BULGARIAN ACADEMY OF SCIENCES

INSTITUTE OF INFORMATION AND COMMUNICATION TECHNOLOGIES

DEPARTMENT OF INFORMATION PROCESSES AND DECISION SUPPORT SYSTEMS

Iliyan Magdalenov Barzev

RESEARCH AND ANALYSIS OF THE POSSIBILITIES FOR DETECTION OF MALWARE THROUGH MACHINE LEARNING

ABSTRACT

of Dissertation
for acquiring the educational and scientific degree "Doctor"
Professional field: 4.6. "Informatics and Computer Science"
Doctoral program "Informatics"

Scientific Supervisor
Prof. Daniela Borissova, D.Sc.

2026

The dissertation has been discussed and admitted to defence at an extended meeting of the Department of Information Processes and Decision Support Systems at the IICT–BAS, held on 15.12.2025.

The dissertation is structured in an introduction, 3 chapters, conclusion, contributions, list of publications, declaration of originality of the results and bibliography. The dissertation has a total volume of 143 pages, 20 tables, 43 figures and 155 literary sources.

The dissertation defense will take place on ……………………..…..2026
from ……………………….… hours in hall …………………… of block 2 of IICT-BAS
at an open meeting of a scientific jury composed of:

Scientific jury
1. ………………………………………………………………………………………………………
2. ………………………………………………………………………………………………………
3. ………………………………………………………………………………………………………
4. ………………………………………………………………………………………………………
5. ………………………………………………………………………………………………………

The reviews and opinions of the members of the scientific jury and the abstract are published on the website of IICT-BAS.

The materials for the defense are available to those interested in room 315 of IICT-BAS, "Akad. G. Bonchev", Str., bl. 2.

Author: **Iliyan Magdalenov Barzev**

Title: **Research and analysis of the possibilities for detection of malware through Machine Learning**

INTRODUCTION

Today, malware protection is important for several reasons. One is data security, as malware can compromise personal and financial data, leading to identity theft or financial losses. The second is system health, as virus and Trojan horse infections can damage computers and networks, leading to loss of information and costly repairs. Malware can also impact productivity by causing outages that disrupt daily operations and waste time and resources. All of these situations impact a company's reputation – data compromise can affect customer trust, which is difficult to restore. Last but not least is compliance with laws and regulations, as many organizations are legally required to protect their customers' data. Violations can result in serious penalties. Malware can spread from one device to another, jeopardizing the entire network security.

Malware analysis is the process of locating and studying malicious software or code in order to understand how it works and develop countermeasures. Malware can take many forms, such as viruses, worms, Trojan horses, and ransomware, and can cause significant harm to individuals, organizations, and even entire countries. To determine the purpose, potential effects, and capabilities of a piece of malware, malware analysis involves examining the behavior, structure, and functionalities of the malware. Malware analysts are essential to the cybersecurity industry because they strive to detect threats, eliminate them, and defend against online attacks. By using the knowledge gained from malware analysis, security solutions can be created that will better protect businesses from dangerous software. Malware analysis is a key part of any successful cybersecurity strategy in today's ever-changing threat landscape.

As digital technologies evolve, the methods of carrying out attacks are also changing, which necessitates the development of new algorithms for effective and timely detection of malicious behavior. Traditional solutions, based primarily on signatures, are no longer sufficient to deal with modern threats. Historically, antivirus systems have used signature-based algorithms that compare files with a database of already known malware. This approach has several key drawbacks: It does not identify new or mutated threats; It requires constant database updates; It is ineffective against polymorphic, metamorphic, and fileless malware. The development of malware significantly outpaces the capabilities of signature systems. On the other hand, today's malware uses advanced techniques for hiding and adapting - the code changes with each execution; and attacks are executed only in memory, and new variants are automatically generated. All these factors increase the complexity of protection and require new methods of detection.

The growth of attacks is growing exponentially, with tens of thousands of new modifications appearing daily according to global research centers. There is an urgent need for proactive protection. Modern systems must detect threats before they cause harm, and this could be achieved through machine learning and behavioral analysis, which study the behavior, structure, and functionalities of malware.

Therefore, the development of new algorithms for malware detection is critical to ensuring information security in the modern digital environment. The evolution of threats requires the use of intelligent, adaptive and scalable approaches that can respond to the dynamics of modern cyberattacks. Investments in innovation and algorithmic models are key to building resilient defense systems.

Taking all these aspects into account, it becomes clear that conducting research and analysis of the possibilities for detecting malicious software using machine learning tools is a relevant and rapidly developing scientific field.

The presentation of the dissertation is structured in an introduction, three chapters, conclusion - a summary of the results obtained, contributions, list of publications, list of noted citations, declaration of originality and bibliography..

In **Chapter 1**, an analysis of various machine learning algorithms is made regarding their performance for malware detection purposes. The applicability of binary classification algorithms for malware detection is tested using a public dataset infected with 9 types of malware, using a proposed methodology. Following this methodology and the defined metrics, the results show the applicability of the studied algorithms, with one of them (Random Forest) demonstrating better performance. It is shown that the CNN-LSTM hybrid model significantly outperforms other tested approaches, achieving an accuracy of 0.97 and equally high values for precision, completeness and F1-Score. This hybrid model combines the strengths of CNN's feature extraction capabilities with the ability of LSTM to capture temporal dependencies, making it particularly effective for IoT-23 data.

In **Chapter 2**, a proposed model for virtual machine selection for malware detection experiments is described. An improved static analysis approach is presented by optimizing feature extraction, combining different machine learning algorithms for malware detection. A proposed framework for static malware classification using feature optimization and ensemble learning is presented. To improve malware classification, a self-aware framework is proposed, integrating model routing based on a trust system for feature selection and explainability. To refine malware classification, a trust-aware adaptive framework is proposed, allowing for malware classification with the possibility of corrections through feedback. This adaptive framework is implemented in a developed demo version of a software application called "Shipka Guard". It integrates a trust-aware routing mechanism that uses multiple generations of models, including a fallback mechanism. It uses a feedback buffer and correction layer for light adaptation from user corrections and synthetic injections.

Chapter 3 presents the results of the testing of the proposed models for software selection for a virtual machine. Results of testing the proposed improved static analysis approach by optimizing feature extraction, combining different machine learning algorithms, are described. Numerical experiments are described using the proposed framework for static malware classification, in which feature optimization is performed and ensemble learning is used. Results of the tested self-aware framework integrating model routing based on a trust system for feature selection and explainability are also shown. Here, the developed and tested Shipka Guard application integrating the proposed adaptive trust-based framework, allowing malware classification with the possibility of corrections through feedback, is presented. It is shown that the feedback buffer allows the user to jointly correct the model, which allows him to export/import files with feedback. On the other hand, the integration of explainability contributes to the trust in the solutions. All this has been implemented in a developed demo version of a software application called "Shipka Guard".

The conclusion summarizes the results obtained as a result of the research conducted, the subject of this dissertation work. Some main directions for future research are indicated.

CHAPTER 1. ANALYSIS OF TECHNIQUES FOR DETECTION AND CLASSIFICATION OF MALWARE

This chapter analyzes and compares various machine learning algorithms regarding their performance for malware detection purposes.

1.1. Analysis and classification of machine learning algorithms for malware detection

Machine learning (ML) as a subfield of artificial intelligence provides the ability to make predictions based on models learned directly from data, without being explicitly programmed for this (Ozkan-Okay et al., 2024). ML plays a key role in detecting and mitigating complex malware threats in various ways. ML can analyze large volumes of data, classify files and programs, predict threats, etc. (Fig. 1.1).

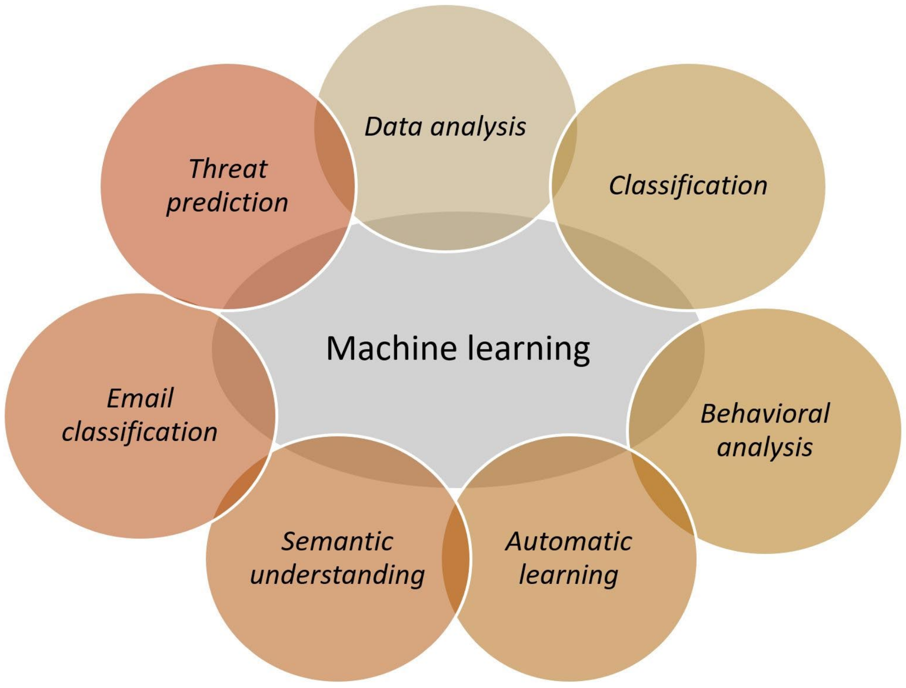


Fig. 1.1. Application of machine learning

1.2. Comparison of different binary classification algorithms

For the purpose of comparing the applicability and accuracy of different malware detection algorithms on the same data volume, the following diagram is used, as shown in Fig. 1.2 (Barzev et al., 2024).

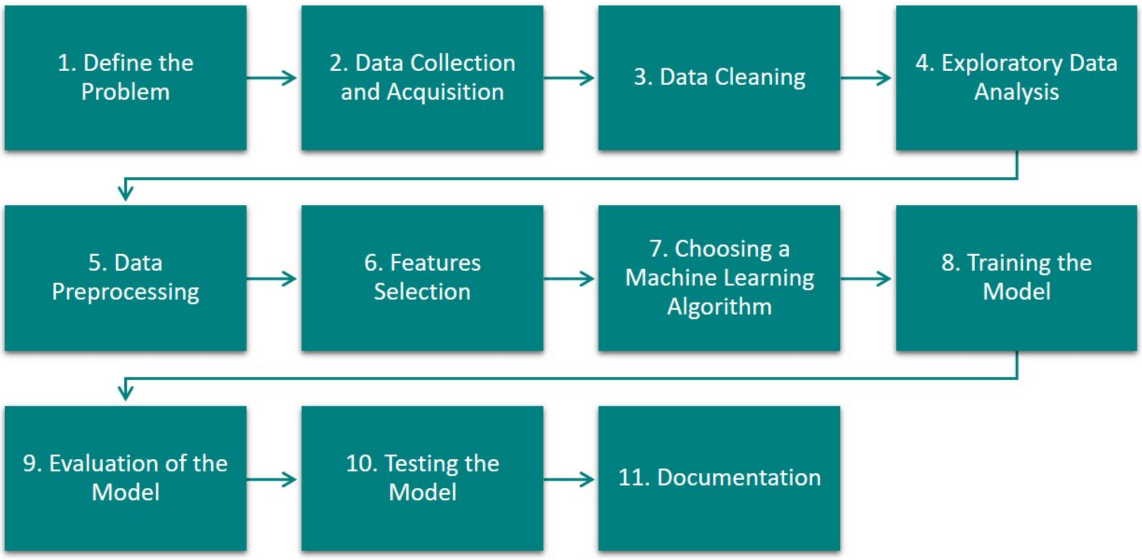


Fig. 1.2. Flowchart of the experiment methodology

The dataset used in this experiment is the BIG2015 Dataset. It contains labeled benign and malicious PE and OLE files. The distribution of malware in the dataset is shown in Fig. 1.3.

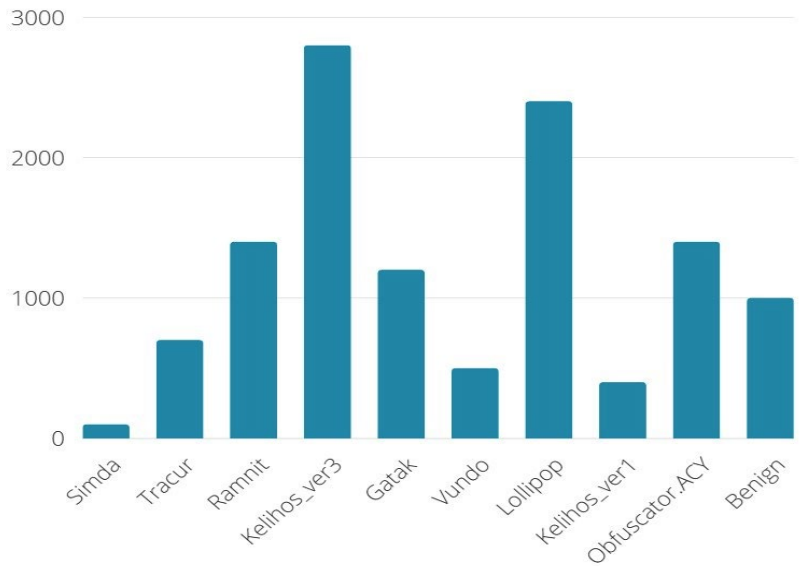


Fig. 1.3. Representation of malware distribution in the dataset

Before getting to the result of the experiment, it is important to define the metrics we will focus on to determine how well the algorithms perform against the dataset. The selected metrics are: 1) Accuracy, 2) Precision), 3) Recall and 4) F-1 score.

The obtained values of the studied metrics for the selected algorithms Random Forest, K-Nearest Neighbors and Support Vector Machines on the studied dataset are shown in Table 1.2.

Table 1.2. Results of the metrics used

Algorithm	**Metrics**			
	Accuracy	**Precision**	**Recall**	**F-1**
RF (Random Forest)	0.9825	0.9812	0.9999	0.9893
K-NN (K-Nearest Neighbors)	0.8632	0.8911	0.9938	0.9731
SVM (Support Vector Machines)	0.7819	0.9653	0.7309	0.8487

From Table 1.2 it can be seen that the best algorithm for the metric used is Random Forest. This is due to the accuracy of Random Forest, which is the best with a value of 0.9825, followed by K-NN with a value of 0.8632 and the last is Support Vector Machines with 0.7819. Regarding the second indicator – precision, the results are similar and the best performance is shown by the Random Forest algorithm, also with a value of 0.9812, followed by Support Vector Machines and finally by K-NN. This is due to the results obtained for the selected metrics. For the remaining metrics, the best results are obtained by the Random Forest algorithm (Table 1.2).

1.3. Performance analysis of malware detection algorithms in an Internet of Things dataset

One of the critical challenges associated with malware infiltration is gaining unauthorized access to IoT devices, where malware attempts to copy authorized devices by mimicking their hardware and software specifications (Praveen et al., 2023).

Network traffic data from a sample of the IoT-23 dataset was used to analyze the performance of the LSTM, SVM, CNN, and CNN-LSTM algorithms for malware detection. The corresponding confusion matrices visualizing and summarizing the performance of the classification algorithms are as follows Fig. 1.4 to Fig. 1.7.

