. The regularization parameters lambda_l1 and lambda_l2 were set to 0.15 to add regularization and prevent overfitting.

After training models, we evaluate final performances using unseen test data. We also applied XAI techniques to identify which features are important for classification and which of them are not.

Experiment Results

As discussed above, our MLP model overfits the training data so its performance on validation data slightly decreases. We stopped the training at the 5.th epoch, where validation score is the highest (0.7151) and used that model to evaluate its performance on test data. We obtained 0.7146 AUC score on test set.

The LightGBM model was trained with early stopping using a patience of 10 epochs. This means that training would halt automatically if the validation AUC score did not improve for 10 consecutive epochs. In this case, early stopping did not trigger, indicating that the validation AUC score continued to improve throughout the training process with 100 boosting iterations. The best validation AUC score achieved was 0.713208. We obtained 0.7125 AUC score on test set.

After evaluation of the models, we used SHAP for MLP model, LIME and LightGBM's own feature importance functionality to see which features contributes most. LightGBM's feature importance table and LIME results are shown in Fig6. And Fig7. SHAP results are shown in Fig8., where the most impactful 20 features are listed.

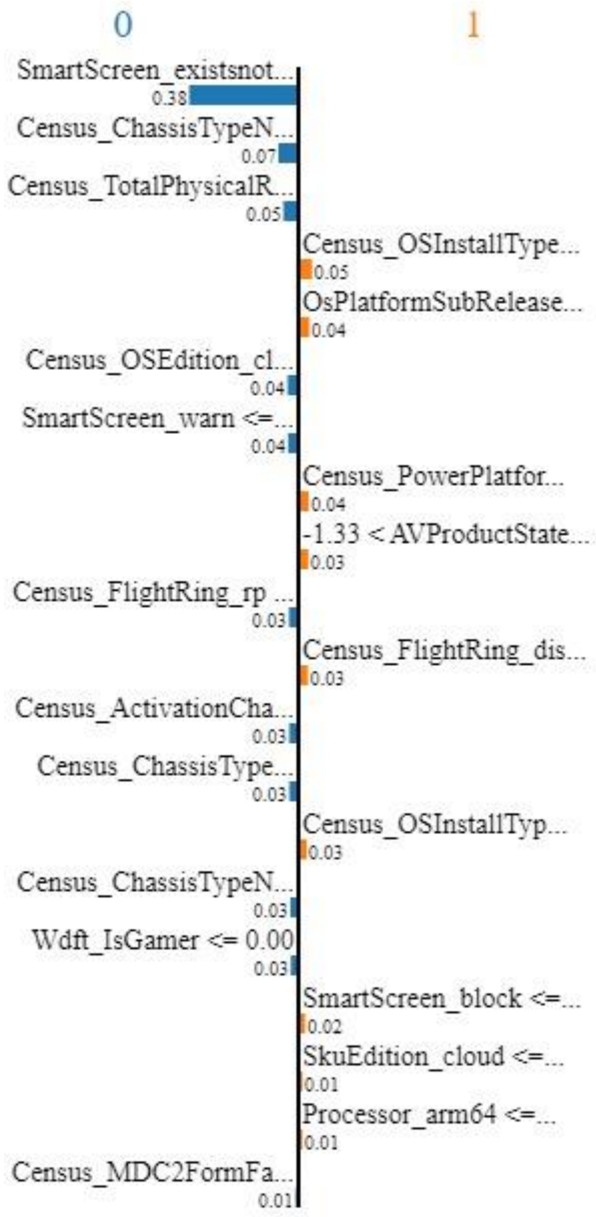


Fig 6 Results of LIME on LightGBM Model

