Baltic J Baltic J. Modern Computing, Vol. 13 (2025), No. 4, 994-1017
https://doi.org/10.22364/bjmc.2025.13.4.12

Method for Determining a Gradient Boosting Model with Optimal Hyperparameters for Classifying Processes in the Volatile Memory of an Organization's Information System Assets

Mariia HANCHENKO, Serhii GAKHOV

State University of Information and Communication Technologies, Kyiv, Ukraine

hanchenkomariia@gmail.com, gakhov@ukr.net

ORCID 0009-0008-0998-0443, ORCID 0000-0001-9011-8210

Abstract. The objective of the research is to determine the effectiveness of the gradient boosting model for classification processes in the volatile memory of the organization's information system assets. A method for determining the optimal hyperparameters for the AdaBoost, HistGB, XGBoost, and LightGBM models using full search (GridSearchCV) and five-fold cross-validation (StratifiedKFold) is proposed. Before training the models, the CIC-MalMem-2022 dataset underwent preprocessing steps. These steps included selecting attributes, normalizing numerical values, encoding categorical variables, and dividing the dataset into a training and a test set. The performance of models with basic and optimized hyperparameters was evaluated by classification metrics (i.e., accuracy, precision, recall, F1-score) and CPU time. The experimental findings revealed no statistically significant differences in model accuracy. However, a variation in computational performance was observed. The optimized HistGB model achieved the highest classification accuracy (99.9943%) at the lowest time cost (CPU time = 3.38s), demonstrating the best results for classifying processes in volatile memory. The findings substantiate the efficacy of gradient boosting methods in systems designed to identify malicious processes within the volatile memory of the organization's information system assets.

Keywords: Gradient Boosting, HistGradientBoosting, hyperparameter optimization, CIC-MalMem-2022, classification, CPU time, volatile memory.

1. Introduction

The modern cyber threat landscape is undergoing rapid evolution, as evidenced by the growing sophistication of attacks and their ability to bypass traditional defense methods. Fileless malware, which leaves no traces in the file system and operates exclusively in RAM, is a hazardous form of malicious software. These threats can mimic the legitimate activity of system processes, making it difficult to identify with signature-based or heuristic detection methods (Hanchenko and Gakhov, 2024). In this regard, there is an urgent need to develop intelligent detection systems capable of analyzing behavioral features of memory and classifying its processes without relying on known attack patterns.

One approach to improving the efficiency of such systems is to use machine learning methods, particularly those based on behavioral analysis of RAM data. These methods facilitate the identification of anomalies associated with the execution of malicious code, even when it is strategically masked. In this context, selecting an appropriate classification method and calibrating its hyperparameters are particularly important. These measures ensure that the model exhibits the capacity for generalization, adaptability to new scenarios, and effective exploitation of computational resources.

The most effective classification methods currently in use (Dener et al., 2022; Louk et al., 2022; Naeem et al., 2023; Aboanber et al., 2024; Ke et al., 2017) include gradient boosting methods, such as AdaBoost, HistGradientBoosting, XGBoost, and LightGBM (Microsoft, 2025, February 15; scikit-learn developers, 2025, January 10). They demonstrated high accuracy in analyzing large amounts of data and detecting complex patterns. At the same time, the performance of these models is heavily reliant on the correct configuration of hyperparameters, which requires the use of optimization methods, such as full search (GridSearchCV) (scikit-learn developers, 2025, January 10), combined with cross-validation (StratifiedKFold) (scikit-learn developers, 2025, January 10). However, comparative studies of gradient-boosting models for detecting fileless memory-based attacks remain limited.

Given the above, the purpose of this study is to compare gradient boosting models and determine the optimal method for classifying processes in the volatile memory of the organization's information system assets. In this work, we used the CIC-MalMem-2022 dataset (Canadian Institute for Cybersecurity, 2022), which contains characteristics of both safe and harmful processes in volatile memory.

The hyperparameters were optimized using GridSearchCV and five-fold cross-validation, and the models were evaluated in terms of classification accuracy (accuracy, precision, recall, F1-score) and computing costs (CPU time). The results of the study allow us to reasonably select the most effective model for detecting fileless malware in the volatile memory of information system assets.

2. Analysis of scientific references

Fileless malware holds a special place among modern cyber threats because of its ability to bypass traditional detection tools based on signatures, static analysis, or file analysis (Hanchenko and Gakhov, 2024). Unlike classical forms of malware, it leaves no traces in the file system, functions exclusively in volatile memory, and disguises itself as a legitimate system process, as demonstrated by research from Aamir (2022) and ANY.RUN (2024). This makes identification difficult, especially in a corporate environment with many concurrently active processes.

Afreen et al. (2020) and Sanjay et al. (2018) note that fileless attacks are characterized by a high ability to evade detection, which makes them invisible to traditional systems. ReliaQuest's (2023) research indicates that in 2023, 86.2% of critical incidents were related to fileless malware, with Living-off-the-Land techniques used in 26.26% of cases. Nevertheless, the memory analysis of activity has the potential to reveal new ways to identify such threats.

Recent research emphasizes the potential of dynamic RAM analysis as a promising approach for detecting malicious processes based on their behavioral characteristics. For example, in Khalid et al.'s (2022) work, the Volatility tool was used to analyze memory dumps, which enabled the creation of several datasets (VirusShare, AnyRun, PolySwarm, HatchingTriage, JoESandbox) and subsequent classification using machine learning methods.

Of particular note is the CIC-MalMem-2022 dataset (Canadian Institute for Cybersecurity, 2022), which contains balanced samples of benign and malicious processes extracted from memory dumps. Its structure is close to real-world conditions and avoids model retraining.

Due to these advantages, CIC-MalMem-2022 is actively used in behavioral memory analysis research. Louk et al. (2022) demonstrated the use of this set along with BODMAS and open data from Kaggle, obtaining high classification accuracy (>99%) when using gradient boosting models - XGBoost, LightGBM, and CatBoost.

The advantages of ensemble models for detecting complex malicious actions are confirmed by Hasan et al. (2023) and Dener et al. (2022), who report that gradient boosting models achieved an accuracy of over 99.9%. Ramesh et al. (2024) analysis confirms the effectiveness of XGBoost, LightGBM, GBM, and CatBoost in terms of precision, recall, F1-score, and Area Under the Curve (AUC), achieving 99.89% accuracy on the CIC-MalMem-2022 dataset.