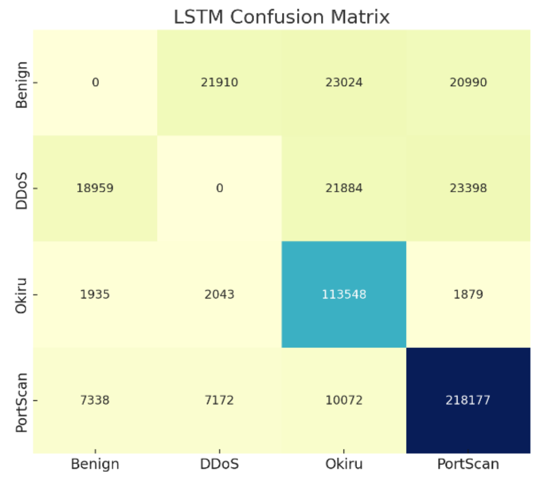


Fig. 1.4. LSTM confusion matrix

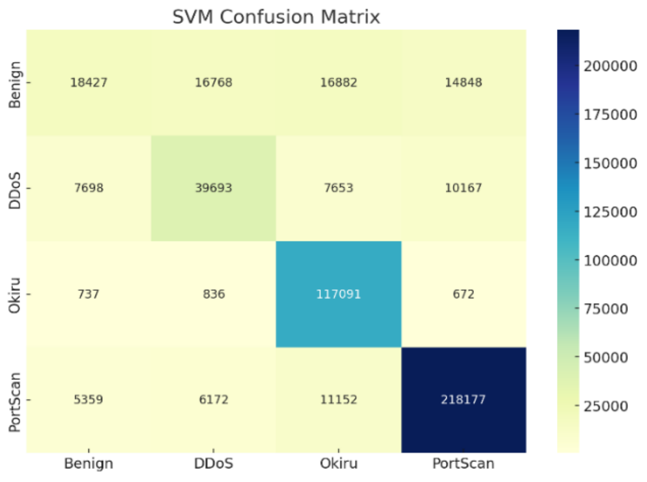
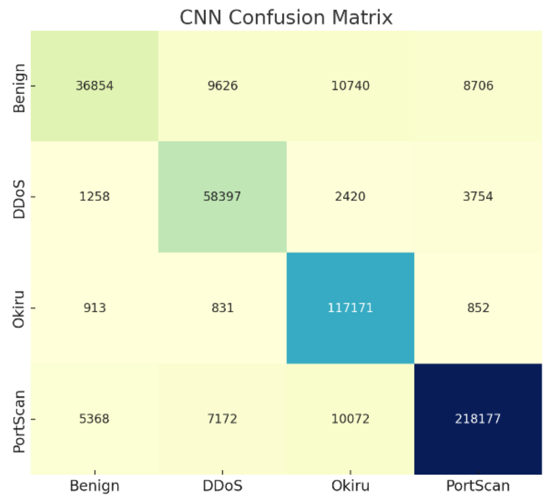


Fig. 1.5. SVM confusion matrix



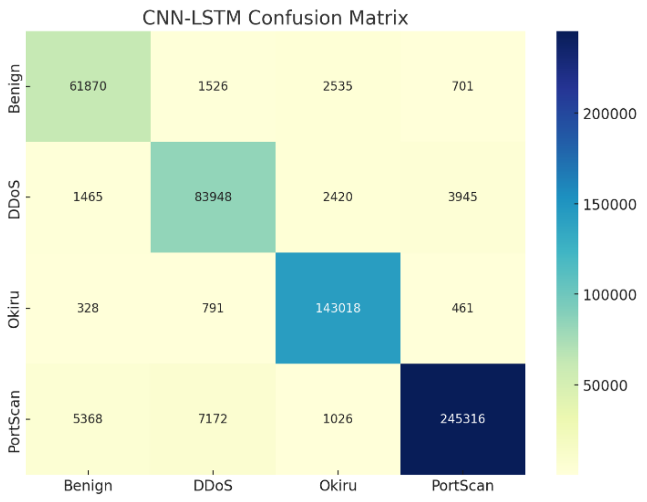


Fig. 1.6. CNN confusion matrix Fig. 1.7. CNN-LSTM confusion matrix

To evaluate the performance of the four classification algorithms discussed above, LSTM, SVM, CNN and CNN-LSTM, Receiver Operating Characteristic (ROC) curve analysis was used. ROC curves graphically illustrate the discriminatory capabilities of each model by reflecting the true positive rate (TPR) versus the false positive rate (FPR) for different threshold values, as shown in Fig. 1.8.

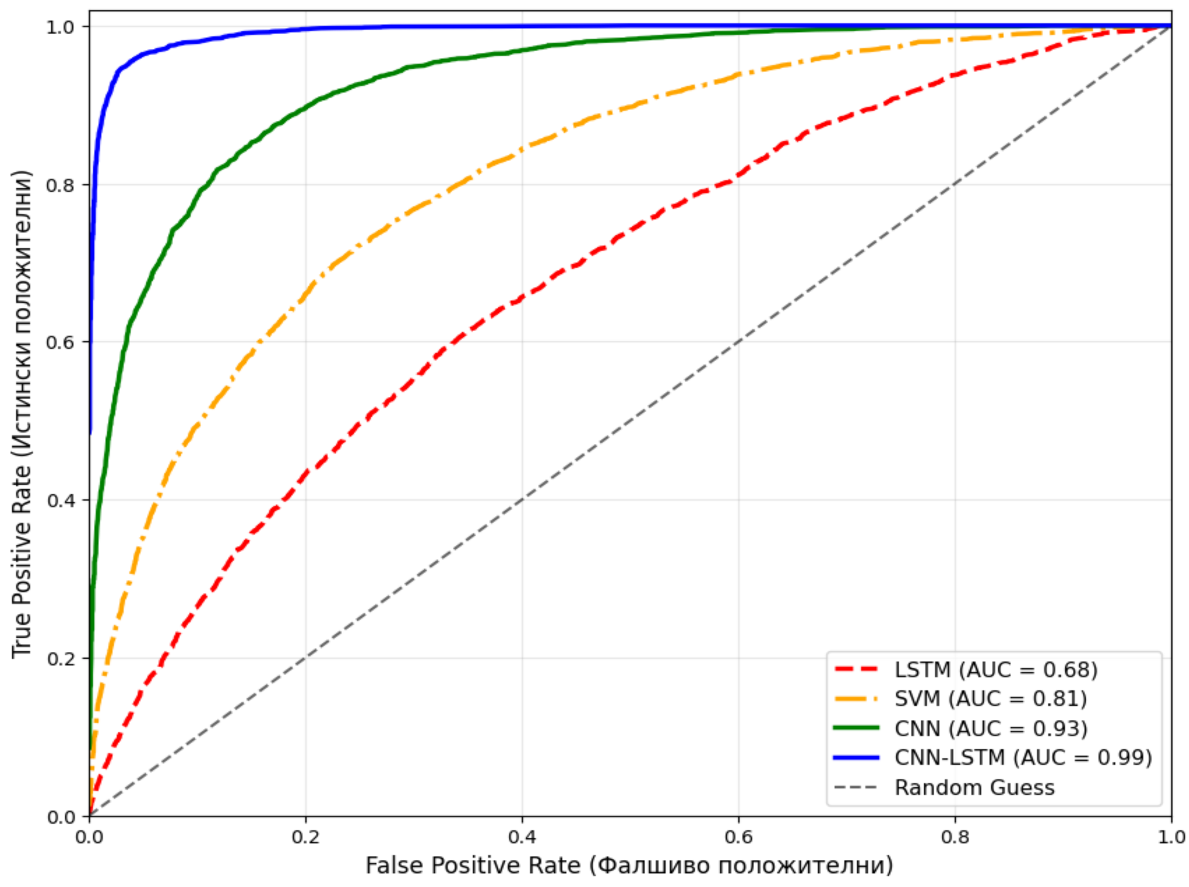


Fig. 1.8. ROC curves for LSTM, SVM, CNN, CNN-LSTM

Each curve in this analysis shows how well a given model (LSTM, SVM, CNN, or CNN-LSTM) distinguishes between classes (Benign, DDoS, Okiru, PortScan). A higher curve with a larger AUC suggests better performance in classifying cases for that particular class. The shape and location of each curve reflect the trade-offs between correctly identifying positive cases and avoiding false positives.

To comprehensively evaluate the classification performance of the four models – LSTM, SVM, CNN and CNN-LSTM – several key metrics were analyzed, including accuracy, precision, repetition completeness and F1 score, as shown in Table 1.3.

Table 1.3. Algorithm performance results

Model	Accuracy	Precision	Recall	F1-Score
LSTM	0.74	0.72	0.74	0.71
SVM	0.84	0.86	0.84	0.84
CNN	0.91	0.92	0.91	0.91
CNN-LSTM	0.97	0.97	0.97	0.97

1.3. Conclusions

Machine learning, as a field of artificial intelligence, uses algorithms to learn from data, identify patterns, and make predictions or decisions with minimal human intervention. Therefore, some specific conclusions can be drawn from the analyses and comparisons conducted, namely.

- There are a variety of machine learning-based models that are capable of detecting malicious software;
- An important stage in the application of machine learning, and in particular supervised learning, is the process of training itself, which aims to train with data for a specific subject area;
- Combining different machine learning models can contribute to overcoming the shortcomings of models used independently..

Therefore, the prerequisites listed above are the basis for developing improved models, frameworks, and applications leading to better results in malware detection and classification.

1.4. Purpose and objectives

The aim of the dissertation is to investigate and analyze the possibilities for detecting malicious software using machine learning tools, based on which to propose suitable hybrid models, frameworks and applications for obtaining better results in detecting malicious software. To achieve this goal, it is necessary to complete the following tasks:

1) to analyze different machine learning algorithms regarding their performance for malware detection purposes;
2) to determine a suitable virtual machine to use when conducting malware detection and classification tests;
3) to propose an improved static analysis approach for malware detection by optimizing feature extraction and combining different machine learning algorithms;
4) to propose a framework for static malware classification using feature optimization and ensemble learning;
5) to propose self-aware malware classification by routing models based on a trust system for feature selection and explainability;
6) to propose an adaptive, trust-aware framework for malware classification with feedback corrections.

CHAPTER 2. VIRTUAL MACHINE SELECTION MODELS, HYBRID ALGORITHMS, AND MALWARE DETECTION FRAMEWORKS USING MACHINE LEARNING METHODS

2.1. Choosing virtual machine software for malware detection purposes

Considering the importance of the virtual machine not only for business purposes but also for malware detection, two group decision-making models for virtual machine selection have been proposed.

2.1.1 Group decision-making models for virtual machine software selection

A specific problem is considered, concerning the selection of a virtual machine to run on a Windows desktop. The criteria “ease of use”, “customer service” and “price-quality ratio” can be used as sufficiently general indicators, which can be easily understood. All these criteria, together with the already given ratings, are used in the following proposed model for group decision-making (Borissova et al., 2023):

$$\max\{A_j^*\} = \sum_{q=1}^{Q} \lambda^q \sum_{i=1}^{I} w_i \, e_i^q \tag{2.1}$$

$$\sum_{i=1}^{I} w_i = 1 \tag{2.2}$$

$$\sum_{q=1}^{Q} \lambda^q = 1 \tag{2.3}$$

where index j is used to denote a set of given alternatives ($j = 1, \dots, J$), with index i the set of evaluation criteria is indicated ($i = 1, \dots, I$), and index q is used to denote the set of decision makers ($q = 1, \dots, Q$) which will form the final group decision. The coefficients w_i express the importance of the evaluation criteria and must comply with the relation (2.2). The other type of coefficients e_i^q express the evaluations of the *i-th* criterion according to the point of view of *q-th* expert. Depending on the competence of each expert, corresponding weights λ^q are assigned to each, which will participate in the formation of the final group decision.

It is also possible, instead of selecting a single type of virtual machine software, to reduce the given set of alternatives to a subset, and to make the selection according to other criteria. For example, a subsequent ranking can be made using the specific characteristics. In this situation, the following combinatorial optimization model can be used (Borissova et al., 2023):

$$max\left\{\sum_{j=1}^{J} x_j \sum_{q=1}^{Q} \lambda^q \sum_{i=1}^{I} w_i \, e_i^q\right\} \tag{2.4}$$

Subject to

$$\sum_{j=1}^{J} x_j = C, x_j \in \{0,1\} \tag{2.5}$$

$$x_j \in \{0,1\} - binary\ integers \tag{2.6}$$

$$C < J \tag{2.7}$$

$$\sum_{i=1}^{I} w_i = 1 \tag{2.8}$$

$$\sum_{q=1}^{Q} \lambda^q = 1 \qquad (2.9)$$

The binary variables x_j used are associated with each alternative, and the constant C expresses the number of alternatives to which we aim to reduce the given set of alternatives. It is obvious that this constant must be smaller than the number of alternatives expressed by (2.7). The objective function (2.4) will select exactly those alternatives with better results, according to the e_i^q estimates.

2.2. Improved static analysis approach by optimizing feature extraction, combining different machine learning algorithms for malware detection

Static malware analysis, as a proactive approach, involves analyzing the characteristics and components of a suspicious file without executing it. This method provides information about the structure, behavior, and potential risks of the file by analyzing various attributes, such as the file header, metadata, strings, resources, and code. This safe static approach features fast scanning and flagging of common threats, making it ideal for antivirus software and initial triage.

2.2.1. Improved malware detection approach by combining different machine learning algorithms

The generalized algorithmic representation of the proposed approach for improved malware detection by combining different machine learning algorithms into 3 hybrid algorithms is illustrated in Fig. 2.2 (Barzev & Borissova, 2025).

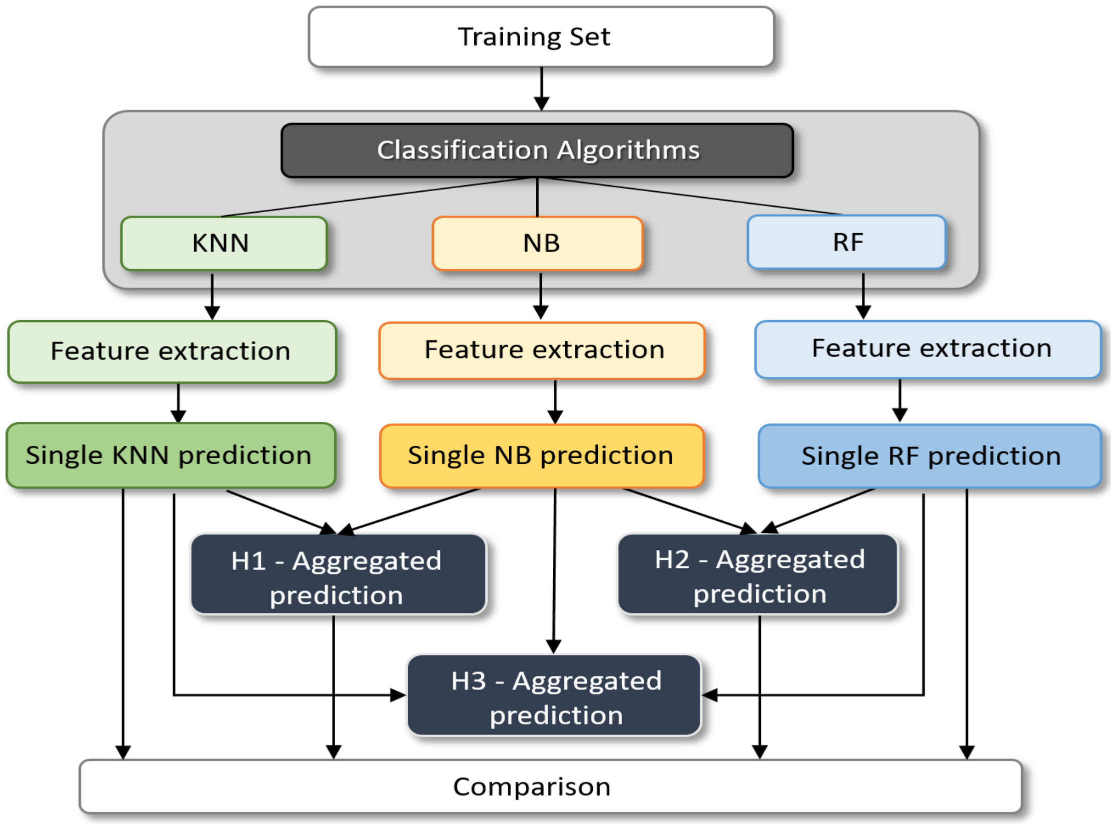


Fig. 2.2. Generalized algorithmic representation of the proposed approach

In the initial phase, the individual accuracies of the classifiers are compared, and in the next phase, the accuracy of the formed hybrid algorithms is compared.:

1) H1 as a combination of NB and KNN;

2) H2 as a combination of NB and RF;

3) H3 as a combination of NB, KNN and RF.

NB as a direct probabilistic classifier works under the assumption of strong feature independence. It relies on Bayes' theorem to calculate the prior probability $P(H)$ and the posterior probability $P(H|E)$ for an event observed before and after the evidence, respectively, and is expressed as Bayes' rule:

$$P\,(H|E) \;=\; (P(E|H)\;\mathrm{x}\;P(H))/P(E)$$

The pseudocode for the naive Bayes algorithm is as follows:

```
Step 1: Training:
      For each feature
         For each feature-value E
              For each class-label H
              P(E/H) = total number of occurrences of feature value with class label)/(total number of occurrences of class label)
              End for
         End for
      End for
Step 2: Testing:
      For each instance in test data
         Calculate probability using P(H/E)=(P(E/H)*P(H))/P(E)
      End for
Step 3: Assign the class label with the maximum probability to the test instance.
```

Similarly, the KNN and RF algorithms can be expressed

After the feature selection process, the number of features identified in the two datasets is thirteen and fifteen, respectively. The final result contains a list of 7 important features, as shown in Table 2.1.