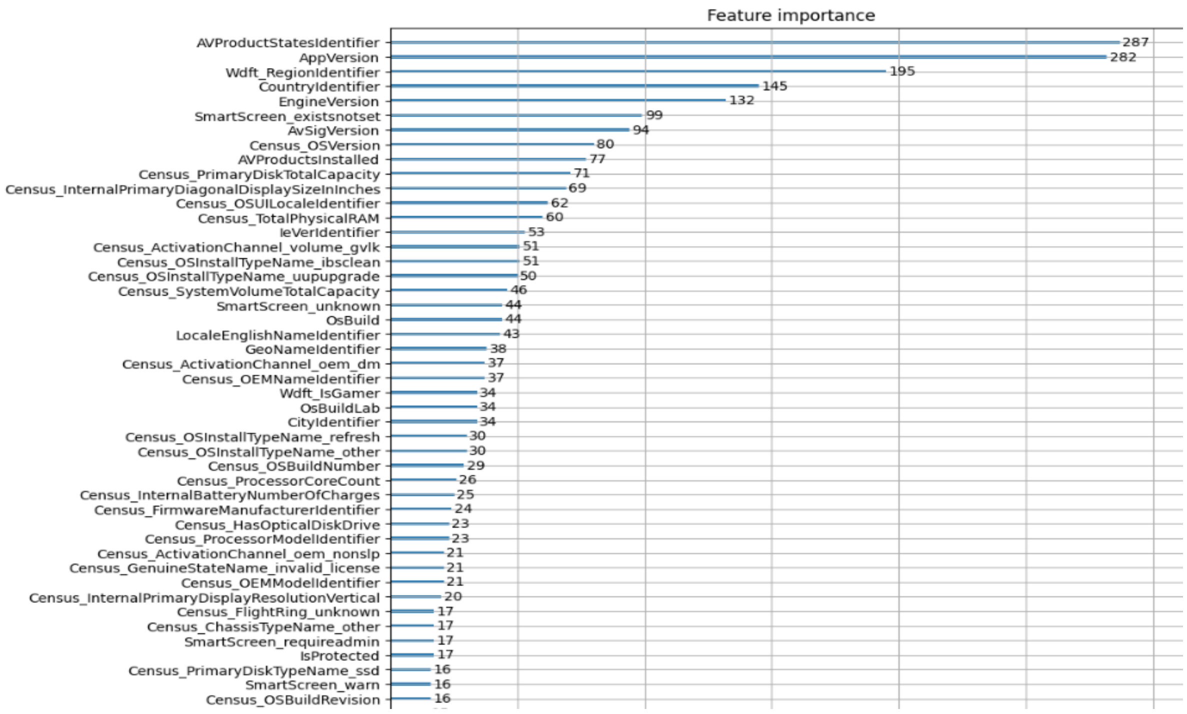


Fig 7 Feauture Importance Table of LightGBM Model

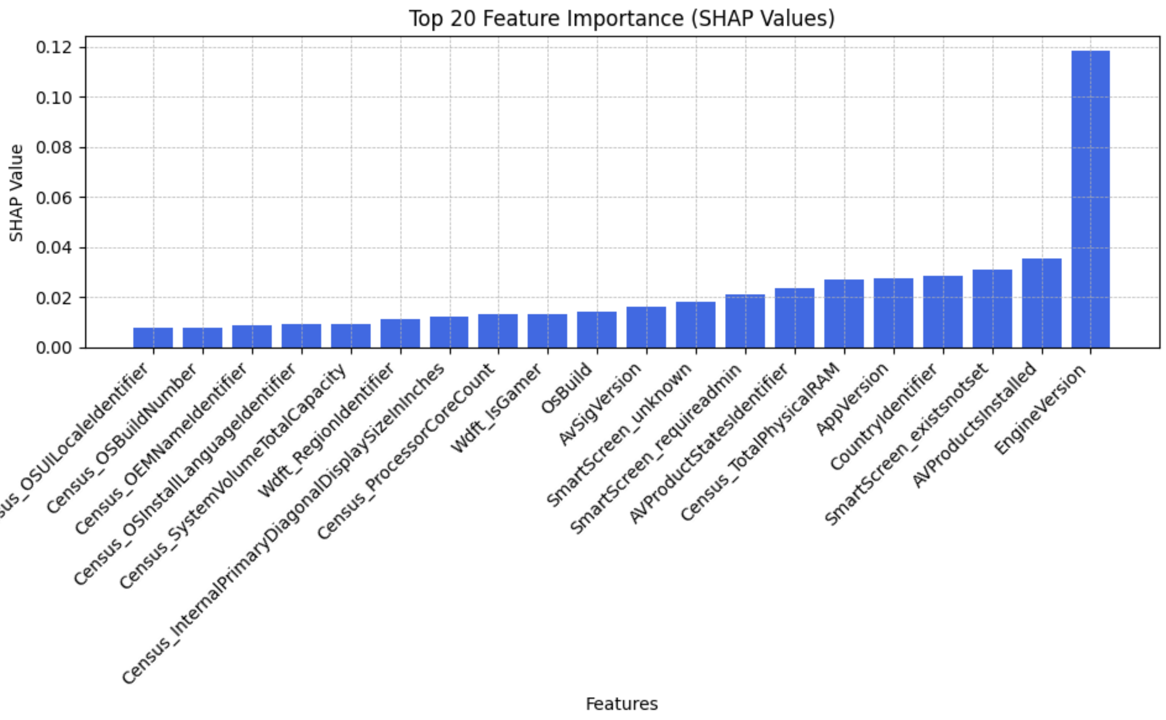


Fig 8 Results of SHAP on MLP Model

5. CONCLUSION

Our study aimed to develop a machine learning model for accurate and interpretable malware detection. We utilized all available features in our initial models and incorporated explainable AI techniques to achieve this goal.

The LightGBM model achieved an AUC score of 0.7138 on the training set and 0.7125 on the test set, demonstrating its potential for effectively detecting malware. Additionally, by incorporating SHAP and LIME, we gained valuable insights into how features influence the model's predictions. This interpretability can be crucial for understanding how the model identifies malware and can guide future feature engineering efforts.

However, there are limitations to consider. The MLP model also exhibited signs of overfitting. Further hyperparameter tuning and regularization techniques could be investigated to improve generalization. While SHAP and LIME provided some explainability, more advanced explainable AI methods could be explored for a deeper understanding of the model's decision-making process.

Looking ahead, several avenues for future work exist. Feature engineering techniques such as dimensionality reduction and feature creation could be explored to potentially improve model performance. Ensemble learning, which combines multiple models, could also lead to better results. Additionally, investigating