A literature review shows that gradient boosting is one of the most effective methods for detecting abnormal activity in memory. Concurrently, the effectiveness of such models depends on the accurate configuration of hyperparameters, including the number of trees, tree depth, learning rate, and level of regularization. Nevertheless, the methodology for optimizing hyperparameters is frequently not fully disclosed. For example, Dener et al. (2022) report high accuracy, but there is no description of the methodology for tuning the model's hyperparameters, making it difficult to repeat the experiment. Louk et al. (2022), on the other hand, use a random search in each range of values (Bergstra and Bengio, 2012), which is a more systematic approach. Thus, there is a need for a method for selecting hyperparameters for gradient boosting models. Using a complete hyperparameter search (GridSearchCV) in combination with multiple cross-validation folds (StratifiedKFold) increases reproducibility and enables a statistically valid performance assessment.

Despite individual studies analyzing gradient boosting models using CIC-MalMem-2022, the literature lacks a comprehensive comparison of gradient boosting method implementations (XGBoost, LightGBM, AdaBoost, HistGB) using identical hyperparameter optimization settings and accounting for computational costs (CPU time). This creates a research niche for the systematic comparison of models that accounts for both classification accuracy and computational resource limitations.

Thus, the analyzed references confirm the relevance of using gradient-boosting ensemble methods for classifying processes in volatile memory. The lack of a systematic approach to comparing models and their parametric optimization determines the scientific novelty and potential practical impact of this research.

3. Methods

The methodology for determining the gradient boosting model for the classification processes of the volatile memory necessitated the consistent resolution of several key tasks: The CIC-MalMem-2022 dataset was selection and preparation was justified (Canadian Institute for Cybersecurity, 2022), as was the selection and building of gradient boosting models - AdaBoost, XGBoost, LightGBM, and HistGB (Microsoft, 2025, February 15; scikit-learn developers, 2025, January 10). The implementation of the method for determining optimal hyperparameters for gradient boosting models was also justified, as was its comprehensive comparison using classification metrics (accuracy, precision, recall, F1-score) and CPU time.

3.1. Justification for the CIC-MalMem-2022 dataset selection

The CIC-MalMem-2022 dataset (Canadian Institute for Cybersecurity, 2022), created by the Canadian Institute for Cybersecurity at the University of New Brunswick, was generated from volatile memory dumps collected in a controlled virtual machine environment (Windows 10, 2 GB RAM) by running both benign programs and malicious samples from VirusTotal (Carrier et al., 2022).

The dataset selection was based on several critical factors. The CIC-MalMem-2022 dataset was published in 2022 (Canadian Institute for Cybersecurity, 2022), thereby ensuring its relevance to modern cyber threats and the behavioral features it captures. The public availability of the dataset (Hanchenko and Gakhov, 2024) enhances the reproducibility of the results, a fundamental criterion for scientific research. The active use of CIC-MalMem-2022 in modern publications (Dener et al., 2022; Louk et al., 2022; Khalid et al., 2022; Neo, 2023) enables a fair comparison with other methods for classifying malicious processes in volatile memory. The CSV data format allows us to effectively work with them in the Jupyter Notebook environment (Project Jupyter, 2014) using the pandas and csv Python libraries.

Class balancing - 29,298 each of malware and safe software samples (Canadian Institute for Cybersecurity, 2022) guarantees an equal distribution of data, which eliminates the additional need for class balancing and minimizes the risk of prejudice of gradient boosting models towards the dominant class (Carrier et al., 2022; Canadian Institute for Cybersecurity, 2022; Dener et al., 2022). The dataset contains attributes that represent the behavioral characteristics of 15 different malware families that focus on activity in the volatile memory (see Tables 1 and 2) (Carrier et al., 2022; Canadian Institute for Cybersecurity, 2022).

All software samples in the CIC-MalMem-2022 dataset have full vector descriptions (see Table 2), which eliminates the need for imputation methods.

Table 1. Malware class samples in the CIC-MalMem-2022 dataset

Malware category	Malware family	Number of samples of the malware class
Spyware	180Solutions	200
	Coolwebsearch	200
	Gator	200
	Transponder	241
	TIBS	141
Ransomware	Conti	200
	MAZE	195
	Pysa	171
	Ako	200
	Shade	220
Trojan horses	Zeus	195
	Emotet	196
	Refroso	200
	Scar	200
	Reconyc	157

Table 2. Software sample attributes in the CIC-MalMem-2022 dataset

№	Software sample attribute	Description
1	Category	Category
2	pslist. nproc	Total number of processes
3	pslist. nppid	Total number of parent processes
4	pslist.avg_threads	Average number of threads for the process
5	pslist.nprocs64bit	Total number of 64-bit processes
6	pslist.avg_handlers	Average number of handlers
7	dllist.ndlls	Total number of loaded libraries for every process
8	dllist.avg_dlls_per_proc	Average number of loaded libraries per process
9	handles.nhandles	Total number of opened handles
10	handles.avg_handles_per_proc	Average number of handles per process
11	handles.nport	Total number of port handles
12	handles.nfile	Total number of file handles
13	handles.nevent	Total number of event handles
14	handles.ndesktop	Total number of desktop handles
15	handles.nkey	Total number of key handles
16	handles.nthread	Total number of thread handles
17	handles.ndirectory	Total number of directory handles