Table 2.1. Features used by the XGB model, with their description

No	Feature	Description
1	conn_state_50	Connection state
2	proto_tcp	Transaction protocol
3	idresp_p	Destination port
4	idresp_h	Destination IP address
5	resp_pkts	Destination packets
6	orig_ip_bytes	Source2destination transaction bytes
7	Idorig_p	Source port

An important condition for achieving better accuracy when combining classifiers is the use of different feature sets or different training sets (Ho et al., 1994; Kittler et al., 1998). An essential feature of a multilevel classifier is the fact that it can stabilize the training of classifiers based on a small sample size.

The pseudocode of the hybrid algorithm H1 based on the aggregation of NB and KNN is as follows:

```
Step 1: Aggregate the probabilities from NB and KNN
  For each test instance Ti
    For each class label Ci
      Probability of Ci= (probability of Ci obtained from NB + probability of Ci
obtained from KNN)/2
    End for
  End for
Step 2: Assign the class label with highest probability to the test instance.
```

The pseudocode of the hybrid algorithm H2, based on the aggregation of NB and RF, is as follows:

```
Step 1: Aggregate the probabilities from NB and RF
  For each test instance Ti
    For each class label Ci
    Probability of Ci= (probability of Ci obtained from NB + probability of Ci
obtained from RF)/2
    End for
  End for
Step 2: Assign the class label with highest probability to the test instance.
```

The pseudocode of the hybrid algorithm H3, based on the aggregation of NB, RF and KNN, is as follows:

```
Step 1: Aggregate the probabilities from NB, RF and KNN
  For each test instance Ti
    For each class label Ci
    Probability of Ci= (probability of Ci obtained from NB + probability of Ci
obtained from RF + probability of Ci obtained from KNN)/3
    End for
  End for
Step 2: Assign the class label with highest probability value to the test
instance.
```

It is proposed that the weighted average accuracy when combining results from different machine learning algorithms be expressed by the following ratio:

$$Conbined\_Accuracy = (\sum_{i=1}^{n} Accuracy_i * Count_i)(\sum_{i}^{n} Count_i)^{-1} \qquad (2.10)$$

where $Accuracy_i$ is the accuracy of the ith algorithm while $Count_i$ represents the number of times the ith accuracy was achieved (i.e. the number of algorithms that led to the ith accuracy), and n is the total number of unique accuracies.

2.3. A framework for static malware classification using feature optimization and ensemble learning

2.3.1. Methodology of a Python-based software framework for static malware classification using feature optimization and ensemble learning

A methodology is proposed to implement a framework for malware detection and classification using Python libraries. This framework combines several ML algorithms, such as Random Forest, XGBoost, LightGBM, and CatBoost (Thai, 2022; Hancock & Khoshgoftaar, 2020). The proposed framework relies on a modular architecture and can easily add additional modules with different ML algorithms. In developing a Python-based framework for automatic malware classification, the following methodology is proposed, illustrated in Fig. 2.6.

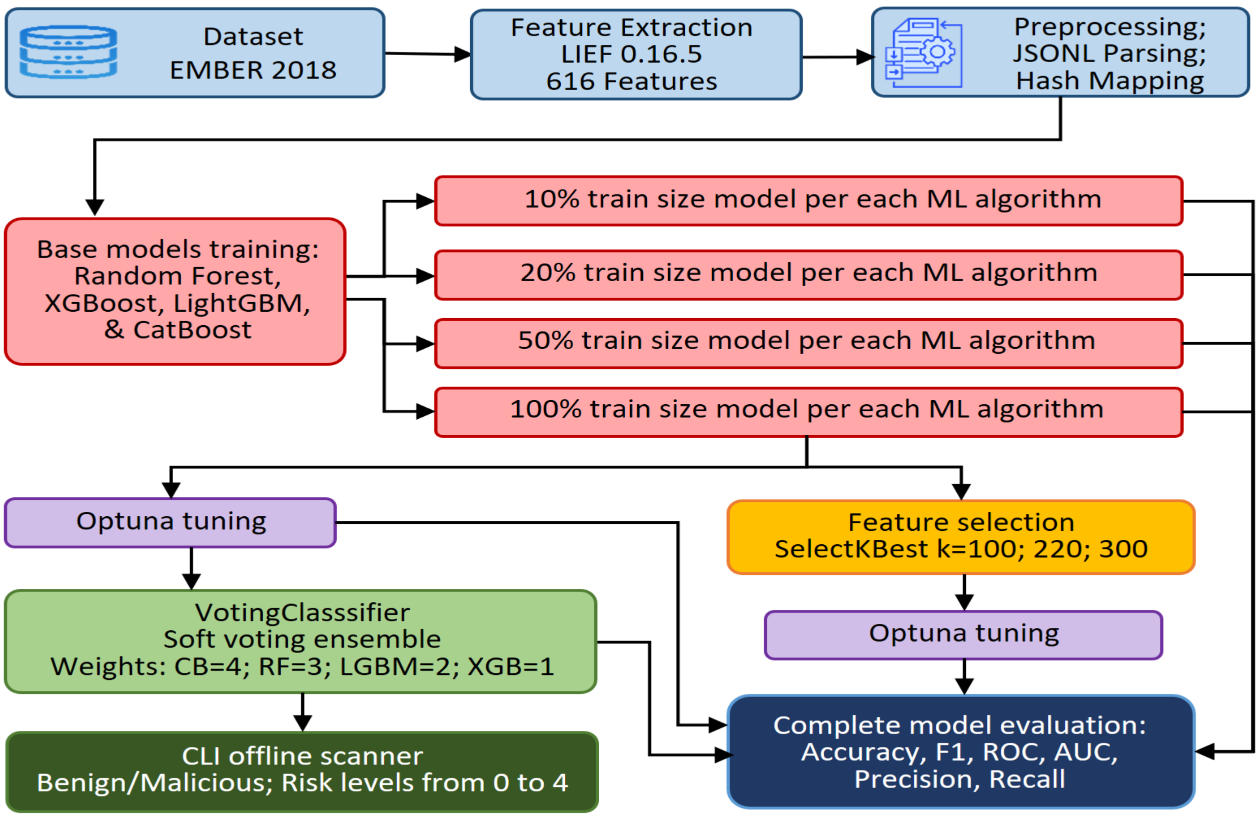


Fig. 2.6. Methodology of the developed Python-based framework for static malware classification

2.3.2. Model training and results

For each experiment, four models are trained for each algorithm (one for each training set), resulting in 16 baseline models. The evaluation metrics of these 45 trained models are shown in Table 2.2.

Table 2.2. Results of the models with different training datasets.

#	Model Name	K	Accuracy	F1-Score	ROC AUC	Precision	Recall	Optuna
	100% Train Size							
1	RandomForest	616	0.9391	0.9392	0.9796	0.9353	0.9433	Yes
2	RandomForest	616	0.9245	0.9241	0.9542	0.9285	0.9198	No
3	RandomForest	300	0.9117	0.9116	0.9426	0.9129	0.9093	No
4	RandomForest	300	0.9362	0.9369	0.9777	0.9332	0.9387	Yes
5	RandomForest	200	0.9371	0.9371	0.962	0.9344	0.9394	Yes
6	RandomForest	200	0.9188	0.9184	0.9435	0.9193	0.9175	No
7	RandomForest	100	0.9357	0.9356	0.9506	0.9328	0.9382	Yes
8	RandomForest	100	0.9154	0.9154	0.9454	0.9161	0.9145	No
9	XGBoost	616	0.9173	0.9184	0.9728	0.9063	0.9308	Yes
10	XGBoost	616	0.9082	0.9171	0.9482	0.8926	0.9281	No
11	XGBoost	300	0.9283	0.9287	0.9586	0.9207	0.9368	Yes
12	XGBoost	300	0.7762	0.8027	0.9124	0.7176	0.9106	No
13	XGBoost	200	0.9268	0.9269	0.9735	0.9201	0.9336	Yes
14	XGBoost	200	0.7604	0.7781	0.9007	0.7246	0.8412	No
14	XGBoost	100	0.9255	0.9258	0.9407	0.9191	0.9324	Yes
15	XGBoost	100	0.8294	0.8215	0.8394	0.8616	0.7852	No
16	LightGBM	616	0.9389	0.9391	0.9691	0.9351	0.9432	Yes
17	LightGBM	300	0.9441	0.9438	0.9853	0.9445	0.9437	Yes
18	LightGBM	200	0.9424	0.9425	0.9841	0.9399	0.9451	Yes
19	LightGBM	100	0.9427	0.9425	0.9576	0.9432	0.9418	Yes
20	LightGBM	200	0.9011	0.9036	0.9283	0.8815	0.9268	No
21	LightGBM	616	0.8970	0.8993	0.9645	0.8795	0.9201	No
22	LightGBM	300	0.8257	0.8408	0.9232	0.7739	0.9203	No
23	CatBoost	616	0.9405	0.9406	0.9706	0.9388	0.9424	Yes
24	CatBoost	616	0.9292	0.9298	0.9597	0.9215	0.9382	No
25	CatBoost	300	0.9341	0.9342	0.9757	0.9299	0.9385	Yes
26	CatBoost	300	0.8447	0.8503	0.8795	0.8206	0.8823	No
27	CatBoost	200	0.9355	0.9359	0.9608	0.9307	0.9411	Yes
28	CatBoost	200	0.9237	0.9236	0.9689	0.9246	0.9226	No
29	CatBoost	100	0.9315	0.9313	0.9464	0.9275	0.9351	Yes
30	CatBoost	100	0.8507	0.8517	0.8665	0.8457	0.8578	No
31	VotingClassifier (XGB, LGBM, CB, All Features)	616	0.9461	0.9461	0.9864	0.9468	0.9454	No
	50% Train Size							
32	RandomForest	616	0.8652	0.8533	0.9253	0.9338	0.7856	No
33	XGBoost	616	0.8668	0.8621	0.9216	0.8934	0.8329	No
34	LightGBM	616	0.8659	0.8605	0.9352	0.8967	0.8272	No
35	LightGBM	100	0.8827	0.8841	0.8989	0.8735	0.8951	No
36	CatBoost	616	0.8891	0.8846	0.9543	0.9223	0.8498	No

	20% Train Size							
37	RandomForest	616	0.7855	0.7375	0.9247	0.9497	0.6028	No
38	XGBoost	616	0.8381	0.8244	0.9256	0.9003	0.7603	No
39	LightGBM	616	0.8349	0.8141	0.9244	0.9314	0.7234	No
40	CatBoost	616	0.8625	0.8487	0.9408	0.9433	0.7713	No
	10% Train Size							
41	RandomForest	616	0.5972	0.3336	0.7517	0.9663	0.2016	No
42	XGBoost	616	0.6626	0.5165	0.8121	0.9112	0.3604	No
43	LightGBM	616	0.7463	0.6683	0.9034	0.9658	0.5112	No
44	CatBoost	616	0.6458	0.4664	0.7613	0.9458	0.3096	No

2.3.3. Feature selection and hyperparameter optimization.

Two important methods are applied to optimize and systematize performance: dimensionality reduction using SelectKBest and hyperparameter optimization using Optuna. A total of 45 models are trained and evaluated with the following combinations: 4 machine learning algorithms (Random Forest, XGBoost, LightGBM, CatBoost), 4 different training set sizes (10%, 20%, 50%, 100%), Optuna on and off, and different values of SelectKBest, k = 100; 200; 300; each model is evaluated by an independent test set on 5 main indicators (see Table 2.2).

The SelectKBest feature selection algorithm using mutual_info_classif is used to identify the most informative features. When comparing the number of features and performance, the parameter k is tested with 100, 200, and 300 features, respectively. There is a small loss of accuracy due to fewer features selected, but both k=200 and k=300 are not very different from the full feature set, meaning that fewer features can be useful in situations where computational resources are a factor. Full optimization of different models with Optuna requires a significant amount of computational and time resources: each of the 4 algorithms (CatBoost, XGBoost, LightGBM, Random Forest) ran 50 trials plus different feature dimensions and model depths, which took approximately 100 to 140 hours of pure computational time.

2.3.4. Optimized hyperparameters from the best performing configurations

The hyperparameters were optimized based on the 50 Optuna runs performed on each model. They are the best configurations based on the F1-score of the validation set, and the final model comparisons used these values. The described hyperparameters are well-balanced measures of performance/speed and were validated using the full feature set (616 features) with SelectKBest transformations (k=100;200;300).

2.3.5. Ensemble model with voting classifier (Voting Classifier)

An ensemble model was created that combined optimized models with unique architectures, giving the most favorable result in the analysis. The VotingClassifier model, which implements soft voting with certain weights, made predictions stronger than individual models, including CatBoost, which had the best performance. This is a testament to harnessing the power of unique models with unique useful algorithms.

The composition of VotingClassifier is as follows: CatBoost (Optuna, 616 features); LightGBM (Optuna, 616 features); Random Forest (Optuna, 616 features); and XGBoost (Optuna, 616 features). Soft Voting is used to pool probabilistic results from the model. Soft Voting is preferred over hard voting because it allows for pooling predictions in a much more informative way, as predicted probabilities can be pooled and examined, rather than just pooling and hard-coded classifications.

2.4. Self-aware malware classification by routing models based on a trust system for feature selection and explainability

2.4.1. Dataset and feature extraction

EMBER 2024 is a large-scale open dataset for static malware detection developed by Future Computing Lab and published in early 2024. EMBER 2024 is the official successor to the EMBER 2018 benchmark dataset. EMBER 2024 includes a total of over 5 million examples, across many labels, including:

- 4 000 000 training examples (Win32, .NET and unknown PEs)
- 1 000 000 test examples (Win32 and other PEs)
- Multiple class/architecture types, including .NET-specific and Authenticode-signed files

Feature extraction was performed using a modified PEFeatureExtractor pipeline with lief==0.16.5 and Python 3.8 to ensure backward compatibility with older PE formats. Each sample is extracted into a 616-dimensional vector that includes byte histograms, entropy parts, and string features. Table 2.3 illustrates the main differences between the EMBER 2018 and EMBER 2024 datasets.

Table 2.3. EMBER 2018 and EMBER 2024 datasets

Attribute	EMBER 2018	EMBER 2024 (our sample)
Release year	2018	2024
Training samples	900 000	1 560 000
Test samples	200 000	360 000
Feature count	616	616 (harmonized)

The recorded best parameters of each model were evaluated using various metrics (accuracy, F1-score, ROC-AUC and precision, completeness), presented in Table 2.4:

Table 2.4. Performance comparison of individual classifiers optimized for Optuna variants and the VotingClassifier ensemble on EMBER 2024 (Win32)

Model	Accuracy	F-1 score	ROC-AUC	Precision	Recall
CatBoost	0.9397	0.9382	0.9881	0.9622	0.9153
LightGBM	0.9438	0.9425	0.9898	0.9650	0.9210
LightGBM (Optuna)	0.9540	0.9531	0.9925	0.9710	0.9359
RandomForest	0.9545	0.9535	0.9927	0.9746	0.9333
CatBoost (Optuna)	0.9546	0.9538	0.9924	0.9710	0.9372
RandomForest (Optuna)	0.9552	0.9542	0.9928	0.9747	0.9346
XGBoost	0.9575	0.9568	0.9936	0.9730	0.9411
XGBoost (Optuna)	0.9664	0.9659	0.9955	0.9795	0.9527
VotingClassifier (All models)	0.9654	0.9649	0.9953	0.9793	0.9509

Ensemble model: VotingClassifier

An ensemble model was built using two combinations of the optimized classifiers via VotingClassifier, with soft voting and weights for each classifier depending on its reliability.

(1) First combination: XGBoost: 10; RandomForest: 2; CatBoost: 1; LightGBM: 4

(2) Second combination: XGBoost: 15; RandomForest: 1; LightGBM: 3

Although we tried to build a robust VotingClassifier algorithm, the optimized XGBoost algorithm outperforms the other algorithms in terms of generalization and is the main classifier in the SAMC framework.

Self-aware model classifier

A self-aware model classifier (SAMC) is developed to dynamically guide to the appropriate model based on the level of confidence of the prediction. The goal of the SAMC logic is to find a balance between confidence, generalizability, and robustness when using old and modern machine learning models, with both models trained on two different datasets: EMBER 2018 (old) and EMBER 2024 (new).

