Enhancing AI-Powered Malware Detection by Parallel Ensemble Learning

Hoang V. Vo*, Phong H. Nguyen*, Hau T. Nguyen*, Duy B. Vu*, and Hoa N. Nguyen*
*Department of Information Systems, VNU University of Engineering and Technology,
Vietnam National University at Hanoi, Vietnam
Email: [vohoangmocvy, 20020065, nguyenhau, duyvb, hoa.nguyen]@vnu.edu.vn

2023 RIVF International Conference on Computing and Communication Technologies (RIVF) | 979-8-3503-1584-4/23/$31.00 ©2023 IEEE | DOI: 10.1109/RIVF60135.2023.10471855

***Abstract*—The efficacy of signature-based malware detection is limited in detecting novel forms of malware that are created through obfuscation and deformation methods. AI-powered techniques are extensively utilized to detect malware by employing classification models that analyze its behavioral patterns. This research presents a novel malware detection approach, referred to as PEMA, which integrates multiple effective AI models that are carefully chosen through an automated machine learning framework. Parallel ensemble learning with XGBoost, CatBoost, and LightGBM algorithms is employed by PEMA to make deep malware analysis faster and more accurate. Since PEMA was put together using three separate models, it can lessen the likelihood that it will be susceptible to poisoning and adversarial attacks. The performance of PEMA is verified by experimenting with well-known datasets, such as the EMBER 2017 and 2018, for evaluation purposes. As a result, PEMA attained F1 scores of 99.41% and 97.64%, respectively. Furthermore, it exhibits superior performance compared to other recent methods utilizing the same datasets and improves significant detection time.**

***Index Terms*—AI-powered Malware Detection, Malware Analysis, Parallel Ensemble Learning.**

I. Introduction

Malware is harmful software intended to disrupt operations, prevent activities, collect user information without authorization, gain unauthorized access to system resources, and engage in other undesirable behaviors [1]. The exponential proliferation of malware has become one of the main risks to computer security with the quick development of information technology. As the variety and quantity of applications in computer security increase, malware identification has gotten more difficult [2]. Hundreds of millions of new malware programs make signature-based malware detection increasingly unfeasible every year. Machine Learning (ML) or Intelligent Artificial (AI) may help us detect malware without unique signatures of its data or behavior, making it a useful countermeasure against trendy malware. This method is considered *AI-powered malware detection*.

Dynamic and static malware detection are two categories of malware detection approaches [3]. Dynamic methods identify malicious software based on its runtime behavior, while static approaches classify samples as malicious or benign without actually executing them. Nevertheless, gathering datasets of malware behavior is difficult because malware can recognize sandbox environments and avoid carrying out destructive deeds. Additionally, dynamic malware detection in a real-world setting necessitates using numerous sandboxes to handle many suspect samples, raising the detection cost [4]. In contrast, enormous datasets can be produced by aggregating binary files and detecting malware before it is executed, even if static malware detection is typically indefensible.

AI-powered malware detectors often use static or dynamic analysis to extract the attributes of malicious programs (and, ideally, benign applications) (portable executable (PE) files) [5]. They subsequently acquire the features to build models to determine whether a particular software is malware. The models determine how likely it is that a specific unknown program is malicious software. The time needed to learn program features when the number or size of features is vast or the number of programs is huge is a challenge that frequently arises in AI-powered malware detection. Although fewer features speed up learning, the detection accuracy will probably suffer [6]. Although reasonable accuracy, a quick learning curve, and small data sets can all be achieved with a mix of well-chosen characteristics, doing so is more complex.

Research Challenges: In summary, there exist "barriers" to the design of PE malware detection methods, with the simplest following challenges:

- Traditional PE malware detection methods often rely on signatures, specific patterns known to be associated with certain types of malware. However, attackers can exploit this by crafting malicious code that evades signature-based detection. It highlights the importance of having diverse detection mechanisms that do not solely rely on signatures to catch malware.
- Effective PE malware detection should focus on accuracy and speed. A model's prediction time becomes crucial in such scenarios. Slow predictions could lead to frustration among users and negatively impact their experience. Thus, optimizing the model's speed while maintaining accurate detection becomes challenging.
- When using AI models to detect malware, if the classification algorithm is compromised, AI-powered malware detectors may face adversarial attacks that aim to spread malware and infect the system. Therefore, reducing the likelihood of infection and enhancing resilience against adversarial attacks on AI-powered malware detection methods is an important challenge that needs to be addressed nowadays.