18	handles.nsemaphore	Total number of semaphore handles
19	handles.ntimer	Total number of timer handles
20	handles.nsection	Total number of section handles
21	handles.nmutant	Total number of mutant handles
22	ldrmodules.not_in_load	Total number of modules missing from the load list
23	ldrmodules.not_in_init	Total number of modules missing from the init list
24	ldrmodules.not_in_mem	Total number of modules missing from the memory list
25	ldrmodules.not_in_load_avg	The average number of modules missing from the load list
26	ldrmodules.not_in_init_avg	The average amount of modules missing from the init list
27	ldrmodules.not_in_mem_avg	The average number of modules missing from the memory
28	malfind.ninjections	Total number of hidden code injections
29	malfind.commitCharge	Total number of Commit Charges
30	malfind.protection	Total number of protection
31	malfind.uniqueInjections	Total number of unique injections
32	psxview.not_in_pslist	Total number of processes not found in the pslist
33	psxview.not_in_eprocess_pool	Total number of processes not found in the psscan
34	psxview.not_in_ethread_pool	Total number of processes not found in the thrdproc
35	psxview.not_in_pspcid_list	Total number of processes not found in the pspcid
36	psxview.not_in_csrss_handles	Total number of processes not found in the csrss
37	psxview. not_in_session	Total number of processes not found in the session
38	psxview. not_in_deskthrd	Total number of processes not found in the desktrd
39	psxview.not_in_pslist_false_avg	Average false ratio of the process list
40	psxview.not_in_eprocess_pool_false_avg	Average false ratio of the process scan
41	psxview.not_in_ethread_pool_false_avg	Average false ratio of the third process
42	psxview.not_in_pspcid_list_false_avg	Average false ratio of the process id
43	psxview.not_in_csrss_handles_false_avg	Average false ratio of the csrss
44	psxview.not_in_session_false_avg	Average false ratio of the session
45	psxview.not_in_deskthrd_false_avg	Average false ratio of the deskthrd
46	modules.nmodules	Total number of modules
47	svcscan.nservices	Total number of services
48	svcscan.kernel_drivers	Total number of kernel drivers
49	svcscan.fs_drivers	Total number of file system drivers

50	svcscan.process_services	Total number of Windows 32-bit owned processes
51	svcscan.shared_process_services	Total number of Windows 32 shared processes
52	svcscan.interactive_process_services	Total number of interactive service processes
53	svcscan.nactive	Total number of actively running service processes
54	callbacks.ncallbacks	Total number of callbacks
55	callbacks.nanonymous	Total number of unknown processes
56	callbacks.ngeneric	Total number of generic processes
57	Class	Benign or Malware

3.2. CIC-MalMem-2022 dataset preparation

To reduce the computational workload and simplify the gradient boosting models, we removed attributes with constant zero values: pslist.nprocs64bit, handles.nport, and svcscan.interactive_process_services (see Table 2).

Considering that the numerical characteristics of the samples covered a wide range of values (Canadian Institute for Cybersecurity, 2022), all numerical features were normalized to the interval [0, 1] using the function normalize() from the scikit-learn library (scikit-learn developers, 2025), which helped to improve the efficiency of model training.

The target variable Class, which has a categorical type (Malware/Benign), was encoded into a numerical format (0 - Malware, 1 - Benign) using LabelEncoder() from the scikit-learn library (scikit-learn developers, 2012), which made possible its use as an initial feature in the task of classifying processes in the volatile memory of the organization's information system assets.

To perform an objective assessment of the accuracy of the models on unknown data, and to mitigate the risk of overfitting of the entire set, the function train_test_split (test_size = 0.3, random_state = 2025) was employed to divide the set into a training (70%) and a test (30%) sample. The random_state parameter was set to 2025, thus ensuring reproducibility of the results.

3.3. Justification for gradient boosting method selection

As demonstrated (Prashant, 2020, July 15), gradient boosting is an ensemble machine learning method that is highly efficient in classification tasks, including malware and network attack detection, and network traffic analysis (Dener et al., 2022; Louk et al., 2022; Naeem et al., 2023; Aboanber et al., 2024; Ke et al., 2017). The central concept of the method is to sequentially train weak models (decision trees) with a focus on correcting failures of previous iterations (Prashant, 2020, December 8; Prashant, 2020, June 30). This results in a generalized model with high classification accuracy (Rathi, 2019), which is critical in the cybersecurity field, where even a minor fault can have serious consequences.

The four popular implementations of gradient boosting were selected for the study: HistGB, LightGBM, XGBoost, and AdaBoost. Their choice was based on both technical characteristics that meet the requirements for processing the CIC-MalMem-2022 dataset and their active use in modern research. Table 3 compares the key features of each method, including performance with large datasets, handling of missing values, parallel computing capabilities, hyperparameter customization, regularization, and more.

HistGB is an optimized implementation of gradient boosting in scikit-learn that discretizes continuous features using histograms, significantly reducing the number of splits when building a tree (Brownlee, 2020; Bhimani, 2022). High speed, low memory consumption, and the ability to handle missing values without imputation make HistGB an effective tool for tasks involving large tabular datasets and the best choice for our research.

LightGBM (Prashant, 2020, July 21) is a method from Microsoft (2025, February 15) that uses GOSS (Gradient-based One-Side Sampling) and EFB (Exclusive Feature Bundling) optimizations (Prokhorenkova et al., 2017) to speed up model training. It supports handling missing values and delivers high performance and efficient resource use, which is especially important in our constrained computing environment (16 GB of RAM, AMD Ryzen 7 5700U 1.80 GHz).

XGBoost is one of the most widely used implementations of gradient boosting, supporting regularization (L1, L2) (scikit-learn developers, 2025, June 5; Trotta, 2017), parallel training, and built-in cross-validation (Chen et al., 2016). The high accuracy of the model and its flexible settings have made XGBoost a standard for solving classification problems (Louk et al., 2022).

Table 3. Gradient boosting method technical characteristics comparison

Method characteristic	**Characteristic presence in the method**			
	HistGB	**LightGB**	**XGBoost**	**AdaBoost**
Large amounts of data processing	+	+	+	–
Missing value support	+	+	+	–
Optimal memory utilization	+	+	±[1]	–
Parallel training	+	+	+	–
Hyperparameters configuration option	+	+	+	+
Regularization (L1, L2)	+	+	+	+
Cross-validation built-in support	+	+	+	+
Training speed	+	+	+	–
Classification accuracy	+	+	+	+

[1] — requires hyperparameter optimization for efficient use of memory resources. “+” indicates the presence or high efficiency of the corresponding characteristic. “-” indicates the absence or low efficiency. “±” indicates partial support or dependence of the efficiency on the setting conditions.

AdaBoost is a classic boosting method that adjusts the weights of misclassified samples at each iteration (scikit-learn developers, 2025, January 10; Baladram, 2024). Despite its simpler architecture and slower speed compared to other methods, AdaBoost has a low tendency to overfit (Baladram, 2024), making it a suitable baseline for comparison with more modern realizations.