deeper neural network architectures or exploring recurrent neural networks (RNNs) for capturing sequential patterns in the data could be beneficial. Finally, implementing more advanced explainable AI techniques could provide a more comprehensive understanding of the model's reasoning. By addressing these limitations and exploring these future work directions, we can continuously improve the accuracy, robustness, and interpretability of our malware detection model.

6. REFERENCES

[1] https://www.kaggle.com/competitions/microsoft-malware-prediction/overview

[2] A. bin Asad, R. Mansur, S. Zawad, N. Evan and M. I. Hossain, "Analysis of Malware Prediction Based on Infection Rate Using Machine Learning Techniques," 2020 IEEE Region 10 Symposium (TENSYMP), Dhaka, Bangladesh, 2020, pp. 706-709, doi: 10.1109/TENSYMP50017.2020.9230624. keywords: {Malware;Prediction algorithms;Decision trees;Neural networks;Training;Machine learning algorithms;Predictive models;malware prediction;machine learning algorithm;lgbm;neural network;k-fold;decision tree;microsoft malware dataset},

[3] Z. Zhang, "Microsoft Malware Prediction Using LightGBM Model," 2022 3rd International Conference on Big Data, Artificial Intelligence and Internet of Things Engineering (ICBAIE), Xi’an, China, 2022, pp. 41-44, doi: 10.1109/ICBAIE56435.2022.9985850. keywords:

{Measurement;Deep learning;Operating systems;Computational modeling;Predictive models;Network security;Malware;malware detection;feature engineering;LightGBM;Auc-Roc},