Highlight Contribution: From the challenges mentioned, we

advocate the AI-powered malware detection method for PF files in this study, with the following main results:

- PE files are modeled as attribute sets and then analyzed by an ensemble model combining several AI models in order to improve the accuracy of malware detection.
- Optimizing the PE features by removing redundant ones, weight scores, and tuning the hyperparameters for each individual AI model.
- Improving the execution time of our AI-powered malware detector through parallel execution of all predictors.
- Reducing the likelihood that it will be susceptible to poisoning and enhancing resilience against adversarial attacks for our AI-powered malware detector.

The rest of this paper is organized as follows: Section §II presents the related work. Section §III describes our proposed method for malware detection in detail, including the overall architecture. In Section §IV, we evaluate the performance of our proposed method in terms of accuracy and speed. Finally, Section §V concludes our work and indicates future works.

II. Related Works

ML/DL-based malware detection has attracted the attention of numerous researchers. This section shows our investigation of State-Of-The-Art (SOTA) malware detection methods regarding (i) Neural Network-based Malware Detection and (ii) Boosting-based Malware Detection.

A. Neural Network-based Malware Detection

Neural Network (NN) approaches effectively detect malware and network attack associations within raw samples, feature learning, and classification tasks. Numerous NN techniques have been implemented in the last few years [7]–[9]. Some studies on attack detection use DL in a real-time environment. For example, Divakarla et al. [10] presented a simple deep neural network-based Windows malware detection system that achieves a test accuracy of 96.76%. Moreover, the authors also perform an improved offensive generative model based on GANs to make the current DNN-based system accurate at 97.42%. This work proves that DNN combined with rigorous static analysis helps build a malware detection system; it can learn complex features with a greater number of layers and more data.

Liu et al. [11] propose a generic ML-based visualization method for malware detection named Visual-AT. In addition, it employs the AT technique to detect and analyze the originally difficult-to-identify malware and potential variants with the transformed image data through two ML models. The Visual-AT achieves up to 97.73% accuracy for the EMBER 2018.

Moreover, Rigakia et al. [12] proposed a method that trains multiple types of surrogate models and sampling strategies to steal stand-alone ML models and four antivirus systems. This method presented a dual FFNN architecture, achieving 98.02% accuracy for the EMBER 2018.

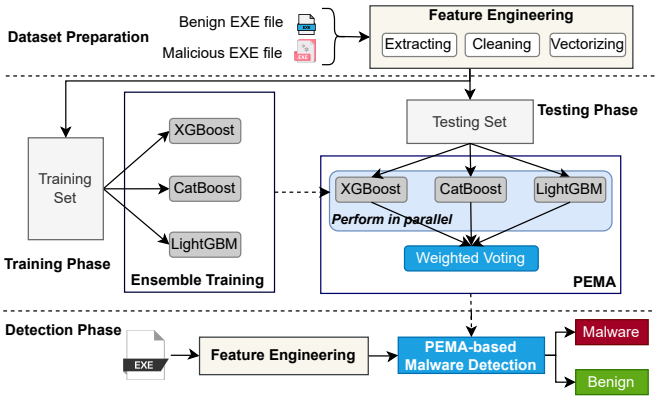

Fig. 1. Architecture of Parallel Ensemble Learning for Malware Detection

B. Boosting-based Malware Detection

The XGB is faster than other popular algorithms in a single ML, gaining popularity due to its performance [8]. Recently, Devan et al. [13] presented a method that uses the XGB technique for feature selection followed by a DNN and gets 97.60% of accuracy. In addition, Mimura et al. [14] proposed a method for malware detection on PE files using printable characters using two language models for feature extraction and machine learning. The author uses the latest FFRI dataset between 2019 and 2021 to evaluate the method. According to the results, the XGB model achieves an accuracy of 99.0%, and the most suitable mix was the Doc2vec and multilayer perceptron, which achieved an F1 score of 98.10%. Each run time showed an almost linear increase with increasing dataset size.