3.4. Building gradient boosting models

All experiments were conducted in the Jupyter Notebook environment (Project Jupyter, 2014) using the Python programming language. The pandas, numpy, seaborn, time, scikit-learn, lightgbm, xgboost, and csv libraries were used for data processing, visualization, and model building. All the considered gradient boosting methods - HistGB, LightGBM, XGBoost, and AdaBoost - have implementations in these libraries (Microsoft, 2025, February 15; scikit-learn developers, 2025, January 10; XGBoost Contributors, 2021; Chen et al, 2016), which made it possible to perform calculations on a local machine, even with limited resources.

Experimental hardware environment configuration:

– Processor: AMD Ryzen 7 5700U with Radeon Graphics, 1.80 GHz
– RAM: 16.0 GB (available: 15.4 GB)
– OS type: 64-bit, x64 architecture
– Operating system: Windows 11 Home, version 23H2
– OS build: 22631.5039

Before starting the training process, the data from the CIC-MalMem-2022 dataset was divided into a training set and a test set, as outlined in Section 3.2. The models were trained on the first subset, and their accuracy was then evaluated on the second.

For each method (i.e., HistGB, LightGBM, XGBoost, and AdaBoost), the model was trained in two stages. First, the default parameters were used; second, the hyperparameters were optimized (Prashant, 2020, July 15; Microsoft, 2025, February 14) using GridSearchCV (see Section 3.5).

In the initial phase, the models were trained with default parameter values, thereby establishing baseline performance before implementing any optimization procedures. The default parameter values are listed in Table 4.

Table 4. Gradient boosting model default parameter values

Model parameters	**HistGB**	**LightGBM**	**XGBoost**	**AdaBoost**
random_state	0	0	0	0
l2_regularization / reg_lambda	0	0	1	-
learning_rate	0.1	0	0.5	1
max_depth	None	-1	6	-
max_iter / n_estimators	100	100	100	50
min_samples_leaf / min_child_samples / min_child_weight	20	20	1	-

3.5. Method for determining optimal hyperparameters

Gradient boosting methods, including HistGB, LightGBM, XGBoost, and AdaBoost, offer a broad spectrum of hyperparameters, whose configurations directly influence the effectiveness of the developed models (scikit-learn developers, 2025, January 10; Chen, 2016; Rani et al., 2023). In this research, we selected the key hyperparameters that have the most significant impact on the process of training a generalization of models were selected: the number of trees in the ensemble (n_estimators / max_iter), the learning rate (learning_rate), the maximum tree depth (max_depth), the minimum number of samples per leaf (min_samples_leaf, min_child_weight, min_child_samples), and regularization coefficients (l2_regularization / reg_lambda).

The value space used to search for optimal hyperparameter combinations is shown in Table 5. The model is formulated with consideration of the characteristics inherent to each implementation of the gradient boosting method (Microsoft, 2025, February 15; scikit-learn developers, 2025, January 10; XGBoost Contributors, 2021), as well as the specifics of the CIC-MalMem-2022 dataset (Canadian Institute for Cybersecurity, 2022).

Table 5. Values space for optimal hyperparameters search

Hyperparameter	**Values space**
max_iter/ n_estimators	[100, 250, 500, 700, 1000]
learning_rate	[0.01, 0.05, 0.1]
max_depth	[3, 5, 10]
min_samples_leaf/ min_child_samples/ min_child_weight	[1, 5, 10]
l2_regularization/ reg_lambda	[0, 0.01, 0.1, 1]

The automated search for optimal hyperparameter values was conducted using the GridSearchCV() method from the scikit-learn library (scikit-learn developers, 2025, January 10). Accuracy was used as an evaluation metric, thereby enabling direct measurement of the model's capacity to classify malware and benign software samples. To maintain a proportionate representation of the target classes during training, a 5-fold cross-validation was performed using the StratifiedKFold() object (scikit-learn developers, 2025, January 10).

To ensure reproducibility of results across all experiments, the value of the random_state parameter was unified (2025), and to speed up the learning process, parallel computations were enabled with n_jobs = -1 (scikit-learn developers, 2025, September 9).

The algorithm for determining the optimal hyperparameters of gradient boosting models for classifying processes in the volatile memory of the organization's information system assets is shown in Figure 1.

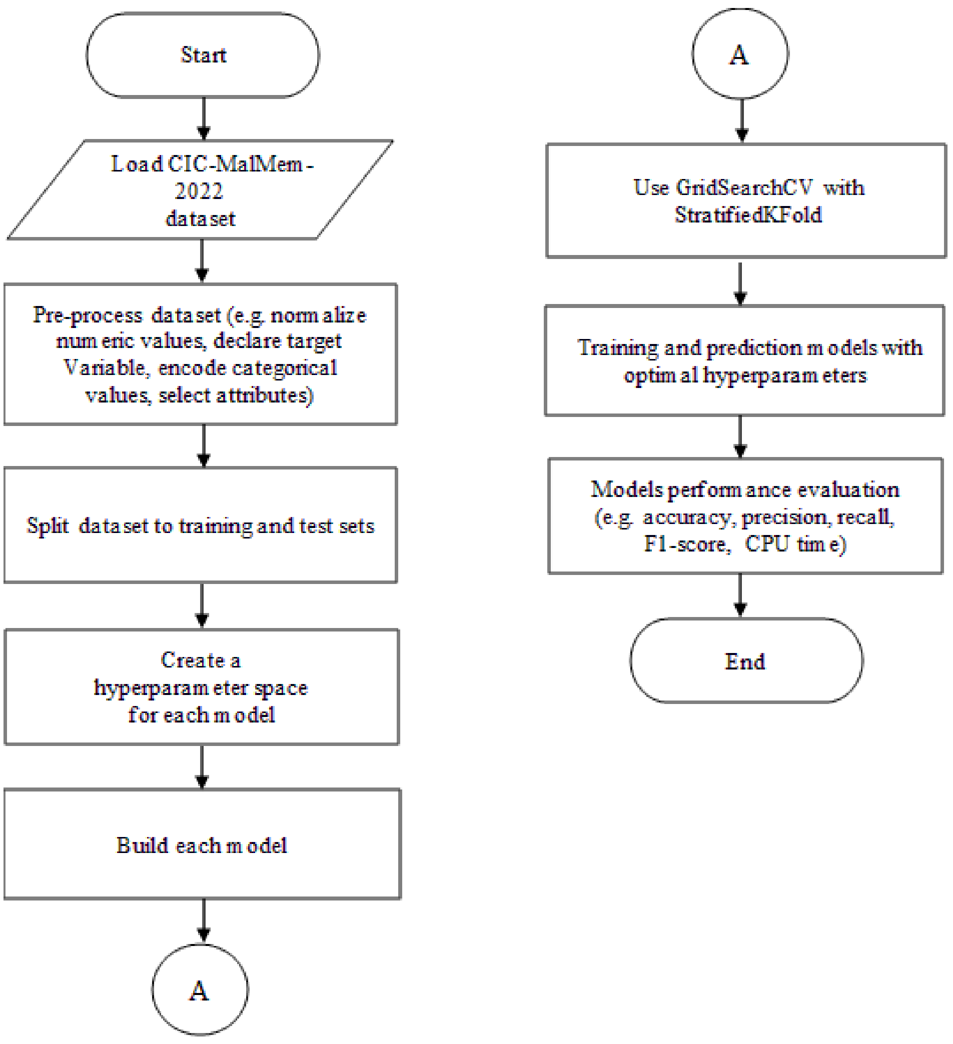
Start
Load CIC-MalMem-2022 dataset
Pre-process dataset (e.g. normalize numeric values, declare target Variable, encode categorical values, select attributes)
Split dataset to training and test sets
Create a hyperparameter space for each model
Build each model
A
A
Use GridSearchCV with StratifiedKFold
Training and prediction models with optimal hyperparameters
Models performance evaluation (e.g. accuracy, precision, recall, F1-score, CPU time)
End