Each .exe file is first statically parsed through the PEFeatureExtractor pipeline, resulting in a 616-dimensional feature vector. The vector is then fed into the following two classifiers, which are trained independently:

- Legacy model: VotingClassifier trained on EMBER 2018, which is composed of the RandomForest, CatBoost, LightGBM, and XGBoost models with soft voting (weights: RF=3, CB=2, LGBM=1, XGB=4), achieving the following best performance on legacy data: Accuracy 0.9646 0.9461, F1-Score 0.9646 0.9461, ROC-AUC 0.9951 0.9864, Precision 9468 , Recall 0.9454
- New model: an XGBoost (Optuna) classifier trained on 2024 Win32 samples from EMBER. It also achieves all the best performance: Accuracy 0.9583 0.9664, F1-Score 0.9576 0.9659, and ROC-AUC 0.9938 0.9527, Precision 0.9795, Recall 9527.

Let maxproba denote the maximum predicted class probability of a given model. SAMC can represent an equation that determines routing based on the following logic:

- If max_proba(new_model) ≥ 0.85: trust the prediction of the new model,
- If max_proba(legacy_model) ≥ 0.85: trust the prediction of the old model,
- Otherwise: combine the predicted class probabilities of both models as a fallback, proposing a weighted average as follows:

$$P_{final} = 0.6 \cdot P_{new} + 0.4 \cdot P_{legacy}$$

The highest probability of this extreme probability will be selected as the final result taken by SAMC. The confidence-based routing mechanism within SAMC ensures that predictions will only be accepted when both individual models are sufficiently confident, limiting the likelihood of accepting overly confident misclassifications of new or ambiguous new data. The routing logic of SAMC is illustrated in Fig. 2.7:

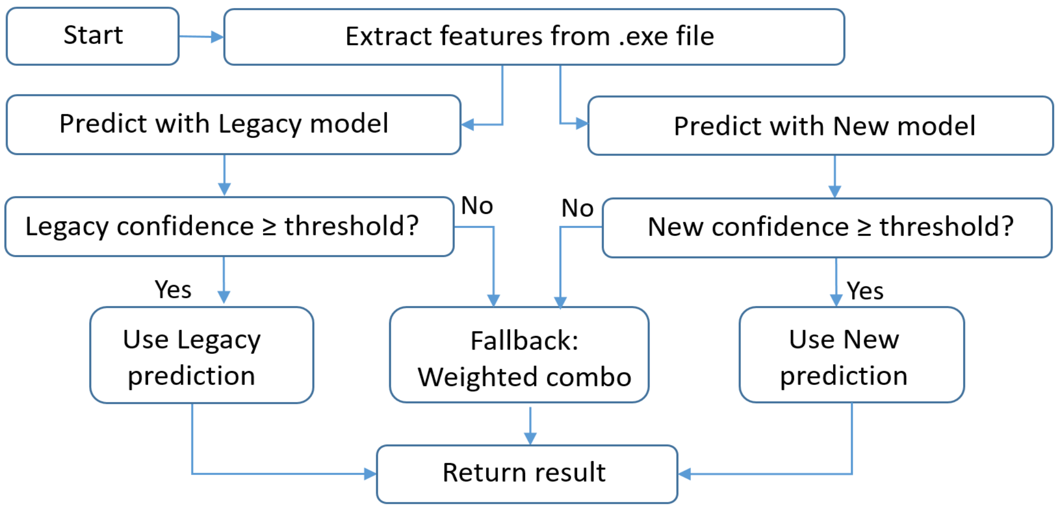

Start
Extract features from .exe file
Predict with Legacy model
Predict with New model
Legacy confidence ≥ threshold?
No
No
New confidence ≥ threshold?
Yes
Yes
Use Legacy prediction
Fallback: Weighted combo
Use New prediction
Return result

Fig. 2.7. SAMC routing logic with trust-based selection and weighted fallback strategy

As a result of the study, a statistically significant deviation was found for many characteristics. Table 2.6 summarizes the results for the 10 characteristics with the highest KS test statistics:

Table 2.6. Top 10 features with KS test statistics

Feature Index	KS Statistic	p-value
Feature 501	0.3983	< 1e-10
Feature 507	0.3976	< 1e-10
Feature 506	0.3971	< 1e-10
Feature 499	0.3959	< 1e-10
Feature 508	0.3959	< 1e-10
Feature 511	0.3959	< 1e-10
Feature 510	0.3958	< 1e-10
Feature 498	0.3950	< 1e-10
Feature 505	0.3945	< 1e-10
Feature 509	0.3943	< 1e-10

2.4.3. Self-aware malware classification scanner with graphical user interface

To improve the transparency, accessibility and explainability of the proposed malware classification models, we need to create a graphical user interface. It should allow the user to scan an executable file using three selection modes: (1) Manual Legacy Mode: uses the original model trained on EMBER 2018; (2) Manual New Model Mode: uses the optimized model trained on EMBER 2024; (3) Auto Mode (SAMC): performs intelligent model selection based on confidence thresholds.

Fig. 2.16 shows an illustration of the integrated routing flow as planned to be implemented through the graphical user interface.

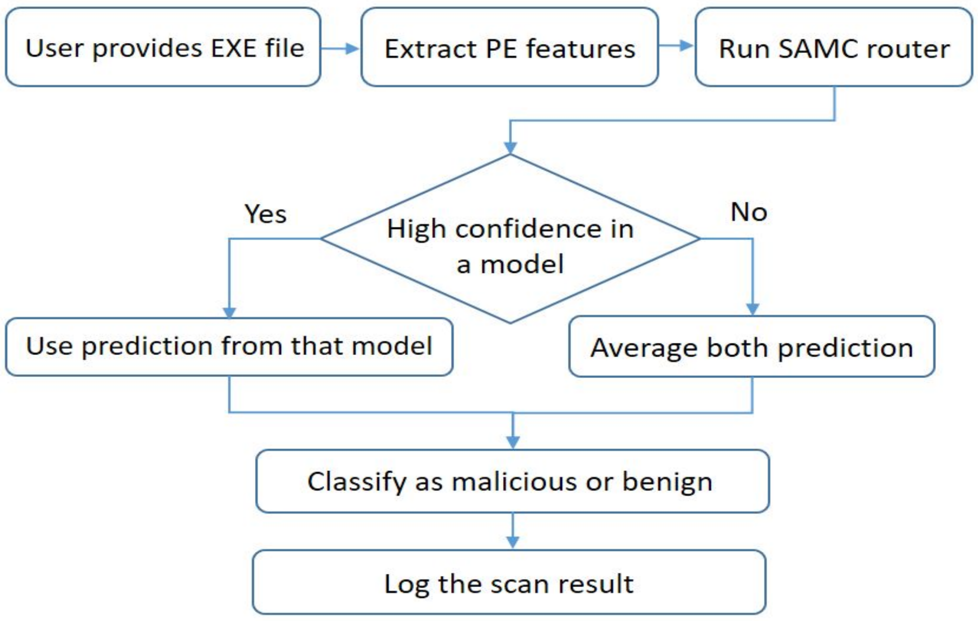


Fig. 2.16. Scanning and routing using a graphical user interface, including trust-based fallback logic.

Once the user selects a file, they enter SAMC after the static features have been extracted. SAMC determines which model (old or new) should be used based on how confident any of the predictions were or, if any of the predictions did not present a significant level of confidence, what weight was given to the fallback. The system will inform of the decision and provide a SHAP-based explanation if the new model was used.

2.5. An adaptive framework integrating trust for malware classification through feedback corrections

2.5.1. System design and implementation

The adaptive feedback malware classification framework, Shipka Guard, is implemented by integrating legacy and modern models with a feedback-based adaptive layer. The system is organized into several interconnected components, which are illustrated in Fig. 2.17.

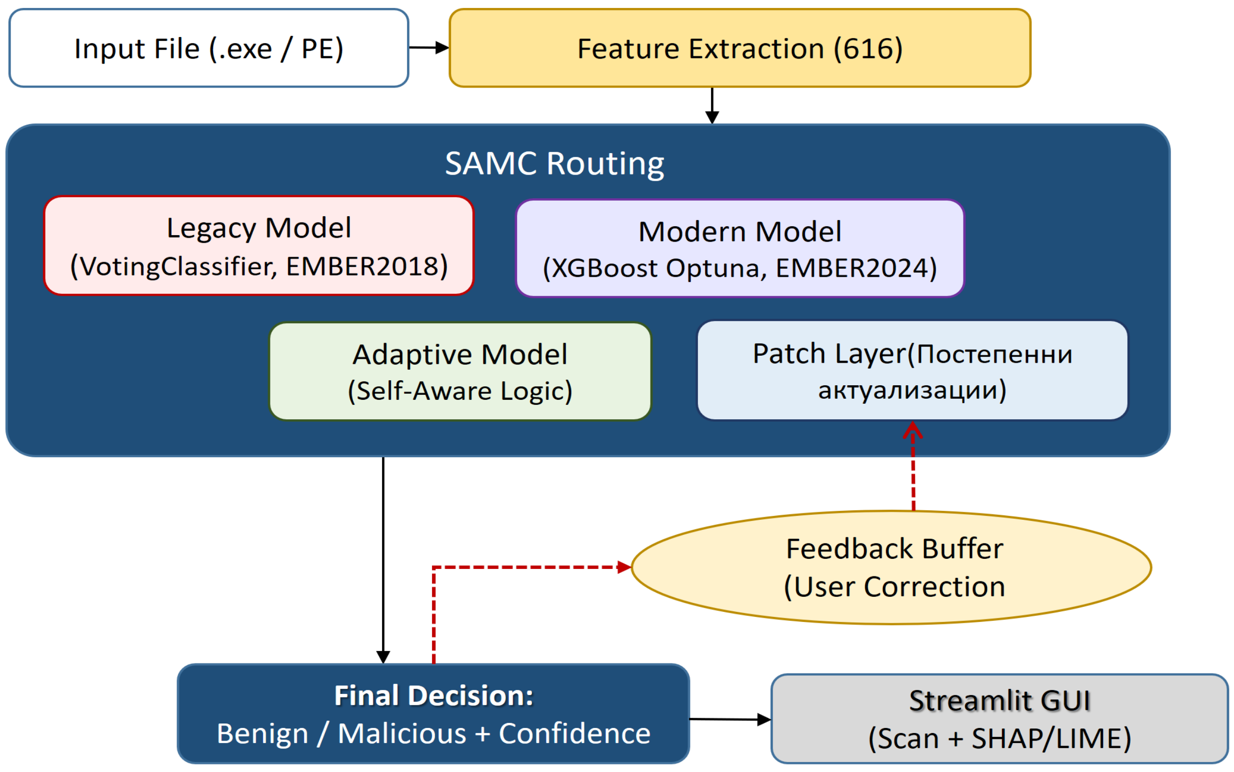

Input File (.exe / PE)
Feature Extraction (616)
SAMC Routing
Legacy Model (VotingClassifier, EMBER2018)
Modern Model (XGBoost Optuna, EMBER2024)
Adaptive Model (Self-Aware Logic)
Patch Layer(Постепенни актуализации)
Feedback Buffer (User Correction
Final Decision: Benign / Malicious + Confidence
Streamlit GUI (Scan + SHAP/LIME)

Fig. 2.17. Shipka Guard components showing feature extraction, SAMC routing, feedback buffer, correction layer, and integration with a graphical user interface.

The framework follows a pipelined design where uploaded PE files are processed through feature extraction, model selection (SAMC routing), optional patching, and decision making. User feedback is continuously collected and integrated to improve the robustness of the model over time.

The system combines three types of models:

(1) Legacy model (trained on EMBER 2018);

(2) Modern model (trained on EMBER 2024); and

(3) Adaptive model (periodically retrained via feedback).

The predictions are routed using SAMC as follows:

- If a model predicts 0.85 or higher, then it is selected for prediction.
- Otherwise, the fallback ensemble with weights: Adaptive model 0.5, Modern model 0.3, Legacy model 0.2 is used.

To ensure flexibility and comparability across environments, the proposed system should support four scanning modes:

(1) Auto (SAMC): Trust-aware routing with a trust threshold and fallback ensemble.

(2) Force Adaptive: Enforces the use of the latest adaptive model.

(3) Force Modern: Enforces the use of the modern model.

(4) Force Legacy: Enforces the use of the legacy model.

The modes support both automated confidence-based behavior and manual comparison between generations of models.

The correction layer is a lightweight SGDClassifier that is trained incrementally on user feedback. Its weight scales dynamically as a function of unique feedback records, ensuring that its correction behavior remains robust for small sample sizes and with many validation corrections, its weight increases over time. Specifically, the correction grows according to the formula:

$$w_{patch} = min\left(\alpha, \frac{n}{n+k}\alpha\right) \qquad (2.11)$$

where n is the number of unique feedback samples, α is the maximum contribution for the selected preset, and k is a smoothing constant controlling the growth rate.

Three operational presets are supported:

(1) Conservative: $\alpha = 0.12$, $k = 300$;

(2) Balance: $\alpha = 0.20$, $k = 100$; and

(3) Aggressive: $\alpha = 0.30$, $k = 50$.

There is also an option for synthetic FN/FP injection, which helps in assessing the robustness and stress testing of the patch mechanism before large-scale deployment. This design allows the patch layer to have a negligible impact when only a few feedback samples are available, while slowly increasing its impact as more stable patches are recruited (Fig. 2.18).

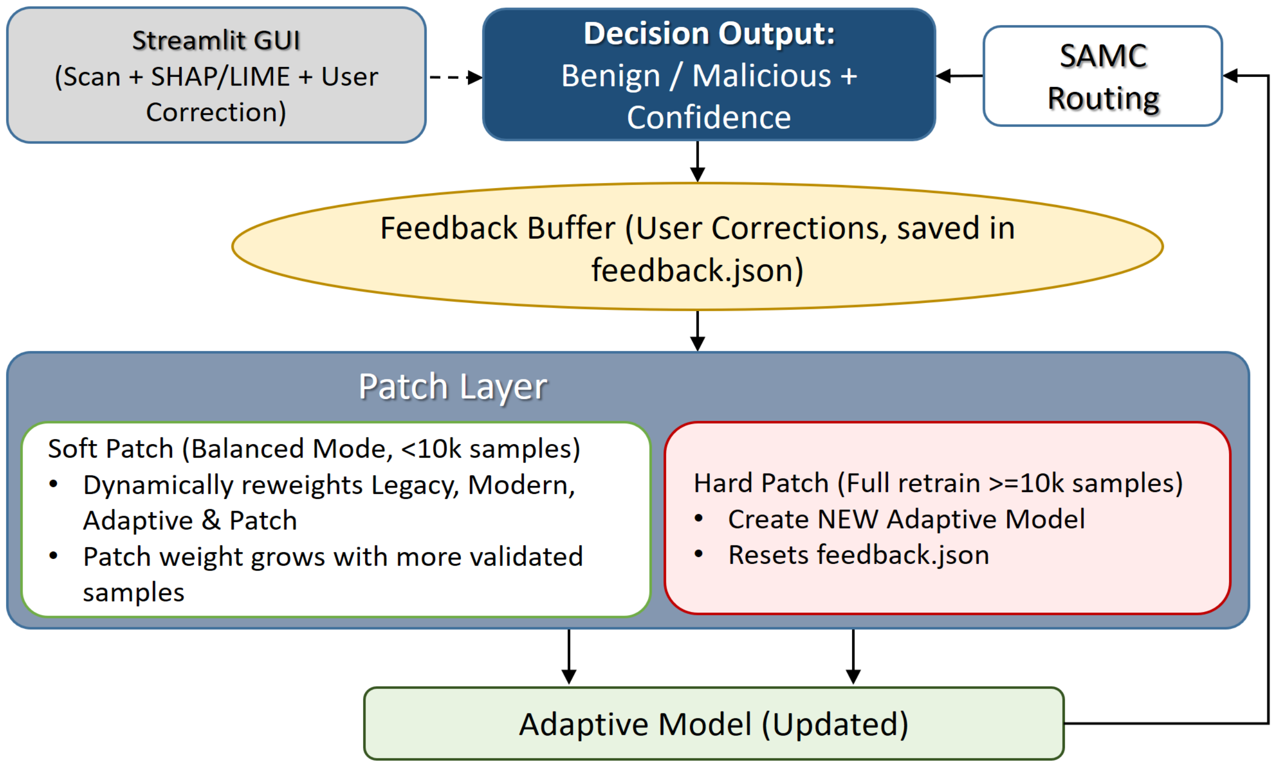

Streamlit GUI
(Scan + SHAP/LIME + User Correction)
Decision Output:
Benign / Malicious + Confidence
SAMC Routing
Feedback Buffer (User Corrections, saved in feedback.json)
Patch Layer
Soft Patch (Balanced Mode, <10k samples)
• Dynamically reweights Legacy, Modern, Adaptive & Patch
• Patch weight grows with more validated samples
Hard Patch (Full retrain >=10k samples)
• Create NEW Adaptive Model
• Resets feedback.json
Adaptive Model (Updated)

Fig. 2.18. Patch layer adaptation: soft patching (feedback-driven weighting) versus hard patching (full retraining with ≥10k samples).