In other approaches, Alani et al. [14] introduced a lightweight obfuscated malware detector based on explainable machine-learning techniques. The authors use the feature selection method (RFE) to reduce the number of features while effectively maintaining high accuracy. This approach improved the system's efficiency when evaluated using the MalMem2022 dataset and achieved a remarkable accuracy of over 99.8% by utilizing only five carefully selected features.

III. Proposal Method

A. Overview

This work presents a novel approach to address the challenge encountered in AI-powered malware detection, as outlined in §I. The proposed method involves the integration of different AI models to improve the accuracy of PE malware classification and bolster resilience against certain types of obfuscation, deformation or adversarial attacks in malware. Moreover, it allows leveraging the parallel execution capabilities of individual AI models within multi-core, multi-chip computing systems, thereby improving the speed of malware detection. The method employed in our study is referred to as PEMA, an acronym for *"**P**arallel **E**nsemble learning-based **M**alware Detection **A**lgorithm"*. Fig. 1 depicts the comprehensive framework of our PEMA method.

Similar to traditional machine learning and specifically deep learning, data preparation is one of the crucial steps, directly impacting the quality of AI models [15]. Currently, one of the best-known datasets commonly used in malware detection is the EMBER datasets (published in 2017 and 2018) [16]. Hence, our study also focuses on harnessing and utilizing this dataset to train AI models. The EMBER dataset provides the raw attributes to aid human readability. It also provides code that transforms these raw features into a numeric feature vector that can be used in model building. Thus, we make use of that code to generate such numeric vectors, which we use for model development, evaluation, and experiments in this study.

During the dataset preparation, PE executable files collected from various sources need to be analyzed, evaluated, and filtered to remove errors or corrupted files. Subsequently, the remaining PE files, both malware and benign, undergo feature extraction and are transformed into feature vectors for training AI models. In PEMA, the data preparation is facilitated by the Feature Engineering phase, consisting of three steps: Feature Extracting, Feature Cleaning, and Feature Vectorizing. The output of this phase is the feature vector for each PE file. The specific steps within this phase are detailed in §III-B.

Based on the Feature Engineering phase, the EMBER datasets are processed to create sets of feature vectors for both training and testing datasets. The training dataset is used to train individual AI models, while the testing dataset is employed to evaluate both individual models and the combined PEMA model for malware detection. Further specifics on our malware detection method using PEMA are elaborated in §III-C.

B. Feature Engineering

1) Feature Extraction: Feature extraction is also one of the important processes in machine learning. It is the process of extracting a set of features from a given dataset. The feature extraction will reduce the feature space of the dataset. Reducing the feature space will avoid unnecessary processing of the whole dataset while training and reduce the computation overhead [8]. EMBER dataset, consisting of PE files in JSON format. In our work, Library to Instrument Executable Formats (LIEF) are used to extract and represent a PE file by a set of 2381 features. This library is used for both the EMBER 2017 and 2018. The detailed description of these features is described in [17].

2) Feature Cleaning: The quality of AI models is also influenced by the set of attributes in the training samples. Therefore, eliminating redundant attributes that do not affect malware detection is one of the tasks of feature engineering. To accomplish this, we utilized an AutoML toolkit called Autogluon [18] to analyze, evaluate, and remove redundant attributes. From experimentation using both the EMBER 2017 and 2018 datasets, we identified 90 redundant attributes out of the 2381 attributes representing each PE sample. As a result, the outcome of the Feature Cleaning step yielded a set of 2291 important attributes for forming the feature vectors.

Moreover, there are training and testing sets in the EMBER datasets. Benign, malicious, and unlabeled data are all available in training datasets in three categories. The dataset defines these categories as -1, 0, and 1, respectively. However, there are no unlabeled data in the testing sets that are part of the dataset. In order to strike a compromise, we excluded unlabeled samples from further processing and rebuilt the training set using only labeled data. This step will equalize the training and testing sets and enhance the deep learning model's performance.

3) Feature Vectorizing: After the extraction and cleaning based on LIEF, the output is usually modeled in JSON format. The latter could not be ingested for training AI models since many famous models, such as the GBMs family model, only accept input as a number vector instead of a JSON format file. Thus, vectorizing is performed to obtain binary format features that can be later stored in CSV for later usage. Since most features of the EMBER dataset have unique value and cannot be categorized easily, the hashing technique is suitable to retain the features of the data [17]. Consequently, in this work, we use the Feature Hashing technique to vectorize them into the feature vector.