Figure 1. Flow chart of the method for determining the optimal hyperparameters of gradient boosting models for classifying processes in the volatile memory of the organization's information system assets

4. Results

The experimental research evaluated the effectiveness of four gradient boosting models - HistGB, LightGBM, XGBoost, and AdaBoost - in the task of classifying both malware and benign software samples. The models were compared based on classification metrics (accuracy, precision, recall, F1-score) and computation time (CPU time), both for models with default parameters and with optimal hyperparameters.

4.1. Performance comparison of models with default parameters

According to Table 6, the highest test accuracy of 99.9943% was achieved by the HistGB model, and the lowest by AdaBoost at 99.9716%. A similar tendency can be observed for the classification metrics - precision, recall, and F1-score.

Table 6. Performance evaluation results of gradient boosting models with default parameters

Classification metrics	HistGB	LightGBM	XGBoost	AdaBoost
Accuracy on a training set, %	99.9756	99.9756	99.9756	99.9610
Accuracy on a test set, %	99.9943	99.9886	99.9829	99.9716
Precision, %	99.9942	99.9885	99.9827	99.9713
Recall, %	99.9944	99.9888	99.9832	99.9718
F1-score, %	99.9943	99.9886	99.9829	99.9716
CPU time, s	6.28	10	7.69	32.1

Figure 2 shows a bar chart comparing the main metrics for classifying gradient boosting models.

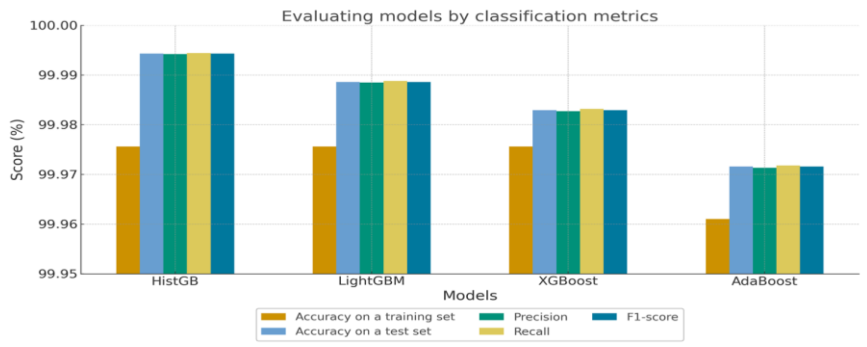

Figure 2. Bar chart comparing model performance with default parameters

Figure 3 shows a radar chart that demonstrates a comprehensive comparison of models across all key metrics, including computational efficiency (CPU time). The CPU time has been normalized and inverted for better visualization: lower CPU time corresponds to greater distance from the center.

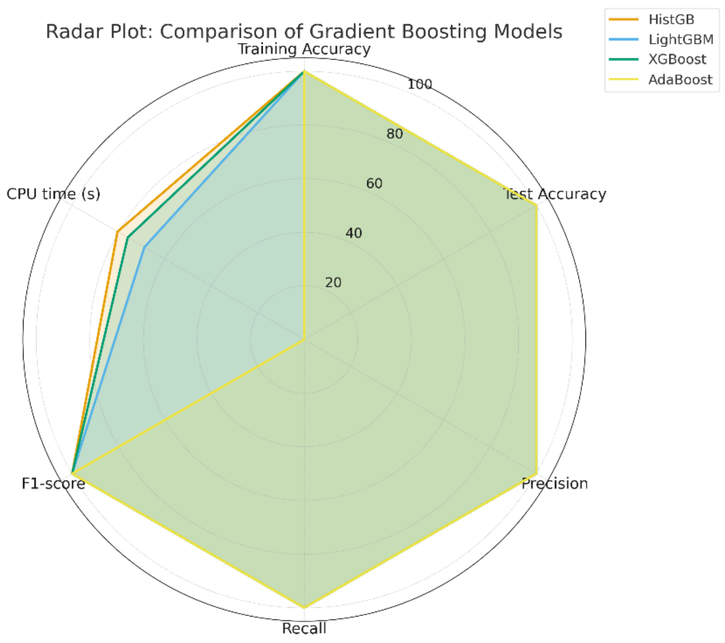

Figure 3. Radar chart comparing models with default parameters

In terms of performance, the fastest model was HistGB with a runtime of 6.28 s, followed by XGBoost (7.69 s) and LightGBM (10.0 s). The most resource-intensive model was AdaBoost, which required 32.1 s for training and prediction. Thus, all four models with default parameters achieved high-quality classification, but HistGB and XGBoost provided a better balance between accuracy and speed.