Once the feedback buffer reaches a configurable threshold (≥ 10,000 samples), the system initiates a full retraining of the adaptive model in XGBoost. The feedback dataset is then exported in JSONL format for reproducibility purposes and the model is given a version in the models directory.

CHAPTER 3. EVALUATION TESTING OF PROPOSED MODELS FOR VIRTUAL MACHINE SELECTION, HYBRID ALGORITHMS AND MALWARE DETECTION FRAMEWORKS

3.1. Evaluation testing of group decision-making models for virtual machine software selection

For the purposes of analysis and malware detection, it is necessary to install the Windows virtual machine desktop software. For the purposes of numerical testing, published information about already evaluated software products was used, which are shown in Table 3.1.

Table 3.1. VM Software for Desktop Windows Deployment

VM Software	Review Rating			
	Ease of Use	***Customer Service***	***Features***	***Value for Money***
Microsoft Azure	4.1	4.2	4.6	4.2
VirtualBox	4.3	4.1	4.5	4.7
NAKIVO Backup & Replication	4.8	4.8	4.7	4.7
Altaro VM Backup	4.8	4.8	4.6	4.7
DiskStation	4.6	4.2	4.7	4.6
Ahsay Offsite Backup Server	3.8	3.4	4.0	3.7
Uranium Backup	4.7	4.7	4.5	4.7
Comet Backup	4.4	4.6	4.5	4.6
VMware Workstation Pro	4.6	4.6	4.7	4.2
Iperius Backup	4.1	4.1	4.1	4.3

Note: Data are taken from:
https://www.capterra.com/virtual-machine-software/?sortOrder=most_reviews&platform=%5B4%5D

3.1.2. Results

To test the effectiveness of the proposed models for selecting virtual machine software, a panel of three experts presented their opinions on the importance of four criteria: 1) ease of use; 2) customer service; 3) features; and 4) value for money. The scores for these criteria, shown in Table 3.1, were normalized to a range from 0 to 1 to obtain a comparable range with other coefficients used in the formulated models. The coefficients that express the importance of the criteria according to the opinions of the top managers are shown in Table 3.2.

Table 3.2. DMs Preferences Concerning Evaluation Criteria

Managers	Coefficients expressing the importance of the criteria			
	Ease of Use	***Customer Service***	***Features***	***Value for Money***
DM-1	0.25	0.25	0.25	0.25
DM-2	0	0.50	0.50	0
DM-3	0	0.50	0	0.50

Using the data from Table 3.2, the formulated optimization model (2.1) – (2.3) and the normalized scores in the range from 0 to 1, the overall performance of all alternatives can be determined according to the different perspectives of the group members, as shown in Fig. 3.1.

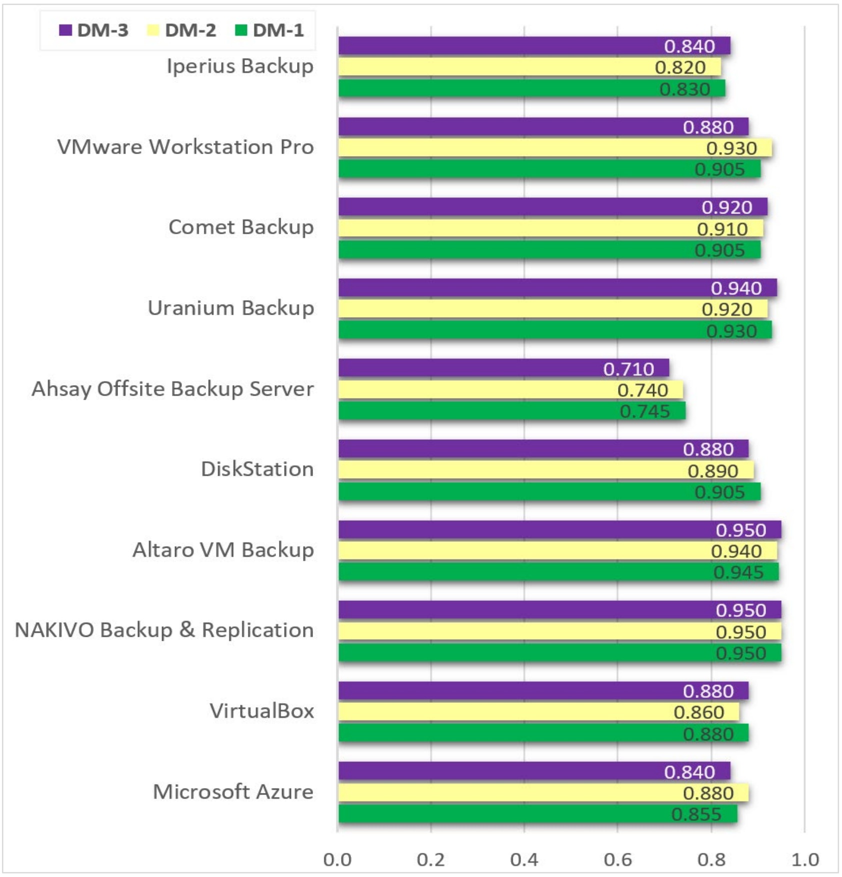


Fig. 3.1. General presentation of all alternatives.

It is interesting to show how the final group decision is affected when additional coefficients for the opinions of group members are used, shown in Table 3.3.

Table 3.3. Weights for importance of experts' opinions

Weights for experts in forming the final group decision		
DM-1	***DM-2***	***DM-3***
0.20	0.55	0.25

Using the data from Table 3.1, Table 3.2 and Table 3.3, together with the proposed group decision-making model (2.1) – (2.3), the following final ranking of the alternatives is obtained, as shown in Fig. 3.2.

The final group decision formed for the choice is composed of the preferences of all group members, together with the additional coefficients from Table 3.3, determines "NAKIVO Backup & Replication" as the best choice, since its performance has the highest value, equal to 0.95.

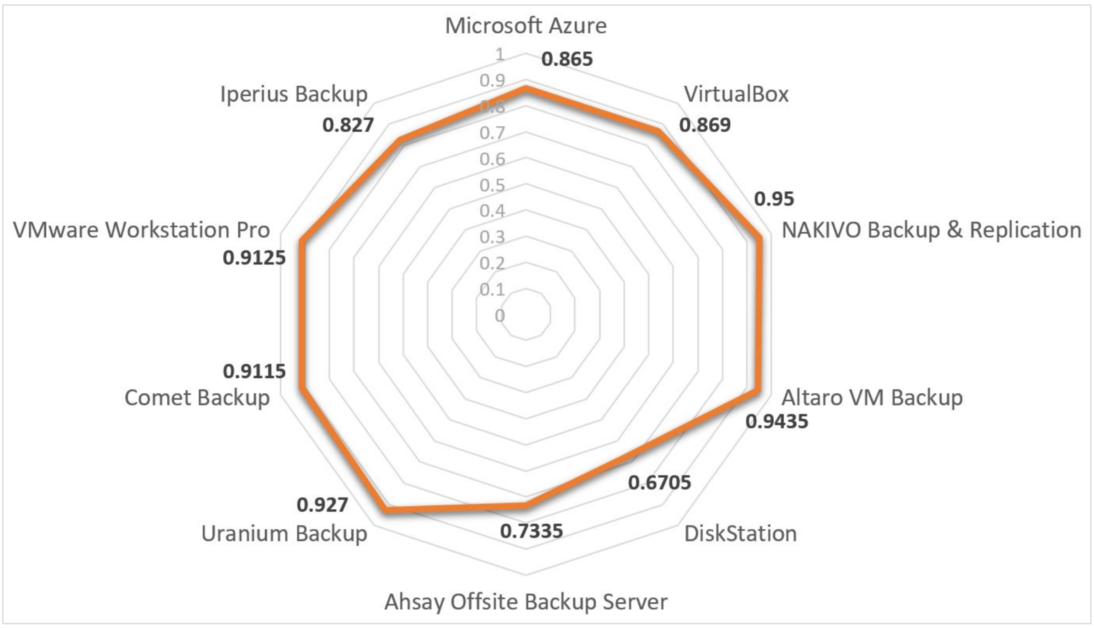


Fig. 3.2. Ranking of alternatives, taking into account the group decision.

It is possible to refine the choice by reducing the set of alternatives from which to choose by using the proposed second optimization model for group decision making (2.4) – (2.9). In this model, an optimization problem is formulated, the solution of which determines more than one alternative simultaneously, reducing the initial set of alternatives. The simultaneous determination of 3, respectively 5 alternatives is shown in Table 3.4.

Table. 3.4. Variable values when choosing multiple alternatives

VM software	Choice of 3 alternatives	Choice of 5 alternatives
Microsoft Azure	0	0
VirtualBox	0	0
NAKIVO Backup & Replication	1	1
Altaro VM Backup	1	1
DiskStation	0	0
Ahsay Offsite Backup Server	0	0
Uranium Backup	1	1
Comet Backup	0	1
VMware Workstation Pro	0	1
Iperius Backup	0	0

The proposed two mathematical models actually show that they can be successfully used for selecting software for a virtual machine. The first model can determine the best alternative, taking into account the criteria scores, the importance of the criteria and the importance coefficients of the opinions of the group members. The second model can determine more than one alternative by solving the corresponding optimization problem, thanks to the use of binary integer variables.

3.2. Testing the proposed improved static analysis approach for malware detection by optimizing feature extraction, combining different machine learning algorithms

3.2.1. Output data

The IoT-23 dataset, generated by Avast AIC Labs using Zeek Network Security Monitor, contains intercepted records of benign and malicious traffic from 2018 to 2019. The dataset includes full and lighter versions. It denotes different attack types such as “Benign”, “DDoS”, “Okiru”, and “Part of a Horizontal Port Scan”. It is interesting to investigate whether the proposed hybrid algorithms could improve the accuracy of malware detection. For this purpose, a subset of the IoT-23 dataset consisting of 500,000 instances is used, which includes both benign and malicious classes.

The experiment is conducted following 5 main stages, as shown in Fig. 3.4:

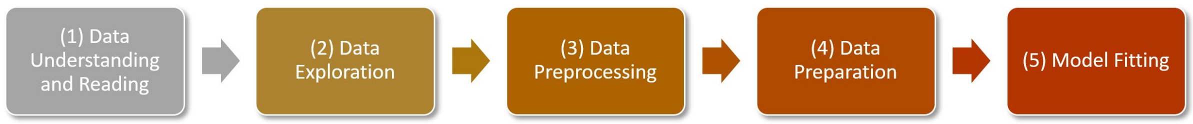


Fig. 3.4. Experiment workflow

3.2.2. Analysis and discussion of results

The performance results using a single implementation of the algorithms and the proposed hybrid algorithms are shown in Table 3.5.

Table 3.5. Algorithm performance results

Model	Accuracy	Precision	Recall	F1-Score
NB	0.92	0.72	0.68	0.70
KNN	0.96	0.86	0.84	0.85
RF	0.98	0.94	0.92	0.93
H1 (NB & KNN)	0.97	0.88	0.86	0.87
H2 (NB & RF)	0.98	0.94	0.92	0.93
H3 (NB & KNN & RF)	0.99	0.96	0.94	0.95

As can be seen from Table 3.5, some of the combined algorithms show better performance. The comparison between all these indicators and the different models and combinations of models is shown in Fig. 3.5.

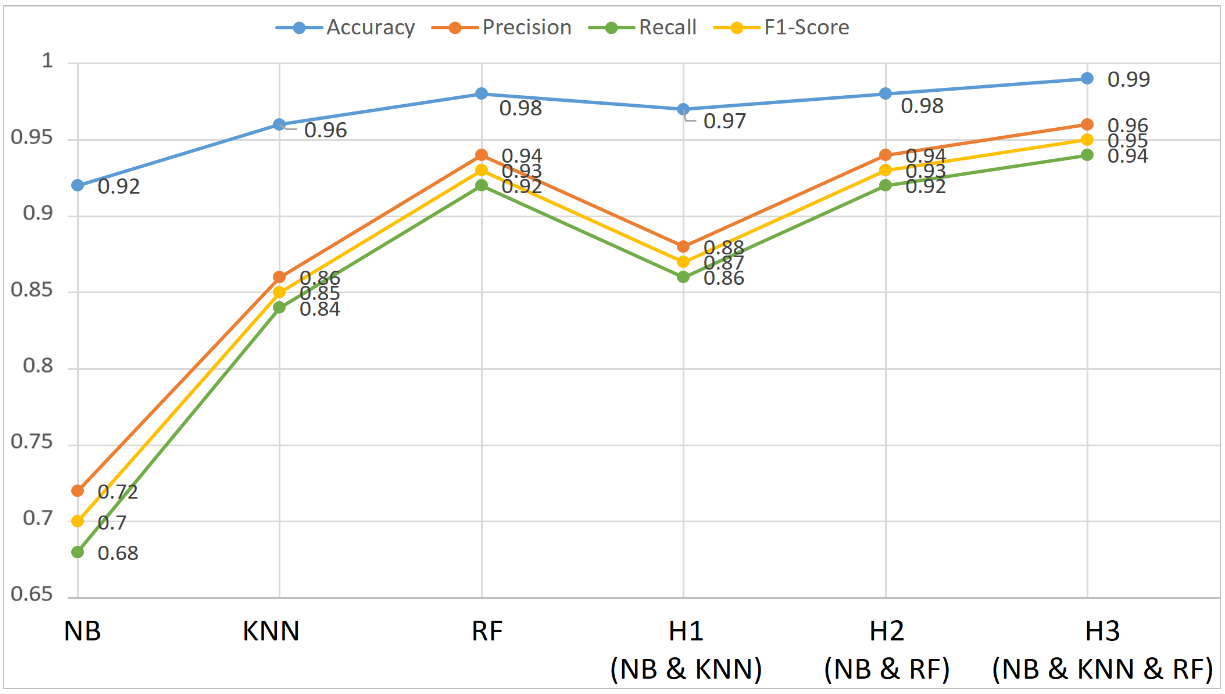


Fig. 3.5. Comparison of the performance of different algorithms and their combinations

The proposed hybrid algorithms contribute significantly to improving performance. The results obtained from the experiments highlight the importance of optimizing feature extraction, which is crucial for increasing the accuracy of machine learning models. Efficient feature extraction transforms raw data into a format that is more informative and discriminative for the task, improving the learning process.

3.3. Evaluation testing of the proposed framework for static malware classification using feature optimization and ensemble learning

Feature extraction is done by modifying existing Python feature code from the EMBER project to work with lief 0.16.5 and Python 3.8. The entropy value, section names, import/export items, and reported statistics for metadata, strings, etc. Each PE file is represented as a vector of 616 variables.

3.3.1. CLI Scanner: Practical Use Case

The CLI scanner is specifically designed for use in an environment (Python 3.8 and lief 0.16.5) to ensure full compatibility with EMBER 2018 feature extraction and to prevent issues with newer PE file structures. The CLI scanner provides a command-line environment that uses a user-supplied input argument (the path to the PE file) to extract features (using a built-in mechanism), load the final VotingClassifier model, and obtain a risk level. The developed CLI scanner is fully offline, so it can run in isolated networks and environments (i.e. virtual). It is built in a Python environment and is a packaged .exe application using PyInstaller. The features are extracted using lief and are exactly 616. The CLI scanner is able to generate and maintain a log file showing detailed scan status and probability estimates.

The performance of the CLI scanner is as follows: The average time to scan a single PE file is ~1.0 seconds. Each console output is recorded and logged, which includes all the details as you would see in console output: date and time, file path, predicted probability, risk score, and final class. The scanner can process a single file or multiple files at a time (with specified paths), but directory scanning will be developed in the future.