C. Parallel Ensemble Learning for Malware Detection

With the approach of using ensemble learning, the analysis, evaluation, and selection of individual AI models to combine in the ensemble model are crucial [19]. This can be effectively achieved through practical experimentation using AutoML frameworks. In our study, Autogluon was employed to evaluate and select the most efficient AI models for incorporation into PEMA. Based on experimental results with the training data from the EMBER 2017 and 2018 datasets, three models, namely XGBoost (XGB), CatBoost (CatB), and LightGBM (GBM), were identified as the most effective models for malware detection. Consequently, they were chosen as individual models to be jointly trained and utilized within our ensemble model PEMA.

From the training datasets of EMBER 2017 and 2018, and after each AI model has been trained, Alg. 1 shows how PEMA-based malware detection can be done. In this algorithm, to accelerate the malware detection process with PEMA, we propose to perform the classification process using the three individual models (XGB, CatB, and GBM) in parallel. After all three parallel classification processes of the models are completed, the classification probabilities from each model are aggregated using a weighted voting strategy. The determination of the weight score of each AI model will be based on the optimization process of these weights using an optimization tool such as Optuna [20].

Moreover, adversary attacks usually use obfuscation and deformation techniques to generate new types that can evade malware detection methods. However, our method combines several machine learning models to detect malware; in this way, an input PE file that embeds malware will be tested in these models. If this model is not detected, then other models may detect it. This work creates the ability to detect and prevent adversary attacks [21].

Algorithm 1 PEMA: **P**arallel **E**nsemble learning-based **M**alware Detection **A**lgorithm

Global: XGB - XGBoost trained model; $CatB$ - CatBoost trained model; GBM - LightGBM trained model, and their weighted score ω_i where $\sum_{i=1}^{3} \omega_i = 1$.
Input: f - PE file.
Output: 0 for Benign or 1 for Malware.

$F \leftarrow LIEF(f)$ ▷ extract features of PE file f by employing the LIEF tool.
$Fin \leftarrow F \backslash F_{unless}$ ▷ remove 90 unused features represented by F_{unless}.
Perform in parallel three processes P1,P2,P3:
P1: $p_X \leftarrow XGB.predict(Fin)$ ▷ perform the prediction using XGBoost model.
P2: $p_C \leftarrow CatB.predict(Fin)$ ▷ perform the prediction using CatBoost model.
P3: $p_G \leftarrow GBM.predict(Fin)$ ▷ perform the prediction using GBM model.
Wait P1, P2, P3 finished.
$score \leftarrow (p_X * \omega_1 + p_C * \omega_2 + p_G * \omega_3)$
$L \leftarrow scores.argmax(axis = 1)$ ▷ compute the final class L of f.
return L

IV. Experiment and Evaluation

In order to prove the performance of PEMA, we conduct intensive experiments using well-known datasets and compare it with other SOTA methods. Our computing platform contains 2 x Intel Xeon Platinum 8160 (24-cores), 384GB of DDR4 RAM, 6.0TB of SSD with the following libraries and frameworks: Pandas v1.5.3; Numpy v1.23.5; Scikit-learn v1.2.2; AutoGluon v0.7.0; XGB v1.7.5; CatB v1.1.1; GBM v3.3.5. We also use Optuna [22] to optimize the hyperparameters for the AI models.

A. Dataset Preparation

The EMBER dataset version 2017 and 2018 [16] is used to evaluate the effectiveness of the PEMA method. Nevertheless, we also used the feature engineering technique described in §III-B to obtain the best predictive for the PEMA model.

As a result, we found 90 useless features in the dataset and removed them in the training and validating process, which has column id including: "627, 636, 638, 648, 650, 652, 659, 663, 666, 667, 669, 670, 672, 673, 674, 675, 676, 849, 859, 862, 867, 883, 891, 894, 896, 898, 899, 900, 904, 905, 908, 909, 910, 912,917, 919, 920, 922, 923, 924, 926, 931, 932, 934, 936, 937, 629, 630, 635, 651, 653, 671, 861, 864, 871, 878, 884, 889, 890, 907, 911, 913, 914, 918, 925, 933, 935, 938, 939, 942, 962, 974, 989, 997, 1022, 1042, 1044, 1052, 1053, 1056, 1057, 1069, 1086, 1120, 1125, 1151, 1173, 1179, 1184, 2367". After performing the feature engineering, we obtained the datasets DS1 from EMBER 2017 and DS2 from EMBER 2018. The detailed statistics of these datasets are shown in Table I.