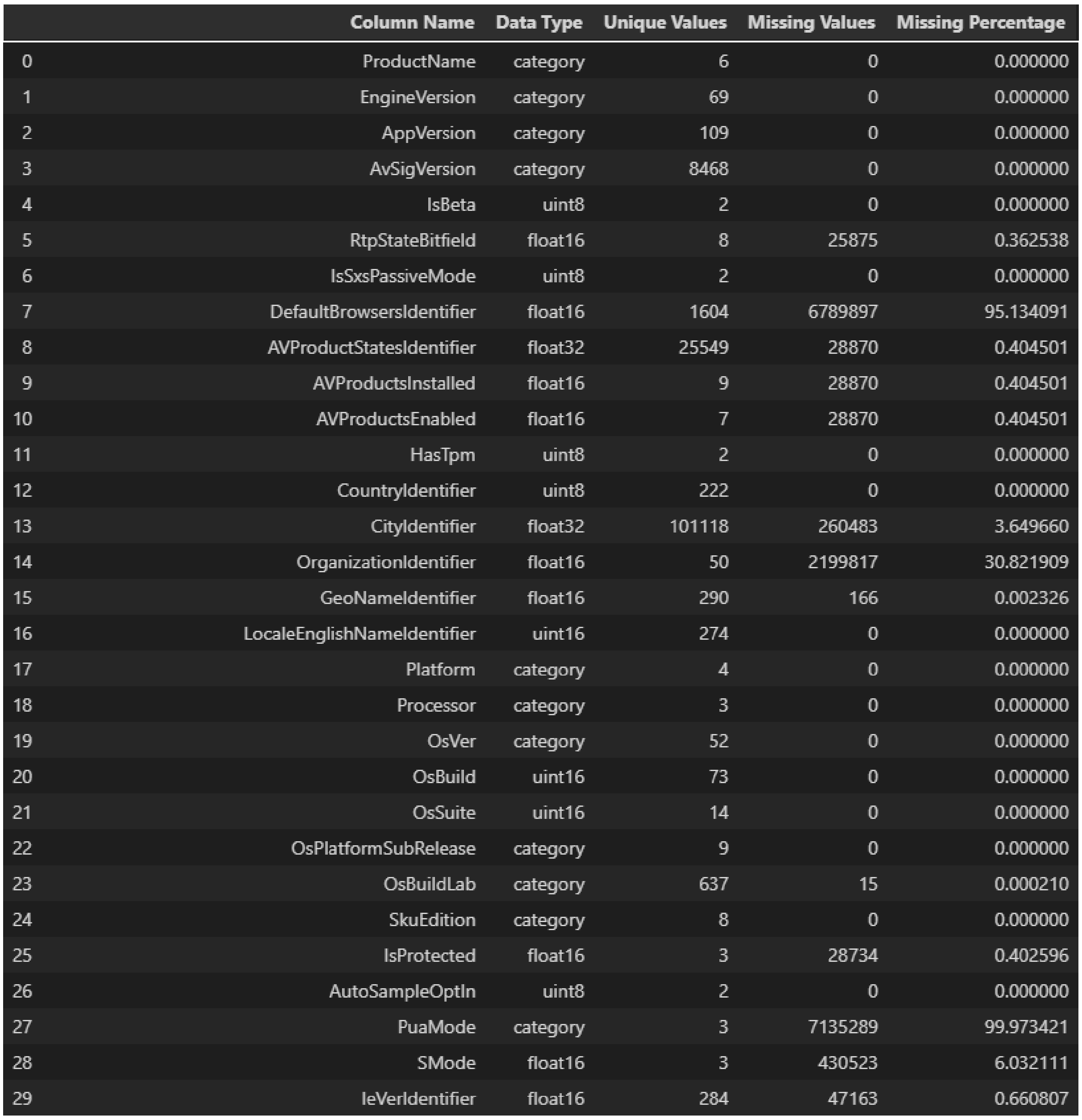
[4] Sokolov, Mark, and Nic Herndon. "Predicting Malware Attacks using Machine Learning and AutoAI." ICPRAM. 2021.

[5] Pan, Qiangjian, Weiliang Tang, and Siyue Yao. "The application of LightGBM in Microsoft malware detection." Journal of Physics: Conference Series. Vol. 1684. No. 1. IOP Publishing, 2020.