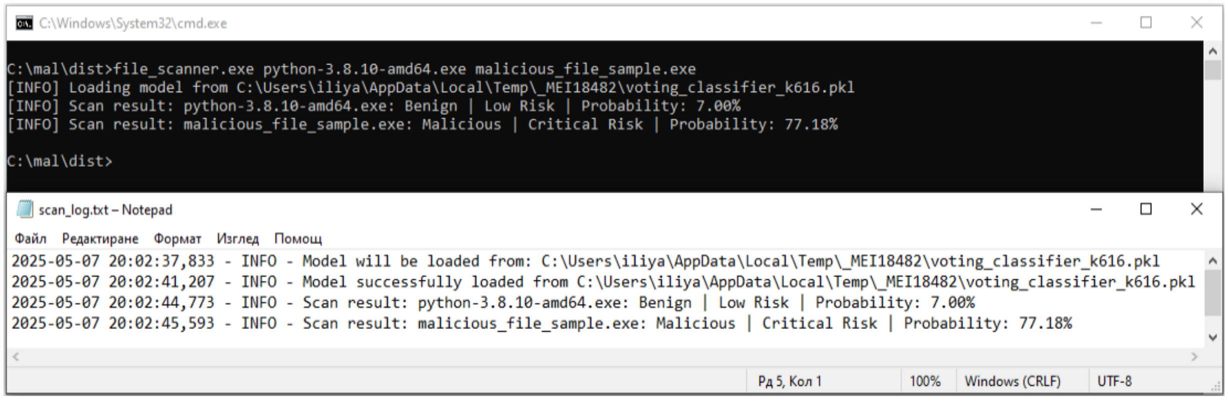
An example CLI scanner command is:

```
python scanner.py --file suspicious.exe
```

An example output of the CLI scanner is:

```
[INFO] Scan result: suspicious.exe: Malicious | Critical Risk |
Probability: 95%
```

Actual console output from a CLI scanner showing an example of scanning 2 PE files, one benign PE file (python-3.8.10-amd64.exe) and one malicious PE file (malicious_file_sample.exe). The output provides a risk category, probability score, and log entry. All output is written to scan_log.txt, as shown in Fig. 3.6.


C:\Windows\System32\cmd.exe
scan_log.txt – Notepad
Файл Редактиране Формат Изглед Помощ
Рд 5, Кол 1
100%
Windows (CRLF)
UTF-8

```
C:\mal\dist>file_scanner.exe python-3.8.10-amd64.exe malicious_file_sample.exe
[INFO] Loading model from C:\Users\iliya\AppData\Local\Temp\_MEI18482\voting_classifier_k616.pkl
[INFO] Scan result: python-3.8.10-amd64.exe: Benign | Low Risk | Probability: 7.00%
[INFO] Scan result: malicious_file_sample.exe: Malicious | Critical Risk | Probability: 77.18%

C:\mal\dist>
```

```
2025-05-07 20:02:37,833 - INFO - Model will be loaded from: C:\Users\iliya\AppData\Local\Temp\_MEI18482\voting_classifier_k616.pkl
2025-05-07 20:02:41,207 - INFO - Model successfully loaded from C:\Users\iliya\AppData\Local\Temp\_MEI18482\voting_classifier_k616.pkl
2025-05-07 20:02:44,773 - INFO - Scan result: python-3.8.10-amd64.exe: Benign | Low Risk | Probability: 7.00%
2025-05-07 20:02:45,593 - INFO - Scan result: malicious_file_sample.exe: Malicious | Critical Risk | Probability: 77.18%
```

Fig. 3.6. Actual console output from a CLI scanner.

The CLI scanner identifies 5 risk levels: 0: Benign; 1: Low risk; 2: Medium risk; 3: High risk; and 4: Critical risk. The proposed CLI scanner can be used by IT administrators of government or corporate systems that do not have access to the Internet. The CLI scanner is compatible with outdated hardware with old operating systems. This scanner can be used as part of script automation or internal security systems.

3.3.2. Visualizations and metrics

Fig. 3.8 shows a clustered bar chart showing three different representative models from three different strength classes – strong (VotingClassifier), medium (Random Forest with k=200) and weak (Random Forest with 10% training set) – with respect to five evaluation metrics.

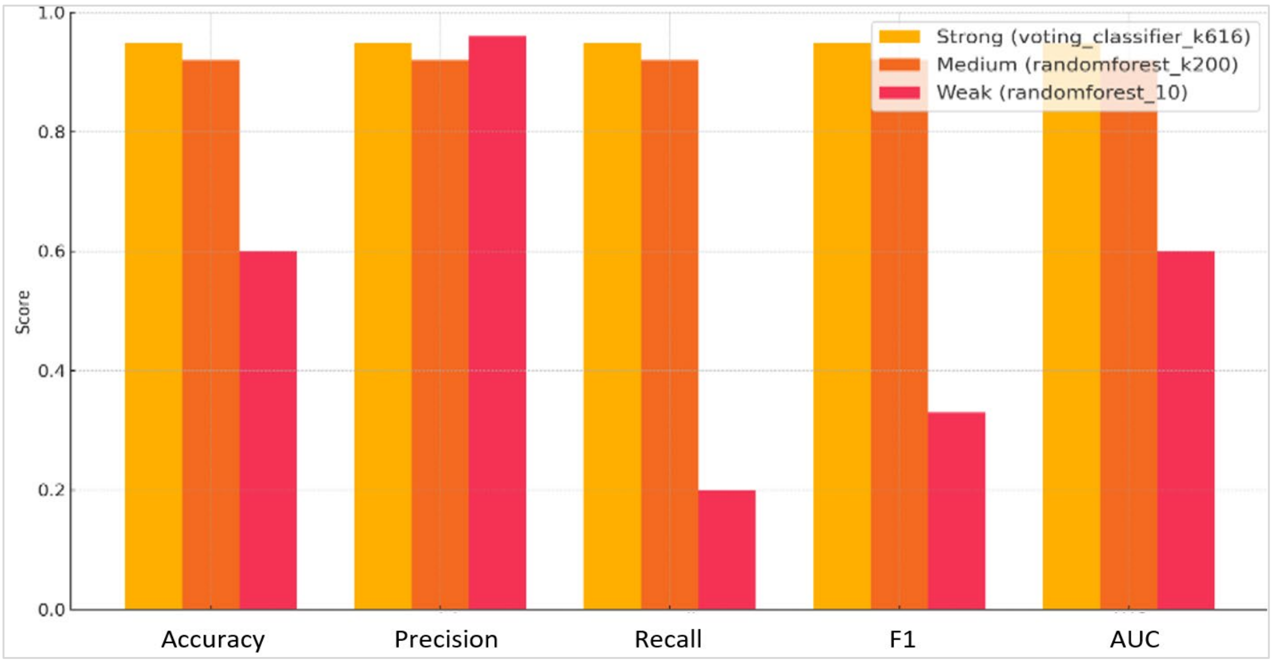


Fig. 3.8. Comparison of strong, medium and weak models in terms of accuracy, precision, recall, F1 and AUC

The ROC curve shown in Fig. 3.9 was implemented to see the results of a strong model (VotingClassifier), a medium model (Random Forest with k=200), and a weak model (Random Forest training size, 10%). The AUC of VotingClassifier is the highest (0.9424); the weak model is slightly better than random guess (0.5972). The difference in AUC overall indicates that training size is less powerful than optimization on classification ability.

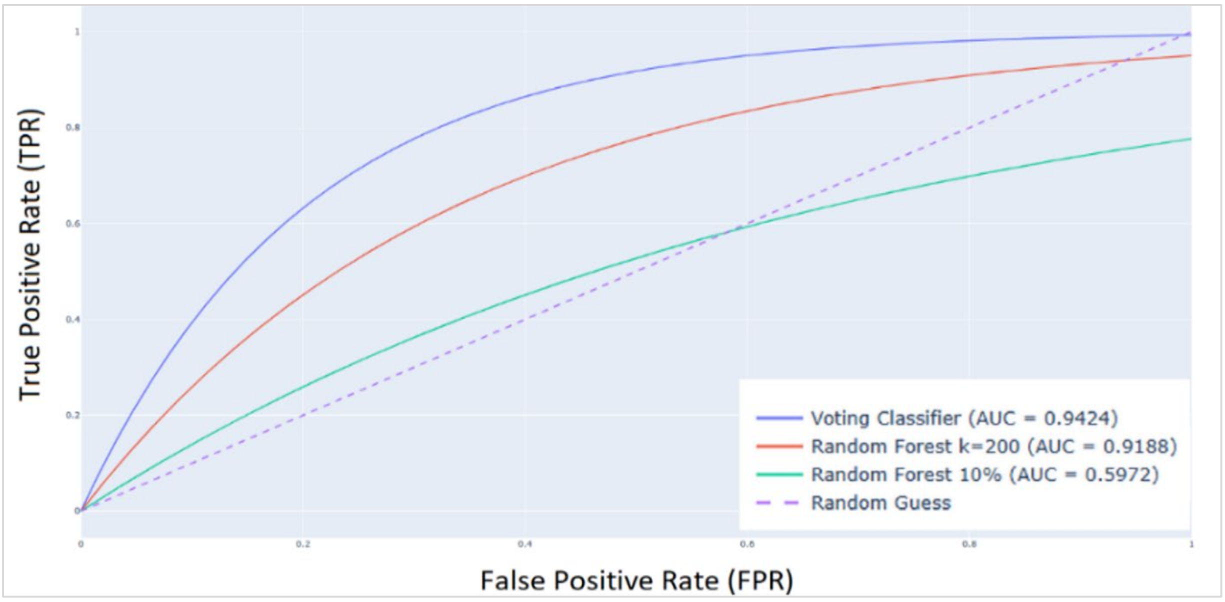


Fig. 3.9. Comparative ROC curves

Fig. 3.11 illustrates the 20 most influential features based on VotingClassifier.

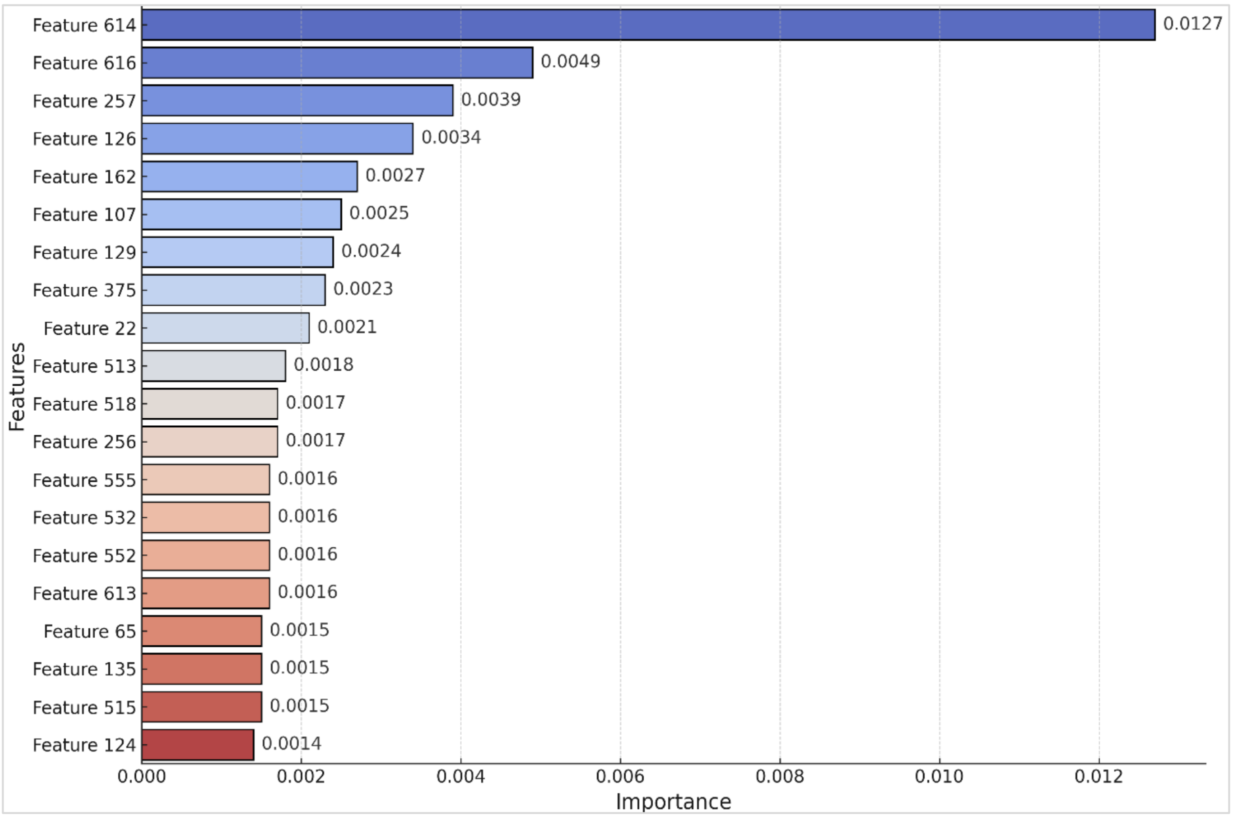


Fig. 3.11. The 20 most influential features based on VotingClassifier.

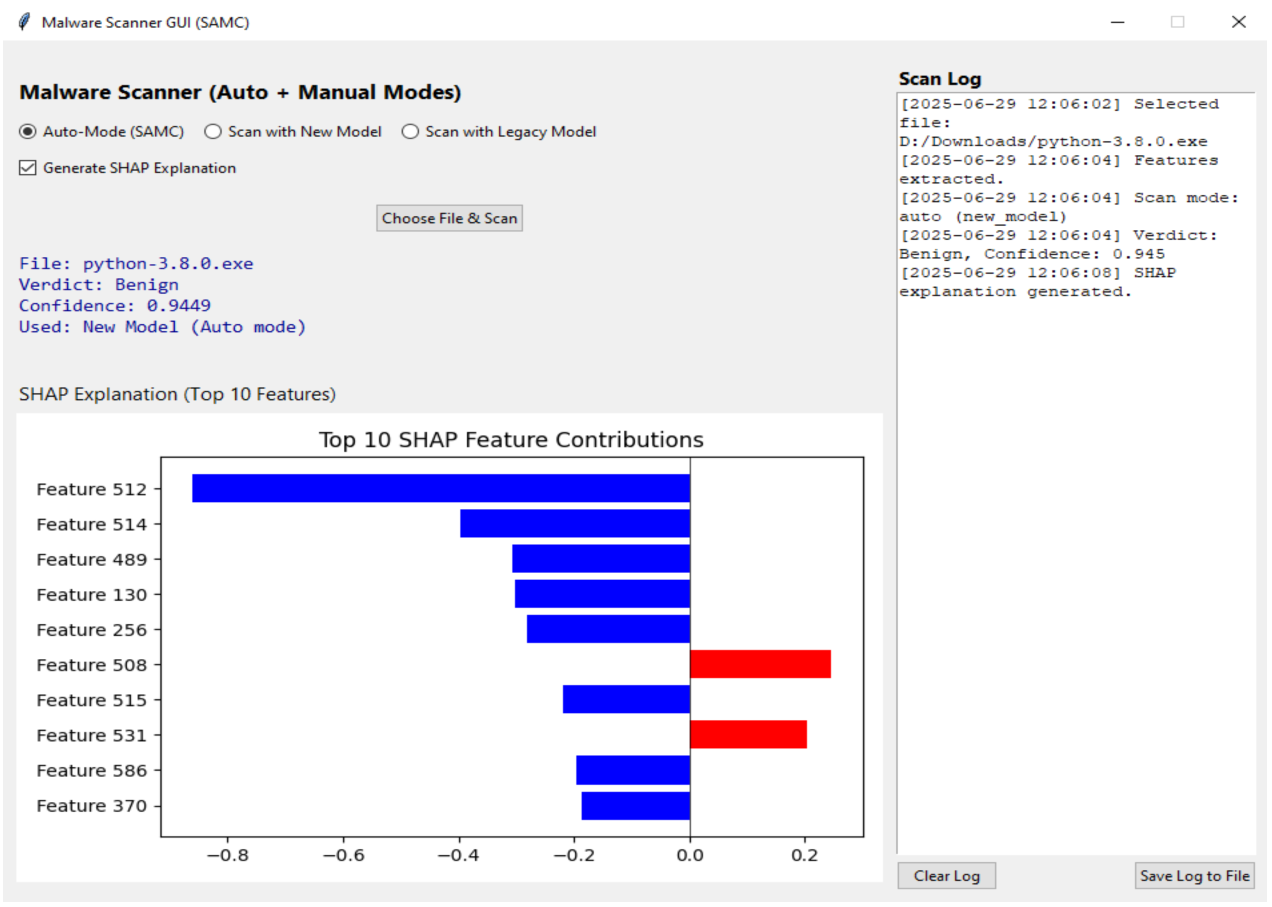
The features are ordered from most to least influential as a feature identifier (based on descending values), where feature 614 is the most influential. The diagram is for model explainability and shows which features are responsible for the greatest impact on decisions.