TABLE I
Dataset Statistics

Dataset	Label	Training Set	Testing Set
DS1	Malware (1)	$300K$	$100K$
	Benign (0)	$300K$	$100K$
DS2	Malware (1)	$300K$	$100K$
	Benign (0)	$300K$	$100K$

TABLE II
Hyperparameter Optimization

Model	Hyperparameter	Value	Optimal
XGB	Learning rate	[0,∞]	0.9
	n_estimators	[1,∞]	300
	Max_depth	[0,∞]	15
CatB	Learning rate	[0,∞]	0.5
	iterations	[1,∞]	5000
	random_seed	[0,∞]	0
GBM	Num iterations	[0,∞]	1300
	Learning rate	[0,1]	0.9
	Num leaves	[0,∞]	2000
	Feature fraction	[0,1]	0.9
	Max_depth	[0,∞]	15
	Min data in leaf	[0,∞]	2

B. Hyperparameter Optimization

We select model parameters based on a technique called Optuna [20], a hyperparameter tuning framework that enables the dynamic construction of the search space. It allows the combination of efficient searching and pruning in order to improve the cost-effectiveness of optimization.

After optimization, our final EMBER datasets' final hyperparameters are shown in Table II. For example, the catB model's hyperparameters can accept values from the range and choice categories. The $learning\_rate$ was set at 0.05, the number of $iterations$ at 5000, and the $random\_seed$ value at 0. For the remaining machine-learning models, the same techniques are used.

We also created a Python tool to adjust the weight ratios of three models in our ensemble learning model to get the best model. The total weight ratio is always 1; every time, the weight ratio is altered by 0.01. Eventually, the XGB, CatB, and GBM models have weight ratios of $0.3, 0.2$, and 0.5, respectively.

C. Evaluation Metrics

To evaluate the malware detection method, we use some common metrics computed from the confusion matrix, such as Accuracy (Acc), Precision (Prec), F1-score (F1), and Recall (Rec) [23].

D. Results & Evaluation

Based on the PEMA method described in Section III, we built the AI-powered malware detection tool successfully. To

TABLE III
PERFORMANCE EVALUATION (%)

Metric	*S1: EMBER 2017*				*S2: EMBER 2018*			
	XGB	CatB	GBM	**PEMA**	XGB	CatB	GBM	**PEMA**
Acc	99.29	99.37	99.35	**99.41**	97.35	97.12	97.29	**97.65**
Prec	99.54	99.51	99.51	**99.63**	97.43	97.10	97.43	**97.80**
F1	99.29	99.36	99.35	**99.41**	97.35	97.12	97.29	**97.64**
Rec	99.04	**99.22**	99.19	99.19	97.26	97.15	97.15	**97.49**
AUC	99.96	99.96	**99.97**	99.59	99.48	99.47	99.47	**99.61**

evaluate PEMA, we perform the two scenarios, described as follows:

- Scenario $S1$: use the DS1 to test our model. The reason is that many old papers only cover the DS1, so we cover this scenario to evaluate our models' effectiveness and compare it with other recent methods.
- Scenario $S2$: use the DS2 to test our model. Since this dataset is described as harder than its former version, we cover this version to compare it with the DS1.

Based on the above-tuned hyperparameters, we use *multi-processing* parallel to get a prediction from all three models simultaneously to reduce our final model's prediction time in both scenarios. After calculating the averaging probability prediction from the child model, the $argmax$ function is utilized to get the label with the highest probability, thus getting the best final results.