4.2. Search results for optimal hyperparameters

As illustrated in Table 7, the optimal values for the hyperparameters of the gradient boosting models were determined through five-fold cross-validation, utilizing the GridSearchCV() function (refer to Section 3.5 for further details).

Table 7 shows that the optimal value of the regularization hyperparameter (l2_regularization / reg_lambda) was 0.01 for the HistGB, LightGBM, and XGBoost models. This indicates a moderate level of regularization, balancing model complexity and overfitting.

Table 7. Gradient boosting models' optimal hyperparameters

Model hyperparameters	**HistGB**	**LightGBM**	**XGBoost**	**AdaBoost**
random_state	0	0	0	0
l2_regularization / reg_lambda	0.01	0.01	0.01	-
learning_rate	0.1	0.1	0.05	0.1
max_depth	3	3	3	-
max_iter / n_estimators	700	1000	1000	1000
min_samples_leaf / min_child_samples / min_child_weight	1	5	1	-

The learning_rate hyperparameter was set to 0.1 for HistGB, LightGBM, and AdaBoost, while 0.05 proved more effective for XGBoost. This confirms the XGBoost model's sensitivity to hyperparameter changes and its suitability for gradual learning.

The maximum tree depth (max_depth) was three across all models. This level of complexity proved to be sufficient for effective classification in the considered dataset.

The number of iterations (max_iter / n_estimators) varied from 700 for HistGB to 1000 for LightGBM, XGBoost, and AdaBoost. This indicates that the last models need more decision trees to achieve high accuracy.

Regarding the minimum leaf size (min_samples_leaf/min_child_samples/min_child_weight), HistGB and XGBoost achieved the best results at 1, while LightGBM achieved the best results at 5. This may indicate that LightGBM is more sensitive to overfitting if the tree structure is too small.

In the case of AdaBoost, some of these parameters (regularization, tree depth, minimum leaf size) are not relevant due to the specifics of the method implementation, so their values are marked as "-".

4.3. Performance comparison of models with optimal hyperparameters

As illustrated in Table 8, the results of evaluating the optimized models' performance are shown. These models were evaluated using the same classification metrics as those used for the models with default parameters (see Table 6).

According to Table 8, the HistGB, LightGBM, and XGBoost models achieved extremely high classification rates, including 99.9943% accuracy, precision, recall, and F1-score. This finding suggests that the models are capable of practical training and generalization on new data following hyperparameter optimization.

Figure 4 shows a bar chart for visual comparison of the main classification quality metrics across models, while Figure 5 illustrates the same metrics in a radar chart.

However, CPU time was found to be the critical factor in distinguishing the models in terms of practicality. As demonstrated by the visualisations, HistGB offers the most expeditious training and prediction at 3.38s, while AdaBoost requires an extensive 682 s, which is almost 200 times slower.

Therefore, when considering classification quality and execution time, the HistGB model provides the most balanced solution for identifying malevolent processes in the volatile memory of the organization's information system assets.

Table 8. Performance evaluation results of the optimized models

Classification metrics	HistGB	LightGBM	XGBoost	AdaBoost
Accuracy on a training set, %	99.9805	99.9805	99.9707	99.9829
Accuracy on a test set, %	99.9943	99.9943	99.9943	99.9829
Precision, %	99.9942	99.9942	99.9942	99.9827
Recall, %	99.9944	99.9944	99.9944	99.9832
F1-score, %	99.9943	99.9943	99.9943	99.9829
CPU time, s	3.38	54.5	62	682

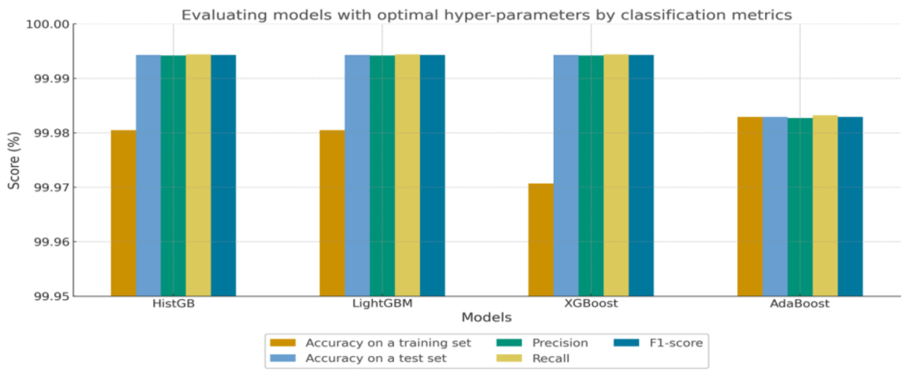

Figure 4. Bar chart of optimized models' performance comparison

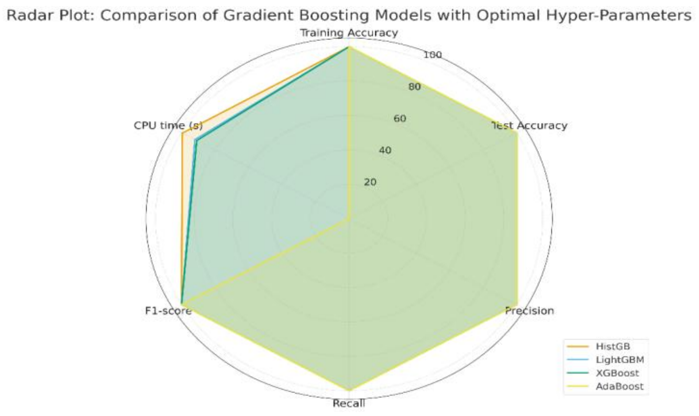

Figure 5. Radar chart of optimized models' performance comparison

4.4. Performance comparison of gradient boosting models with default parameters and optimal hyperparameters

Table 9 presents the confusion matrices for each gradient boosting model before and after hyperparameter optimization. The visual representation of these findings is delineated in Figures 6 and 7.