3.4. Results of testing the proposed self-aware malware classification by routing models based on a trust system for feature selection and explainability

This section outlines the tests conducted using the proposed SAMC-based malware detection framework, measuring performance and operational behavior, as described in Chapter 2, Section 2.4. Here, we present not only the numerical results performed, but also analyze the flexibility of the framework, as well as how the reasoning is explicable and how the framework provides trust and adaptive change.

3.4.1. Graphical User Interface Design and Features

The user interface design is divided into 3 main panels (Fig. 3.12):



Settings: ● Auto-Mode (SAMC) ○ Scan with New Model ○ Scan with Legacy Model; ☑ Generate SHAP Explanation

```
File: python-3.8.0.exe
Verdict: Benign
Confidence: 0.9449
Used: New Model (Auto mode)
```

```
[2025-06-29 12:06:02] Selected
file:
D:/Downloads/python-3.8.0.exe
[2025-06-29 12:06:04] Features
extracted.
[2025-06-29 12:06:04] Scan mode:
auto (new_model)
[2025-06-29 12:06:04] Verdict:
Benign, Confidence: 0.945
[2025-06-29 12:06:08] SHAP
explanation generated.
```

Fig. 3.12. Example GUI interface with SHAP explanation for a harmless file scanned via automatic mode using the new model.

The left panel allows the user to select the scan mode, enable/disable SHAP annotations, select a file to scan, and view the result. The right panel allows viewing a real-time scan log with timestamped messages. The SHAP annotations section presents a horizontal bar chart depicting the 10 most influential features contributing to the predictions, colored by sign (red = positive class, blue = negative class) An experimental result of a scan of a harmless file with SHAP explanation is presented in Fig. 3.12:

3.4.2. Routing execution time and latency

To determine the perceived real-time performance of the SAMC framework, a latency benchmark was performed on all .exe examples located in the test directory. The execution time was measured from the point at which feature extraction was initiated to the point at which the prediction identified by the terminal using the SAMC routing logic (legacy model, new model or fallback) was reached. The scans of these 100 files can be summarized as shown in Table 3.6:

Table 3.6. Metrics and values from scanning 100 files

Metric	Value (seconds)
Fastest	0.0569
Slowest	0.5129
Average	0.2334

This evidence shows that even using logic based on the confidence in the prediction has enabled near real-time classification. Fallback logic has sometimes been used for cases where it has been determined that there is no uncertainty about the prediction outcome (e.g. below threshold), as it is a self-aware design that takes into account the ambiguity/unknownness of either the input data or the models that have provided confidence scores that are rated as low.

3.4.3. Performance Comparison: Routing vs. Non-Routing

To evaluate the practical effectiveness of trust-based routing mechanisms and fallback mechanisms, we conducted an experiment in which we evaluated four different inference strategies.:

1. **Legacy Model Only** – uses VotingClassifier trained on EMBER 2018 and relies solely on historical models.
2. **New Model Only** – uses the tuned and Optuna-optimized XGBoost classifier trained on EMBER 2024, which uses the newer data distribution.
3. **Combined Voting (No Routing)** – simply a naive ensemble of the two models with soft voting without any confidence thresholds.
4. **SAMC Routing + Fallback** – the proposed intelligent model that dynamically selects a model based on the confidence of the prediction and that uses weighted fallback logic if no model passes the threshold.

For this experiment, a balanced test set of 50 known malware samples and 50 benign executables was created, with each file passing through all four inference channels after being statically analyzed into a 616-dimensional PE feature vector. The results are summarized in Table 3.7:

Table 3.7. Comparative performance of different inference strategies on a balanced test set (50 malicious and 50 benign files)

Mode	Accuracy	F1-Score	ROC-AUC	False Positives	False Negatives
Legacy Only (EMBER 2018)	0.6740	0.5000	0.9425	1	30
New Model Only (EMBER 2024)	0.8260	0.8421	0.8820	9	5
Combined Voting (No Routing)	0.8680	0.8608	0.9683	3	7
SAMC Routing + Fallback	0.9140	0.9036	0.9981	1	6

These results show that the SAMC routing strategy outperformed all other modes on all evaluation metrics, achieving an F1-score above 0.9 and perfect precision, but also very high completeness and ROC-AUC. Unlike the traditional model, which showed overly conservative reporting behavior (no false positives but very low responsiveness), SAMC achieves a middle ground by either selectively trusting the more confident model or applying a weighted fallback method. This supports the conclusion that a confidence-based model selection strategy not only improves robustness but also helps reduce false positives and false negatives when comparing performance.

When examining classification performance against naive ensemble voting, the SAMC approach outperformed all other methods because it avoided blind averaging and instead adapted to use the more confident model selectively for each sample.

3.4.4. SHAP explanation for malicious file

To provide another example of explainability and trust, we prepared a local SHAP explanation for one of the malignant sample results from the new model. Fig. 3.13 shows the 10 most important features that contributed to the "malicious" conclusion.

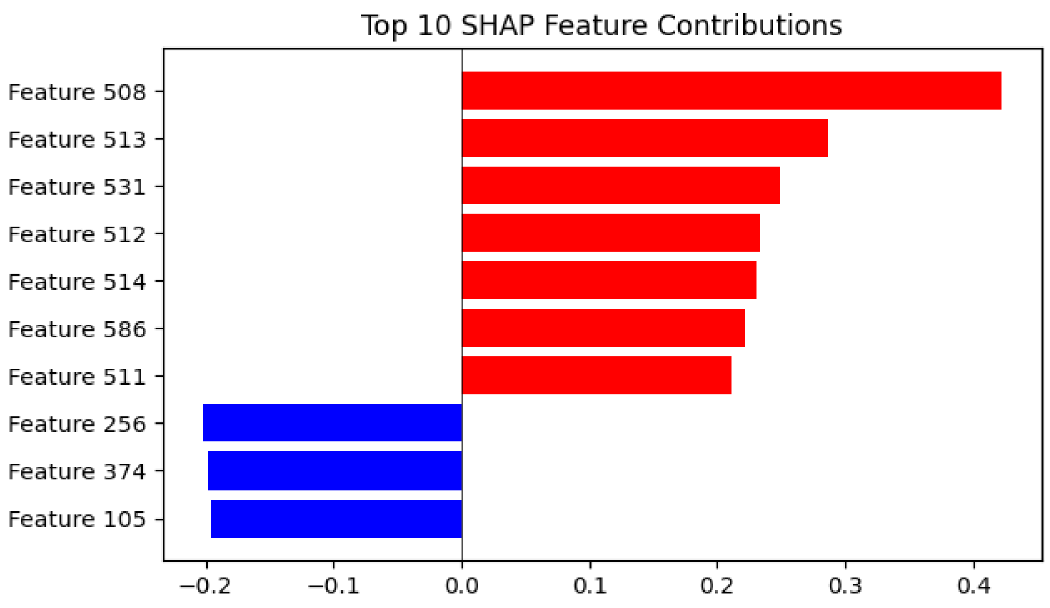


Fig. 3.13. SHAP explanation for a malicious file. Red bars indicate positive contributions to the malicious prediction, while blue bars indicate benign features

SHAP visually shows that many of the entropy and byte histogram segments have a dominant significance for the decision, which means that the model strongly pays attention to anomalies in the statistical features. Some of the same features also show a large bias in the KS test (e.g. feature 507), which supports the rotation of adaptive and explainable detection.

3.5. Results of the developed and tested Shipka Guard application based on the proposed adaptive trust-based framework for malware classification with feedback corrections

The graphical user interface (GUI) of the developed application, named "Shipka Guard", is built using the Streamlit framework. This solution allows for rapid deployment and provides analysts with access without installation or complex configurations. Within the GUI, users can upload PE files, make real-time predictions with associated confidence values, and obtain explainable (XAI) visualizations. Both SHAP and LIME explanations are included, with a complementary view provided; SHAP shows global feature contributions based on the model architecture, while LIME shows locally interpretable approximations based on the local solution. In addition, the application has an option that can be used to manually select adaptive versions in a drop-down menu to compare and contrast parallel adaptive models. The built-in feedback import/export also allows for saving or sharing of analyst feedback, which contributes to the reproducibility of each experiment and achieves a collaborative workflow. The combination of detailed logs and adjustable thresholding options, with the capabilities, makes Shipka Guard an interactive research and operational platform.

In the developed version of the "Shipka Guard" application, two XAI methods are integrated: (1) LIME locally explains all models and (2) SHAP globally explains the contribution of built-in features in tree-based models (Adaptive XGBoost). Embedded visualizations are presented in the graphical user interface to help understand and validate the solutions. These solutions are integrated into the proposed and implemented architecture of the machine learning system with feedback for classification and explainability, shown in Fig. 3.14.

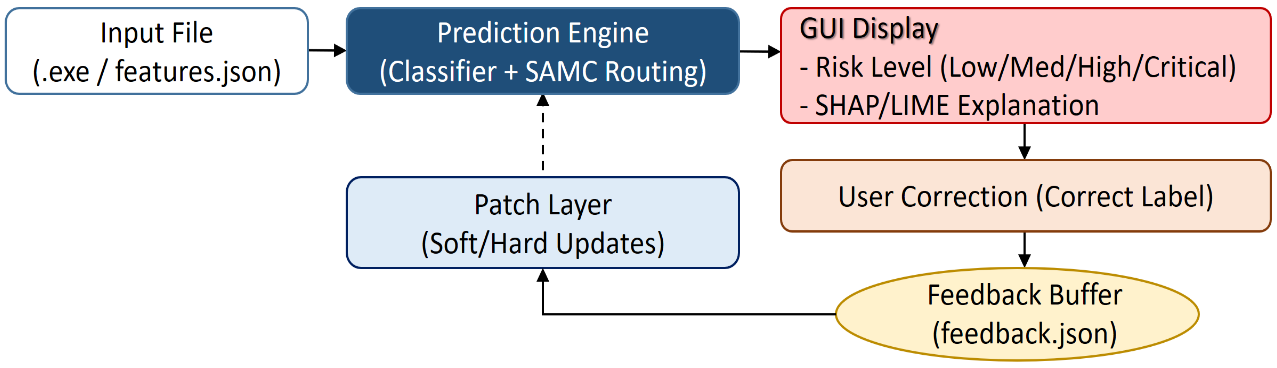


Fig. 3.14. Architecture of the machine learning system with feedback for classification and explainability (XAI)

All system events (scans, feedback records, retraining) are logged in logs/scan.log. Additionally, the framework tracks the execution time (in ms) for each prediction to ensure that the system will run in real-time on the user's hardware. This feature provides insight into the feasibility and deployment of the project in enterprise environments, including scan results, feedback records, patch updates, and retraining events.

3.5.1. Evaluation of the developed Shipka Guard application, based on the proposed framework

The feasibility assessment of the developed Shipka Guard application, based on the proposed framework, uses two publicly available benchmark datasets: EMBER 2018, containing over 1 million PE samples characterized by 616 static features (i.e. histograms, entropy, string statistics) and used as a baseline "legacy" model for the study; and EMBER 2024 with approximately 960,000 samples. EMBER 2024 contains modern malware families and is harmonized in the same 616-dimensional feature space for comparison with the EMBER 2018 dataset. In addition, a third user-generated dataset is used. The framework is structured to integrate user-generated data in a simple way to harmonize in the same 616-dimensional feature space.

For the purpose of fairness and reproducibility of the experiments, a balanced set of 100,000 samples (50,000 benign, 50,000 malicious) was created from EMBER2018 and EMBER2024. Initially, the baseline models and the SAMC automatic routing method were compared in the balanced evaluation, as shown in Table 3.8.

Table 3.8. Performance of Legacy, Modern and SAMC Auto (T=0.85)

Mode	Accuracy	Precision	Recall	F1	ROC-AUC	FN	FP
Force Legacy	0.92409	0.9540	0.8906	0.9213	0.9621	5472	2148
Force Modern	0.96641	0.9793	0.9529	0.9660	0.9955	2353	1006
Auto (SAMC) [T=0.85]	0.95917	0.9852	0.9324	0.9580	0.9942	3381	702

The considered effect of Patch's slight adaptation versus full retraining, as shown in Table 3.9.

Table 3.9. Ablative study comparing Modern, Patch with 500 feedback samples, and full retraining with 10,000 feedback samples

Mode	Accuracy	Precision	Recall	F1	ROC-AUC	FN	FP
Base (Modern)	0.96641	0.9793	0.9529	0.9660	0.9955	2353	1006
Patch (500 fb)	0.96727	0.9799	0.9541	0.9668	0.9918	2294	979
Full Retrain (10k)	0.94719	0.9672	0.9257	0.9460	0.9906	3713	1568

The Patch layer trained with 500 feedback samples shows noticeable improvements: both FN (-59) and FP (-27) decrease, and the F1 score increases slightly. Thus, even a small and possibly unbalanced feedback buffer can be used to apply consistent, fast corrections. In contrast, a full retraining with 10,000 feedback samples yields a worse result. Although it seems counterintuitive, the explanation is rooted in bias and imbalance in the feedback pool: we multiply problematic samples and increase errors instead of correcting them. This really provides a critical lesson: quality and variety are just as important as the quantity of feedback.

SAMC Auto using different confidence thresholds is shown in Table 3.10.

Table 3.10. Metrics at different confidence thresholds

T	Precision	Recall	%Uncertain
0.80	0.9854	0.9310	6.5%
0.85	0.9852	0.9324	8.2%
0.90	0.9851	0.9341	10.6%

Lower thresholds (T=0.80) result in fewer holds but a higher risk of misclassifications. Using higher thresholds (T=0.90) increases the number of holds, which also improves reliability but comes at the cost of reduced throughput. This further supports the idea that T is an adjustable safety button.

The execution time per file was measured directly from the system log files for each model and the results are given in Table 3.11.

Table 3.11. Execution latency normalized to 100 samples per model

Mode	n_samples	Mean (ms)	Std (ms)	p50	p90	p99
Modern	100	47.6	24.5	33.1	83.6	91.3
Legacy	100	131.4	0.0	131.4	131.4	131.4
Adaptive	100	109.7	29.5	109.7	133.3	138.6

The Modern model achieves an average inference time of less than 50 ms, which is well below the target limit of 90 ms/sample. Legacy and Adaptive are slower (~110–130 ms), but still within real-time. These results confirm the feasibility of implementing the system even with SHAP/LIME enabled.

Three example cases illustrating the dynamics of feedback and explanations:

- **False negative corrected by Patch.** Fig. 3.15 (LIME) shows a malicious sample misclassified as benign. After correction with 500 feedback samples, the solution was corrected, reducing the FN.
- **False positive corrected by feedback.** A benign sample misclassified as malicious was added to the feedback buffer. The Patch model was corrected and the false positive was eliminated.
- **Uncertain case explained by LIME and SHAP.** At T=0.90, SAMC abstained. LIME showed conflicting local evidence, while the global significance of SHAP (Fig. 3.16) explained the influences of dominant features.