1) S1 Evalution: In this scenario, the models start without any specified parameters since the above algorithm has a mechanic to generate hyperparameters based on the dataset automatically. Based on running all three models on the DS1 without tuning hyperparameters, the experiment result saw that with XGB and CatB, accuracy is already above 99%. That is reasonable since many researchers, including original authors of the dataset, commented that the EMBER dataset is pretty *"easy"*. Thus, we concentrated on tuning GBM - the model has the lowest accuracy. The experiment result of S1 is shown in Table III

2) S2 Evalution: On the DS2, the same strategy as DS1 - run without hyperparameters first, then tune the model around initial parameters. In this case, unlike the 2017 dataset, the initial accuracy could not be better this time. Thus, this is partly due to DS2 being harder than its previous version. The accuracy for the XGB model is 97.35%, the CatB model is 97.12%, and the CatB model's is 97.29%. Finally, the PEMA gets 97.65% of accuracy. The evaluation results are shown in Table III.

3) Speed Evaluation: For evaluating the performance of PEMA in terms of speed, all experiments of PEMA are measured for execution time, both for the three individual AI models and PEMA. To avoid the influence of other processes, we performed the experiments and measured the execution time over an average of 16 run times. In both scenarios, our approach reduces prediction time by performing base models in parallel. For DS1, we reduce analysis time from $2645.06ms$

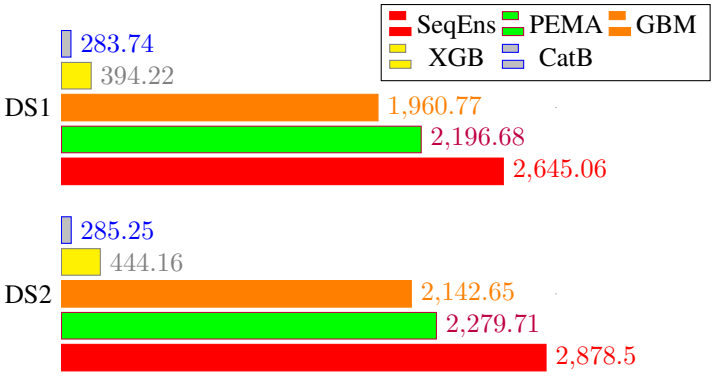

Fig. 2. Time Evaluation of PEMA-based Malware Detection (ms)

TABLE IV
COMPARISON WITH SOTA METHODS (%)

Method	Venue	Acc	Prec	F1	Rec
DS1: EMBER 2017					
PEMA (our)		**99.41**	**99.63**	**99.41**	**99.19**
AutoML [24]	2023	99.20	–	99.20	–
DNN [25]	2022	97.53	98.85	95.23	95.40
KiloGrams [26]	2019	99.20	–	–	–
DS2: EMBER 2018					
PEMA (our)		**97.65**	**97.80**	**97.64**	**97.49**
DL [27]	2023	95.57	–	–	–
AutoML [24]	2023	95.80	–	95.80	–
MLMD [10]	2023	97.42	–	–	–
DNN [25]	2022	94.09	90.14	88.66	88.85
TransferLearning [17]	2021	94.40	–	–	–
Pipelining [28]	2021	96.90	–	–	–
RF [29]	2020	95.51	96.90	–	94.9

for testing 200,000 samples by using the sequence ensemble of XGB, CatB, and GBM (denoted by SeqEns; XGB needs $394.22ms$, CatB $283.74ms$ and GBM $1960.77ms$) down to $2196.68ms$ with PEMA, which means that our prediction brings a speedup of 1.21. Moreover, for DS2, we decrease analysis time from $2878.50ms$ by performing SenEns (XGB consumes $444.16ms$, CatB $285.25ms$ and GBM $2142.65ms$) down to $2279.71ms$ with our parallel ensemble learning model PEMA, which means that our prediction has a speedup of 1.26. The bar chart compares the running time of each way shown as Fig. 2.

E. Comparison

This section compares the EMBER 2017 and 2018 versions with other methods because most papers used these datasets. Our experiment results can be benchmarked with other SOTA methods in the EMBER dataset. The comparison of malware detection implementation between PEMA and SOTAs is summarized in Table IV. For the DS1, it is clear that the F1-score and accuracy of our PEMA method reach the same 99.41%, higher than the accuracy of all compared models, such as NN [25], which is 97.53% of accuracy, or AutoML [24], 99.20% of accuracy. With the DS2, the accuracy of PEMA is 97.65%; it is also higher than the accuracy of other compared models; for example, the DNN [27] has 95.57% of accuracy, the RF [29] has an accuracy of 95.51%, or Detection Pipeline [28] has

an accuracy of 96.90%. Consequently, the accuracy, precision, and speed results demonstrate that PEMA is currently the most efficient learning model.