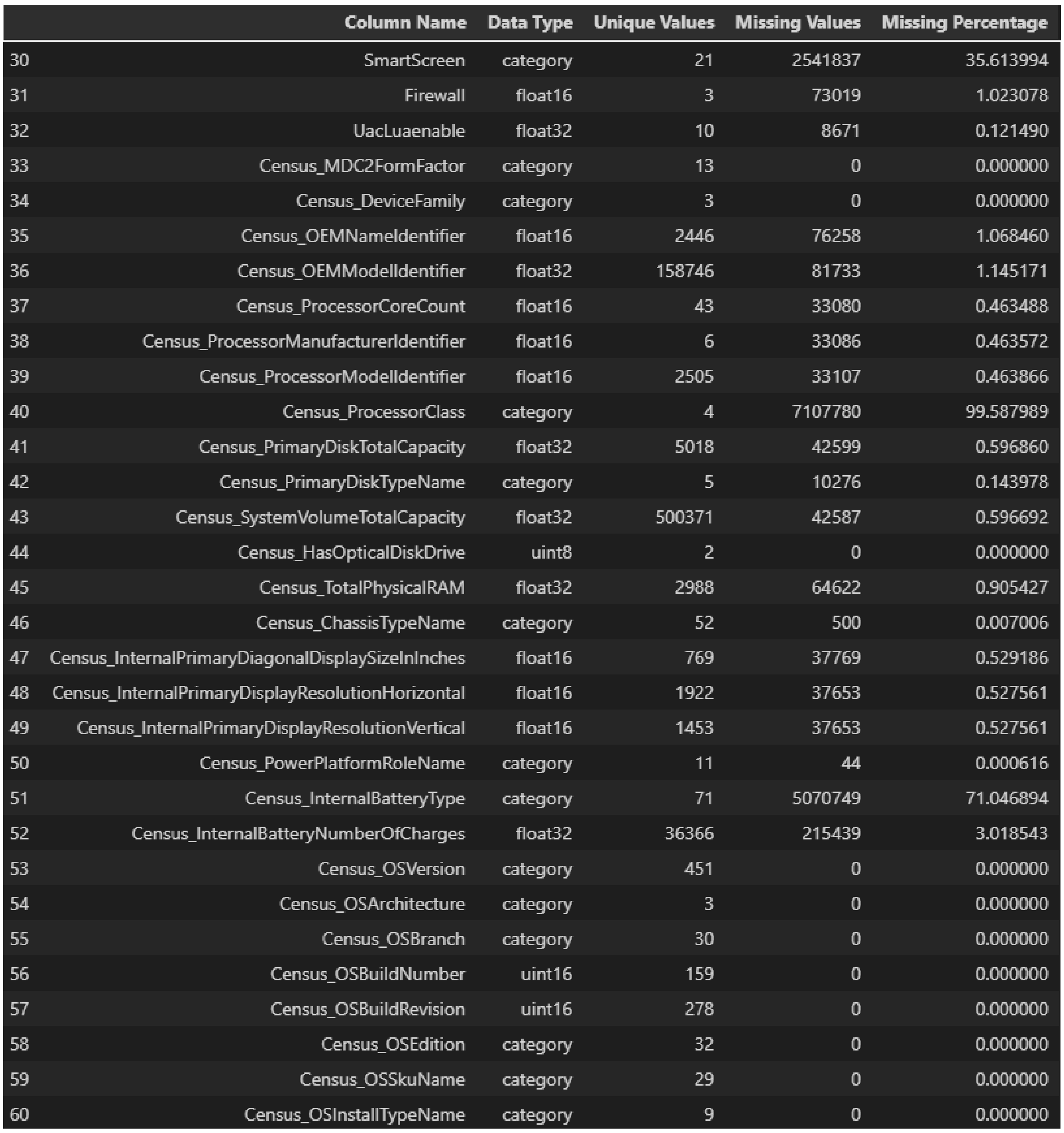
[6] Ambekar, Namrata Govind, N. Nandini Devi, and Surmila Thokchom. "TabLSTMNet: enhancing android malware classification through integrated attention and explainable AI." Microsystem Technologies (2024): 1-19.