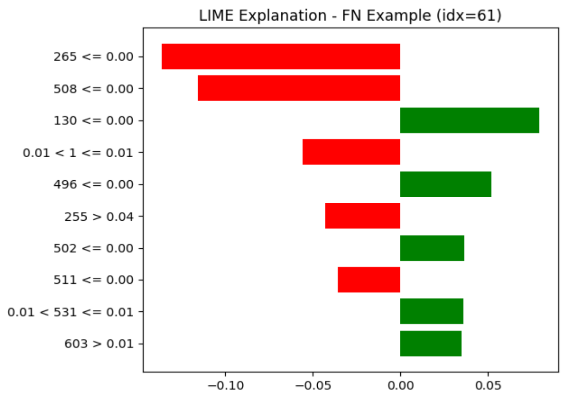


Fig. 3.15. LIME's explanation of FN has been corrected by Patch

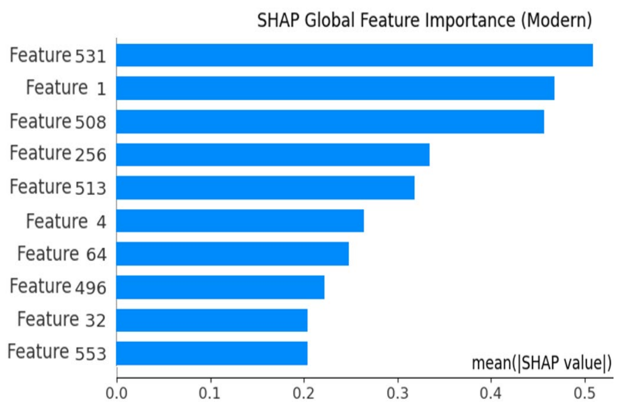
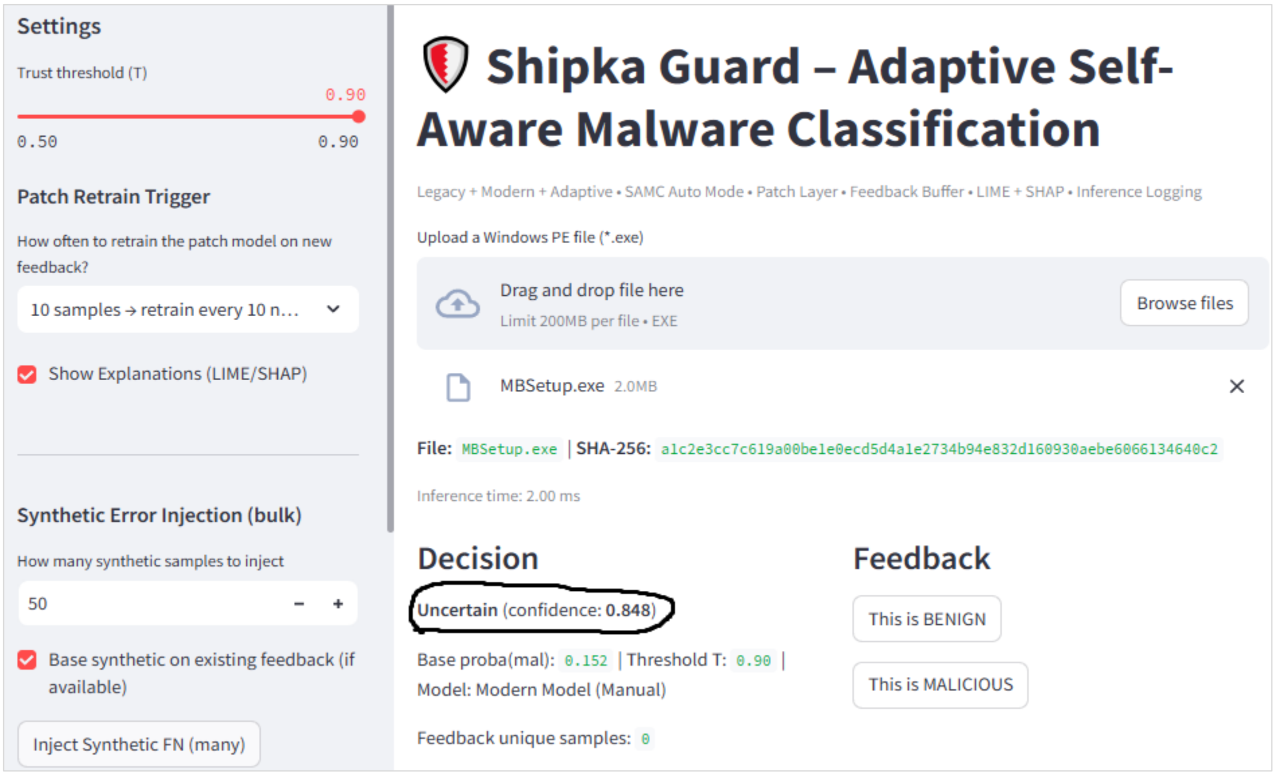


Fig. 3.16. Importance of global SHAP features for the Modern model

The GUI shown in Fig. 3.17 showed the restraint of the analyzer that could provide corrective feedback.


Settings

Trust threshold (T)

0.90

0.50 0.90

Patch Retrain Trigger

How often to retrain the patch model on new feedback?

10 samples → retrain every 10 n…

☑ Show Explanations (LIME/SHAP)

Synthetic Error Injection (bulk)

How many synthetic samples to inject

50

☑ Base synthetic on existing feedback (if available)

Inject Synthetic FN (many)

Shipka Guard – Adaptive Self-Aware Malware Classification

Legacy + Modern + Adaptive • SAMC Auto Mode • Patch Layer • Feedback Buffer • LIME + SHAP • Inference Logging

Upload a Windows PE file (*.exe)

Drag and drop file here
Limit 200MB per file • EXE

Browse files

MBSetup.exe 2.0MB

File: `MBSetup.exe` | **SHA-256:** `a1c2e3cc7c619a00be1e0ecd5d4a1e2734b94e832d160930aebe6066134640c2`

Inference time: 2.00 ms

Decision

Uncertain (confidence: **0.848**)

Base proba(mal): `0.152` | Threshold T: `0.90` |
Model: Modern Model (Manual)

Feedback unique samples: `0`

Feedback

This is BENIGN

This is MALICIOUS

Fig. 3.17. Screenshot of the user interface with "Uncertain decision" (T=0.90)

The evaluation confirms the effectiveness and limitations of the proposed framework. The old models are outdated, while the modern models perform well but produce more FP. SAMC Auto balances risks through dynamic routing. The Patch layer provides easy and lightweight error cleaning with small processed pools, and full retraining does not provide a preventive advantage unless there is an appropriate amount of feedback. The confidence threshold offers a tunable balance between completeness and restraints. The latency remains within the real-time range, while the case studies show how the explanation module (LIME/SHAP) and uncertain restraints subsequently increased the analysts' confidence or at least validated the results. Therefore, we justify the SAMC process as an effective and reliable process for adapting malware classification.

3.5.2. Discussions and conclusions

Shipka Guard integrates 4 scanning modes, synthetic error injection, JSON-based feedback export/import, and scalable analysis. This makes the Shipka Guard framework not only more adaptable, but also more suitable for real-world deployment conditions.

The Shipka Guard framework is based on an interactive graphical user interface that lowers the barrier for non-specialized users to experiment with malware classification. The feedback buffer allows the user to collaboratively correct the model, allowing them to export/import feedback files. The integration of explainability contributes to the confidence in the solutions, which is increased by more information about the features from a local and global perspective. The scalability logging reflects the adaptability of the system, which does not hinder real-time workflows. All these aspects show that adaptive malware detection can be practical and transparent.

The framework offers several new contributions. The first is a trust-aware routing mechanism that uses multiple generations of models, including a fallback mechanism. The next is a feedback buffer and a patch layer for lightweight adaptation from user patches and synthetic injections. It is implemented through an interactive Streamlit GUI that provides four scanning modes, JSON-based feedback sharing, and local/global explanations (LIME+SHAP). It also provides scalable logging that presents real-time (<90ms) inferences on user hardware.

The combination of adaptability, interpretability, and scalability addresses the challenges of continuously developing practical next-generation malware detection systems that consistently stay "true" to given threat models, rather than constantly changing. Including a version selection and feedback interface as part of the application allows for reproducibility and collaborative usability, making Shipka Guard not only a practical malware protection tool, but also a tool for collaborative research or evaluations.

CONCLUSION – SUMMARY OF THE RESULTS OBTAINED

Malware can take many different forms, such as viruses, worms, Trojan horses and ransomware, and can cause significant damage to people, organizations and even entire countries. Malware remains one of the most serious threats to information security. With the evolution of digital technologies, the methods of carrying out attacks are also changing, which requires the development of new algorithms for effective and timely detection of malicious behavior. Traditional solutions, based primarily on signatures, are no longer sufficient to deal with modern threats.

Considering the need to develop effective means of countering malware, the main goal of this dissertation work was determined, related to Research and analysis of the possibilities for detecting malicious software using machine learning. In order to implement experiments safely in a controlled environment, a mathematical model for selecting a suitable virtual machine is proposed. In a safe and controlled environment, experiments were conducted to determine the effectiveness of various machine learning algorithms and their performance for malware detection was analyzed. Based on the findings, an improved static analysis approach for malware detection was proposed by optimizing feature extraction and combining different machine learning algorithms. In addition to these efforts, a static malware classification framework using feature optimization and ensemble learning was proposed. The results of the testing showed that the false positive analysis for the ensemble is significantly lower than that of individual models. To improve malware classification, a self-aware framework was proposed, integrating model routing based on a trust system for feature selection and explainability. The routing logic increases the power of the ensemble with trust-based decisions and provides a flexible mechanism useful for both historical and contemporary malware characteristics. To refine malware classification, a trust-aware adaptive framework is proposed, allowing for feedback-based malware classification. The feedback buffer allows the user to co-correct the model, allowing the user to export/import feedback files. The integration of explainability contributes to the confidence in the solutions, which is increased by more information about the features from a local and global perspective. This adaptive framework is implemented in a developed demo version of a software application called "Shipka Guard". It integrates a trust-aware routing mechanism that uses multiple generations of models, including a fallback mechanism. It uses a feedback buffer and a correction layer for easy adaptation from user corrections and synthetic injections. Thanks to the combination of adaptability, interpretability and scalability, challenges related to the continuous development of practical next-generation malware detection systems that constantly remain "true" to the given threat models, instead of constantly changing, are solved.

Through numerous experiments with different types of data, the practical applicability of both the proposed hybrid algorithms and the developed application under the name "Shipka Guard" has been established.

As a future development of the research in the dissertation work, research related to simultaneous and multi-class identification of malware is planned.

The results obtained on the topic of the dissertation research have been reported at 3 international conferences. The presented results are reflected in a total of 4 scientific publications in editions that are referenced and indexed in world-renowned databases of scientific information – Scopus.

CONTRIBUTIONS

The results obtained, described in this dissertation, can be summarized in the following scientific and scientific-applied contributions:

1. Two mathematical models are proposed, through which a selection of software for a suitable virtual machine can be made for the purposes of experimental testing for malware detection.
2. An improved static analysis approach for malware detection is proposed by optimizing feature extraction by combining different machine learning algorithms. Tests conducted with the proposed hybrid algorithms show better performance.
3. A framework for static malware classification is proposed, which uses feature optimization and ensemble learning. The results show that the false positive analysis for the ensemble is significantly lower than that of individual models.
4. Self-aware malware classification is proposed by routing models based on a trust system for feature selection and explainability. The routing logic increases the power of the ensemble with trust-based decisions and provides a flexible mechanism useful for both historical and contemporary malware characteristics.
5. A trust-aware adaptive framework for malware classification with feedback corrections is proposed. This framework is both adaptive and robust, as it includes a self-aware model classifier that uses adaptive logic to automatically choose between traditional and contemporary model layers by measuring the reliability of the prediction. The integration of explainability contributes to the confidence in the decisions, which is increased by more information about the features from a local and global perspective.

LIST OF PUBLICATIONS ON THE DISSSERTATION

1. **Barzev**, I., Borissova, D.: An improved static analysis approach for malware detection by optimizing feature extraction combining different ML algorithms. In: Bennour, A., Bouridane, A., Almaadeed, S., Bouaziz, B., Edirisinghe, E. (eds) Intelligent Systems and Pattern Recognition. ISPR 2024. Communications in Computer and Information Science, vol. 2304, pp. 102–115, 2025, Springer, Cham. https://doi.org/10.1007/978-3-031-82153-0_8. Print ISBN 978-3-031-82152-3, **SJR Q4=0.182**
2. **Barzev**, I., Borissova, D.: Performance analysis of LSTM, SVM, CNN and CNN-LSTM algorithms for malware detection in IoT dataset. WSEAS TRANSACTIONS on COMPUTER RESEARCH, vol. 13, 288–296, 2025, http://dx.doi.org/10.37394/232018.2025.13.27. E-ISSN: 2415-1521, **SJR Q4=0.140**
3. **Barzev, I.**, Borissova, D., Buhtiyarov, N.: Comparison of different binary classification algorithms for malware detection. In: Rocha, Á., Ferrás, C., Hochstetter Diez, J., Diéguez Rebolledo, M. (eds) Information Technology and Systems. ICITS 2024. Lecture Notes in Networks and Systems, vol. 932, pp. 369–378, 2024, Springer, Cham. https://doi.org/10.1007/978-3-031-54235-0_33. Print ISBN 978-3-031-54234-3, Online ISBN 978-3-031-54235-0, **SJR Q4 =0.17.**
4. Borissova, D., **Barzev, I.**, Yoshinov, R., Kotseva, M.: Group decision-making models for selection of virtual machine software for malware detection purposes. In: Proc. of 12th Mediterranean Conference on Embedded Computing (MECO), Budva, Montenegro, 2023, https://doi.org/10.1109/MECO58584.2023.10155084. ISBN 979-8-3503-2291-0.

Accepted publications:

- Barzev, I., Borissova, D.: Trust-Aware Adaptive Framework for Malware Classification with Feedback Patching. 9th International Conference on Information Technology & Systems, 16-18 February 2026
- Barzev, I., Borissova, D.: A scalable Python Framework for static malware classification using feature optimization and ensemble Learning. International Conference on Advanced Research in Technologies, Information, Innovation, and Sustainability (ARTIIS 2025). 21-23 October 2025.
- Barzev, I., Borissova, D.: Self-aware malware classification via confidence-guided model routing and explainable feature attribution. 5th International Conference on Intelligent Systems and Pattern Recognition, 25-27 September 2025

BIBLIOGRAPHY

1. Ozkan-Okay, M., Akin, E., Aslan, O., Kosunalp, S., Iliev, T., Stoyanov, I., Beloev, I.: A comprehensive survey: Evaluating the efficiency of artificial intelligence and machine learning techniques on cyber security solutions. IEEE Access, vol. 12, pp. 12229-12256, 2024, https://doi.org/10.1109/ACCESS.2024.3355547.
2. Barzev, I., Borissova, D.: Performance analysis of LSTM, SVM, CNN and CNN-LSTM algorithms for malware detection in IoT dataset. WSEAS TRANSACTIONS on COMPUTER RESEARCH, vol. 13, 288-296, 2025, http://dx.doi.org/10.37394/232018.2025.13.27.
3. Barzev, I., Borissova, D., Buhtiyarov, N.: Comparison of different binary classification algorithms for malware detection. In: Rocha, Á., Ferrás, C., Hochstetter Diez, J., Diéguez Rebolledo, M. (eds) Information Technology and Systems. Lecture Notes in Networks and Systems, vol. 932, pp. 369-378, 2024, https://doi.org/10.1007/978-3-031-54235-0_33.
4. Praveen, E. Kumar, S. Priyanka, A comprehensive survey on hardware-assisted malware analysis and primitive techniques, Computer Networks, vol. 235, 109967, 2023, https://doi.org/10.1016/j.comnet.2023.109967.
5. Borissova, D., Barzev, I., Yoshinov, R., Kotseva, M.: Group decision-making models for selection of virtual machine software for malware detection purposes. In: Proc. of 12th Mediterranean Conference on Embedded Computing (MECO), Budva, Montenegro, 2023, https://doi.org/10.1109/MECO58584.2023.10155084.
6. Barzev, I., Borissova, D.: An improved static analysis approach for malware detection by optimizing feature extraction combining different ML algorithms. In: Bennour, A., Bouridane, A., Almaadeed, S., Bouaziz, B., Edirisinghe, E. (eds) Intelligent Systems and Pattern Recognition. ISPR 2024. Communications in Computer and Information Science, vol. 2304, pp. 102–115, 2025, https://doi.org/10.1007/978-3-031-82153-0_8.
7. Ho, T.K., Hull, J.J., Srihari, S.N.: Decision combination in multiple classifier systems. IEEE Trans. Pattern Analysis and Machine Intelligence, vol. 16(1), pp. 66-75, 1994, https://doi.org/10.1109/34.273716.
8. Kittler, J., Hatef, M., Duin, R.P.W., Matas, J.: On combining classifiers. IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 20(3), pp. 226-239, 1998, https://doi.org/10.1109/34.667881.
9. Thai, H.-T.: Machine learning for structural engineering: A state-of-the-art review. Structures, vol. 38, pp. 448-491, 2022, https://doi.org/10.1016/j.istruc.2022.02.003.
10. Hancock, J.T., Khoshgoftaar, T.M. CatBoost for big data: an interdisciplinary review. Journal of Big Data, vol. 7, 94, 2020, https://doi.org/10.1186/s40537-020-00369-8.