V. Conclusions

In this study, we advocate an AI-based malware detection approach, combining several modern ML models with a weighted voting mechanism named PEMA, aimed at enhancing both accuracy and speed. The LIEF tool is employed to extract and model each PE file through 2381 attributes. After cleaning redundant attributes, we identified 2291 useful features and used them to determine whether a PE file is malware or not. PEMA is proposed by employing three classification models: XGB, CatB, and GBM. These models are chosen as the top three performers determined from experiments using the AutoML tool, namely AutoGluon. The integration of these individual models within PEMA brings several advantages, notably reducing susceptibility to poisoning and bolstering resilience against adversarial attacks. This proposed method also reduces the execution time of individual models by parallelizing the classification processes. Through rigorous testing on the EMBER 2017 and 2018 datasets, PEMA demonstrates outstanding performance, achieving F1-scores of 99.41% and 97.64%, respectively. Furthermore, PEMA's execution efficiency is improved, with the speedup of 1.21 and 1.26 for the respective datasets. Compared to other SOTA methods, our PEMA outperforms across all evaluation metrics on the same EMBER datasets.

In the near future, we anticipate expanding and enhancing PEMA to accommodate the classification of various types of malware, such as spyware, ransomware, etc. Proving the ability to be resilient against adversarial attacks will also be another perspective for future work.

References

[1] D.-O. Won, Y.-N. Jang, and S.-W. Lee, "Plausmal-gan: Plausible malware training based on generative adversarial networks for analogous zero-day malware detection," *IEEE Transactions on Emerging Topics in Computing*, vol. 11, no. 1, pp. 82–94, 2023.

[2] I. Alsmadi, B. Al-Ahmad, and M. Alsmadi, "Malware analysis and multi-label category detection issues: Ensemble-based approaches," in *2022 International Conference on Intelligent Data Science Technologies and Applications (IDSTA)*, 2022, pp. 164–169.

[3] G. V. Le, T. H. Nguyen, P. D. Pham, O. V. Phung, and H. N. Nguyen, "Guruws: A hybrid platform for detecting malicious web shells and web application vulnerabilities," *Transactions on Computational Collective Intelligence*, vol. 11370, pp. 184–208, 2019.

[4] V. Bansal, N. Baliyan, and M. Ghosh, "Dynamic android malware detection using light gradient boosting machine," in *2022 4th International Conference on Artificial Intelligence and Speech Technology (AIST)*, 2022, pp. 1–6.

[5] K. Sethi, R. Kumar, L. Sethi, P. Bera, and P. K. Patra, "A novel machine learning based malware detection and classification framework," in *2019 International Conference on Cyber Security and Protection of Digital Services (Cyber Security)*, 2019, pp. 1–4.

[6] H. Haidros Rahima Manzil and M. Naik S, "Dynamaldroid: Dynamic analysis-based detection framework for android malware using machine learning techniques," in *2022 International Conference on Knowledge Engineering and Communication Systems (ICKES)*, 2022, pp. 1–6.

[7] H. V. Vo, D. H. Nguyen, T. T. Nguyen, H. N. Nguyen, and D. V. Nguyen, "Leveraging ai-driven realtime intrusion detection by using wgan and xgboost," in *Proceedings of the 11th International Symposium on Information and Communication Technology*. New York, NY, USA: Association for Computing Machinery, 2022, p. 208–215.

[8] H. V. Vo, H. N. Nguyen, T. N. Nguyen, and H. P. Du, "Sdaid: Towards a hybrid signature and deep analysis-based intrusion detection method," in *GLOBECOM 2022 - 2022 IEEE Global Communications Conference*, 2022, pp. 2615–2620.

[9] M. A. Ferrag, L. Maglaras, S. Moschoyiannis, and H. Janicke, "Deep learning for cyber security intrusion detection: Approaches, datasets, and comparative study," *Journal of Information Security and Applications*, vol. 50, p. 102419, 2020.

[10] U. Divakarla, K. H. K. Reddy, and K. Chandrasekaran, "A novel approach towards windows malware detection system using deep neural networks," *Procedia Computer Science*, vol. 215, pp. 148–157, 2022.