APPENDIX

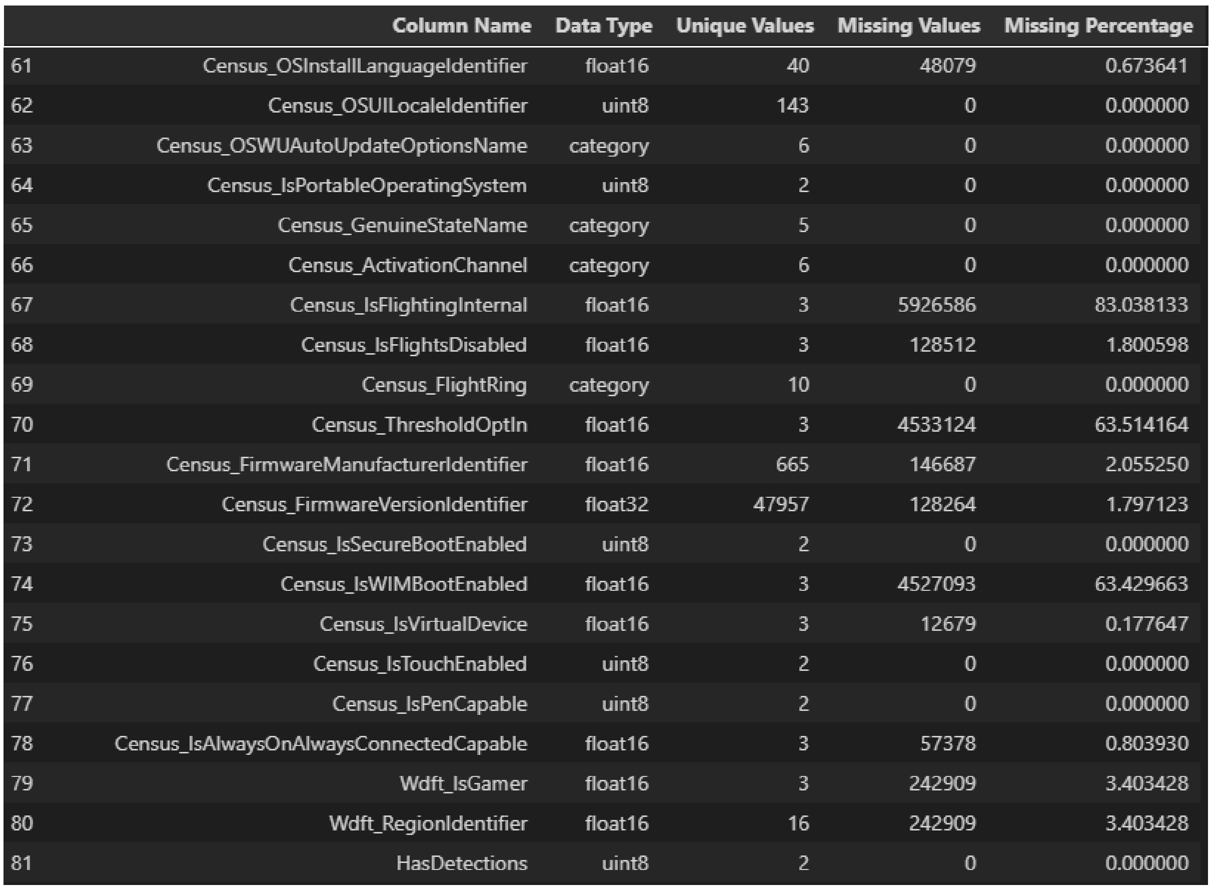
1) Attributes of All Features in the Dataset



	Column Name	Data Type	Unique Values	Missing Values	Missing Percentage
0	ProductName	category	6	0	0.000000
1	EngineVersion	category	69	0	0.000000
2	AppVersion	category	109	0	0.000000
3	AvSigVersion	category	8468	0	0.000000
4	IsBeta	uint8	2	0	0.000000
5	RtpStateBitfield	float16	8	25875	0.362538
6	IsSxsPassiveMode	uint8	2	0	0.000000
7	DefaultBrowsersIdentifier	float16	1604	6789897	95.134091
8	AVProductStatesIdentifier	float32	25549	28870	0.404501
9	AVProductsInstalled	float16	9	28870	0.404501
10	AVProductsEnabled	float16	7	28870	0.404501
11	HasTpm	uint8	2	0	0.000000
12	CountryIdentifier	uint8	222	0	0.000000
13	CityIdentifier	float32	101118	260483	3.649660
14	OrganizationIdentifier	float16	50	2199817	30.821909
15	GeoNameIdentifier	float16	290	166	0.002326
16	LocaleEnglishNameIdentifier	uint16	274	0	0.000000
17	Platform	category	4	0	0.000000
18	Processor	category	3	0	0.000000
19	OsVer	category	52	0	0.000000
20	OsBuild	uint16	73	0	0.000000
21	OsSuite	uint16	14	0	0.000000
22	OsPlatformSubRelease	category	9	0	0.000000
23	OsBuildLab	category	637	15	0.000210
24	SkuEdition	category	8	0	0.000000
25	IsProtected	float16	3	28734	0.402596
26	AutoSampleOptIn	uint8	2	0	0.000000
27	PuaMode	category	3	7135289	99.973421
28	SMode	float16	3	430523	6.032111
29	IeVerIdentifier	float16	284	47163	0.660807