Table 9. Gradient boosting model confusion matrices

Gradient boosting model		True-Positives (TP)	True-Negatives (TN)	False-Positives (FP)	False-Negatives (FN)
Model with default parameters	HistGB	8665	8913	0	1
	LightGBM	8665	8912	0	2
	XGBoost	8665	8911	0	3
	AdaBoost	8664	8910	1	4
Model with optimal hyperparameters	HistGB	8665	8913	0	1
	LightGBM	8665	8913	0	1
	XGBoost	8665	8913	0	1
	AdaBoost	8665	8911	0	3

According to the results in Table 9 and Figures 6-7, all models achieved zero False-Positive (FP) classifications after optimization, indicating their ability to identify negative processes in volatile memory without false positives accurately.

The lowest number of False-Negative (FN) classifications before optimization was observed with the HistGB and LightGBM models, indicating high sensitivity. At the same time, the AdaBoost model had the worst result, with 4 FN and 1 FP.

After optimization, there was an overall improvement in detection accuracy; notably, the XGBoost model reduced the number of FN from 3 to 1, achieving results identical to those of HistGB and LightGBM. This indicates the positive impact of hyperparameter optimization on XGBoost's ability to correctly classify positive classes without incurring losses. The AdaBoost model, while improving TP and TN, remained least effective at reducing FN, limiting its usefulness in tasks where detection is critical for all malware samples.

The hyperparameter optimization provided a stable reduction in classification failures and increased the reliability of the models in detecting actual positive samples.

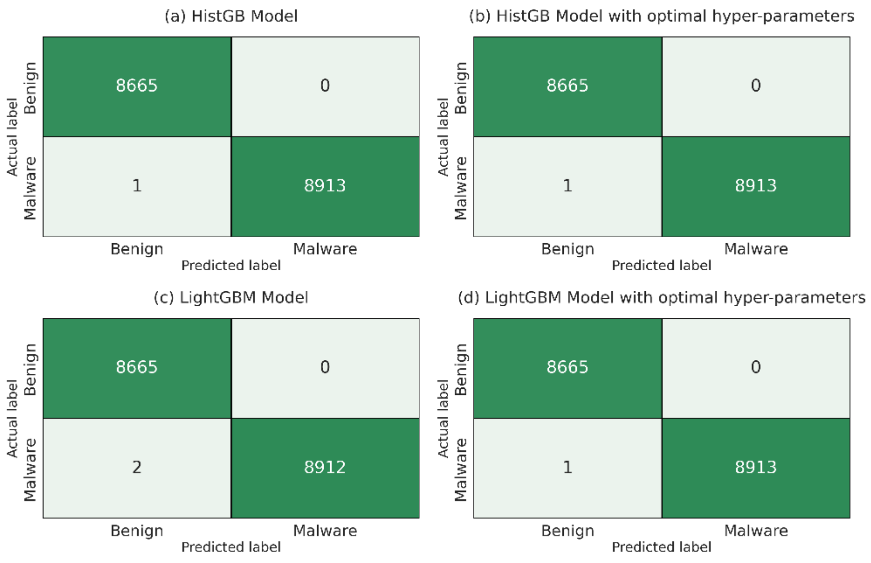

Figure 6. HistGB and LightGBM confusion matrices before and after optimization

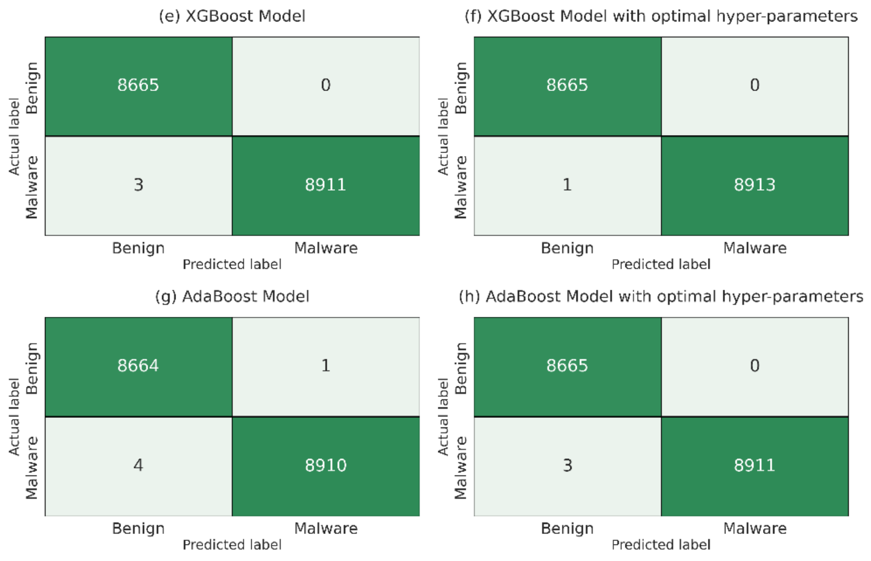

Figure 7. XGBoost and AdaBoost confusion matrices before and after optimization

4.5. Performance comparison of the models before and after hyperparameter optimization in terms of recall and AUC

Figure 8 illustrates the Receiver Operating Characteristic (ROC) curves for the gradient boosting models before and after hyperparameter optimization. All the curves approach the upper-left corner of the diagram, indicating a high True Positive Rate (TPR) and a low False Positive Rate (FPR). This confirms the models' ability to detect malware with a minimum number of false positives effectively.

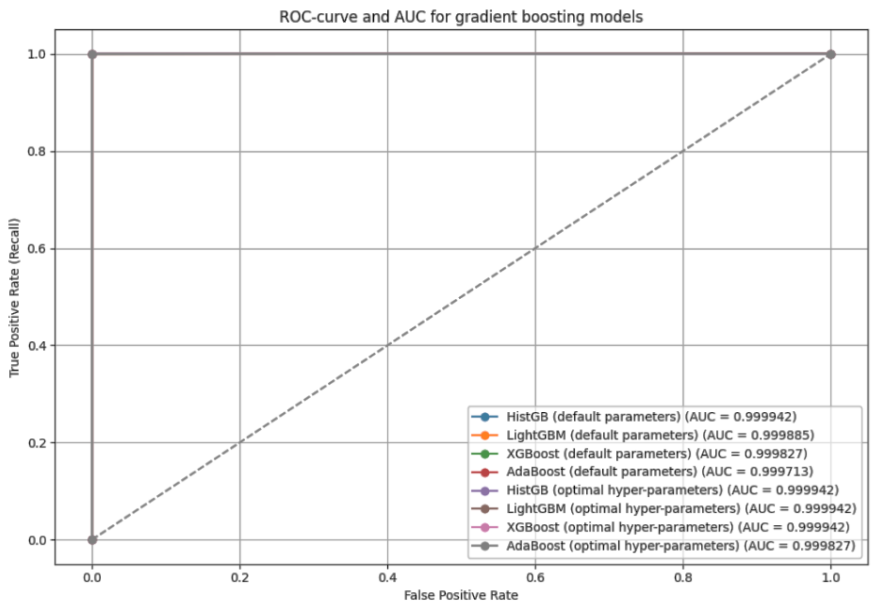

Figure 8. ROC curve and AUC values for models with default parameters and optimized hyperparameters