[11] X. Liu, Y. Lin, H. Li, and J. Zhang, "A novel method for malware detection on ml-based visualization technique," *Computers & Security*, vol. 89, p. 101682, 2020.

[12] M. Rigaki and S. Garcia, "Stealing and evading malware classifiers and antivirus at low false positive conditions," *Computers & Security*, vol. 129, p. 103192, 2023.

[13] P. Devan and N. Khare, "An efficient xgboost–dnn-based classification model for network intrusion detection system," *Neural Computing and Applications*, vol. 32, 08 2020.

[14] M. Mimura, "Evaluation of printable character-based malicious pe file-detection method," *Internet of Things*, vol. 19, p. 100521, 2022.

[15] H. V. Vo, H. P. Du, and H. N. Nguyen, "Apelid: Enhancing real-time intrusion detection with augmented wgan and parallel ensemble learning," *Computers & Security*, vol. 136, p. 103567, 2024.

[16] H. S. Anderson and P. Roth, "EMBER: An Open Dataset for Training Static PE Malware Machine Learning Models," *ArXiv e-prints*, Apr. 2018.

[17] P. Aggarwal, S. F. Ahamed, S. Shetty, and L. J. Freeman, "Selective targeted transfer learning for malware classification," in *2021 Third IEEE International Conference on Trust, Privacy and Security in Intelligent Systems and Applications (TPS-ISA)*, 2021, pp. 114–120.

[18] N. Erickson, J. Mueller, A. Shirkov, H. Zhang, P. Larroy, M. Li, and A. Smola, "Autogluon-tabular: Robust and accurate automl for structured data," *arXiv Machine Learning*, 2020.

[19] H. V. Vo, H. P. Du, and H. N. Nguyen, "Ai-powered intrusion detection in large-scale traffic networks based on flow sensing strategy and parallel deep analysis," *Journal of Network and Computer Applications*, vol. 220, p. 103735, 2023.

[20] T. Akiba, S. Sano, T. Yanase, T. Ohta, and M. Koyama, "Optuna: A next-generation hyperparameter optimization framework," 07 2019, p. 2623–2631.

[21] J. Chen, C. Yuan, J. Li, D. Tian, R. Ma, and X. Jia, "Elamd: An ensemble learning framework for adversarial malware defense," *Journal of Information Security and Applications*, vol. 75, p. 103508, 2023.

[22] H. V. Le, T. N. Nguyen, H. N. Nguyen, and L. Le, "An efficient hybrid webshell detection method for webserver of marine transportation systems," *IEEE Transactions on Intelligent Transportation Systems*, vol. 24, no. 2, pp. 2630–2642, 2023.

[23] H. V. Le, H. V. Vo, T. N. Nguyen, H. N. Nguyen, and H. T. Du, "Towards a webshell detection approach using rule-based and deep http traffic analysis," in *Computational Collective Intelligence*, N. T. Nguyen, Y. Manolopoulos, R. Chbeir, A. Kozierkiewicz, and B. Trawiński, Eds. Cham: Springer International Publishing, 2022, pp. 571–584.

[24] A. Brown, M. Gupta, and M. Abdelsalam, "Automated machine learning for deep learning based malware detection," *arXiv Cryptography and Security*, pp. 1–16, 2023.

[25] S. Lad and A. Adamuthe, "Improved deep learning model for static pe files malware detection and classification," *International Journal of Computer Network and Information Security*, vol. 14, pp. 14–26, 04 2022.

[26] E. Raff, W. Fleming, R. Zak, H. S. Anderson, B. Finlayson, C. Nicholas, and M. McLean, "Kilograms: Very large n-grams for malware classification," *CoRR*, vol. abs/1908.00200, 2019.

[27] F. T. ALGorain and A. S. Alnaeem, "Deep learning optimisation of static malware detection with grid search and covering arrays," *Telecom*, vol. 4, no. 2, pp. 249–264, 2023.

[28] N. Loi, C. Borile, and D. Ucci, "Towards an automated pipeline for detecting and classifying malware through machine learning," *ArXiv Cryptography and Security*, pp. 1–12, 06 2021.

[29] C. Galen and R. Steele, "Evaluating performance maintenance and deterioration over time of machine learning-based malware detection models on the ember pe dataset," in *2020 Seventh International Conference on Social Networks Analysis, Management and Security*, 2020, pp. 1–7.