	Column Name	Data Type	Unique Values	Missing Values	Missing Percentage
30	SmartScreen	category	21	2541837	35.613994
31	Firewall	float16	3	73019	1.023078
32	UacLuaenable	float32	10	8671	0.121490
33	Census_MDC2FormFactor	category	13	0	0.000000
34	Census_DeviceFamily	category	3	0	0.000000
35	Census_OEMNameIdentifier	float16	2446	76258	1.068460
36	Census_OEMModelIdentifier	float32	158746	81733	1.145171
37	Census_ProcessorCoreCount	float16	43	33080	0.463488
38	Census_ProcessorManufacturerIdentifier	float16	6	33086	0.463572
39	Census_ProcessorModelIdentifier	float16	2505	33107	0.463866
40	Census_ProcessorClass	category	4	7107780	99.587989
41	Census_PrimaryDiskTotalCapacity	float32	5018	42599	0.596860
42	Census_PrimaryDiskTypeName	category	5	10276	0.143978
43	Census_SystemVolumeTotalCapacity	float32	500371	42587	0.596692
44	Census_HasOpticalDiskDrive	uint8	2	0	0.000000
45	Census_TotalPhysicalRAM	float32	2988	64622	0.905427
46	Census_ChassisTypeName	category	52	500	0.007006
47	Census_InternalPrimaryDiagonalDisplaySizeInInches	float16	769	37769	0.529186
48	Census_InternalPrimaryDisplayResolutionHorizontal	float16	1922	37653	0.527561
49	Census_InternalPrimaryDisplayResolutionVertical	float16	1453	37653	0.527561
50	Census_PowerPlatformRoleName	category	11	44	0.000616
51	Census_InternalBatteryType	category	71	5070749	71.046894
52	Census_InternalBatteryNumberOfCharges	float32	36366	215439	3.018543
53	Census_OSVersion	category	451	0	0.000000
54	Census_OSArchitecture	category	3	0	0.000000
55	Census_OSBranch	category	30	0	0.000000
56	Census_OSBuildNumber	uint16	159	0	0.000000
57	Census_OSBuildRevision	uint16	278	0	0.000000
58	Census_OSEdition	category	32	0	0.000000
59	Census_OSSkuName	category	29	0	0.000000
60	Census_OSInstallTypeName	category	9	0	0.000000



	Column Name	Data Type	Unique Values	Missing Values	Missing Percentage
61	Census_OSInstallLanguageIdentifier	float16	40	48079	0.673641
62	Census_OSUILocaleIdentifier	uint8	143	0	0.000000
63	Census_OSWUAutoUpdateOptionsName	category	6	0	0.000000
64	Census_IsPortableOperatingSystem	uint8	2	0	0.000000
65	Census_GenuineStateName	category	5	0	0.000000
66	Census_ActivationChannel	category	6	0	0.000000
67	Census_IsFlightingInternal	float16	3	5926586	83.038133
68	Census_IsFlightsDisabled	float16	3	128512	1.800598
69	Census_FlightRing	category	10	0	0.000000
70	Census_ThresholdOptIn	float16	3	4533124	63.514164
71	Census_FirmwareManufacturerIdentifier	float16	665	146687	2.055250
72	Census_FirmwareVersionIdentifier	float32	47957	128264	1.797123
73	Census_IsSecureBootEnabled	uint8	2	0	0.000000
74	Census_IsWIMBootEnabled	float16	3	4527093	63.429663
75	Census_IsVirtualDevice	float16	3	12679	0.177647
76	Census_IsTouchEnabled	uint8	2	0	0.000000
77	Census_IsPenCapable	uint8	2	0	0.000000
78	Census_IsAlwaysOnAlwaysConnectedCapable	float16	3	57378	0.803930
79	Wdft_IsGamer	float16	3	242909	3.403428
80	Wdft_RegionIdentifier	float16	16	242909	3.403428
81	HasDetections	uint8	2	0	0.000000