The respective AUC values for all models, in the default configuration and after optimization, are greater than 99.9%. This indicates the models' almost perfect ability to distinguish between malware and benign software classes. In particular, the HistGB, LightGBM, and XGBoost models achieved an AUC of 0.999942 after hyperparameter tuning, the highest among all variants.

Therefore, regardless of the hyperparameters, all models demonstrate exceptionally high sensitivity (recall) and classification performance, but optimizing hyperparameters ensures even greater consistency and reliability, particularly in critical areas where minimizing False-Negative results is important.

4.6. Model performance comparison (CPU time) before and after hyperparameter optimization

Tables 6 and 8, and Figure 9, illustrate the comparative performance of the gradient boosting models in terms of CPU time during training and prediction. The analysis covers both configurations - with default parameters and with optimal hyperparameters.

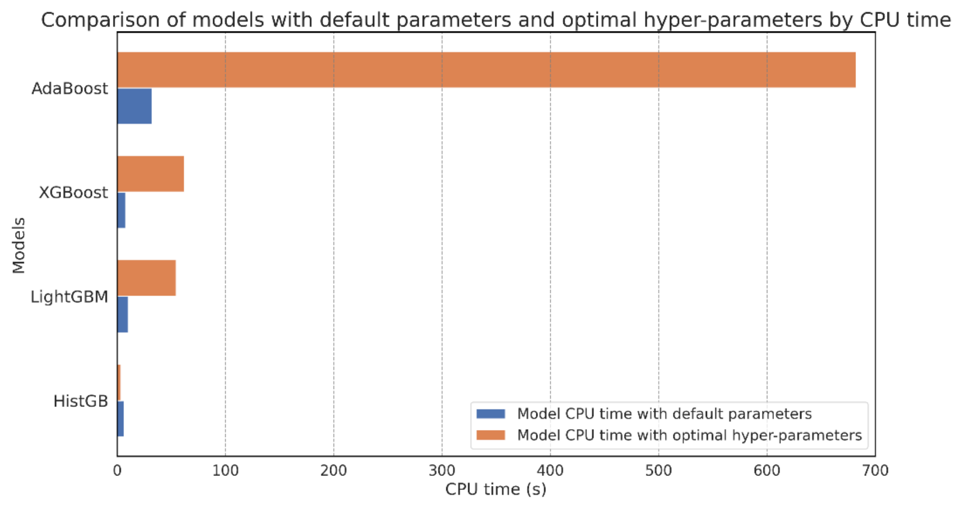

Figure 9. Bar chart of model performance comparison before and after hyperparameter optimization

According to Table 6, the HistGB, LightGBM, and XGBoost models in the default configuration showed comparatively low CPU time - 6.28 s, 10.0 s, and 7.69 s, respectively. Meanwhile, the AdaBoost model was the slowest at 32.1 s, which is significantly higher than the other methods.

After optimizing the hyperparameters (see Table 8), the models' performance changed. The HistGB model reduced CPU time to 3.38 s, indicating improved computational performance. On the other hand, for LightGBM and XGBoost, the computation time increased to 54.5s and 62s, respectively, likely due to increased model complexity after optimization. The most significant increase in time was observed in the AdaBoost model, which reached 682s, the highest among all the considered models.

Therefore, the HistGB model not only maintained high classification accuracy but also demonstrated the best computational performance after optimization, unlike other models that required more resources with a minor improvement in metrics.

5. Discussion

The results of the research have demonstrated the high efficiency of gradient boosting models for classifying processes in volatile memory using the CIC-MalMem-2022 dataset (Canadian Institute for Cybersecurity, 2022). In particular, the HistGB and LightGBM models demonstrated the best correlation between classification quality (accuracy, recall, AUC) and computational costs (CPU time), outperforming XGBoost and AdaBoost. This assertion is further demonstrated by the visualizations of the models' multidimensional performance profiles in Figures 3 and 5.

The selected CIC-MalMem-2022 dataset is one of the most modern and representative datasets for detecting fileless attacks, as it contains actual examples of malicious and benign processes in volatile memory (Dener et al., 2022; Louk et al., 2022; Canadian Institute for Cybersecurity, 2022). It maintains class balance and enables the formation of reliable behavioral classification models. At the same time, it should be noted that in real-world environments, malware behavior can change significantly through the use of new evasion techniques to avoid detection, which can limit the generalizability of results (Aqua Security, 2023; Fortinet, 2025).

During the experiment, special attention was paid to the impact of hyperparameters on model performance. Parameters such as max_depth, max_iter/n_estimators, learning_rate, min_samples_leaf/min_child_samples/min_child_weight affected both classification accuracy and computational resources (scikit-learn developers, 2025, January 10; Microsoft, 2025, February 14; Chen, 2016). The results showed that optimization of hyperparameters can both improve performance (in particular for HistGB) and significantly increase computing time (especially for AdaBoost, where CPU time increased to 682 s). At the same time, the limited range of values used in the GridSearchCV procedure may have affected the optimality of some model configurations.

CPU time for training and prediction was a key factor in applying the models as real-time, fileless attack detection systems. In this context, the HistGB model, after optimization, was the most efficient, achieving 99.9943% accuracy and 3.38 seconds of CPU time, making it suitable for practical implementation. Although the AdaBoost model retained high accuracy, it proved too resource-intensive, requiring more than 11 minutes of CPU time, making it unsuitable for operational systems without prior optimization and complexity reduction.

Notably, the CPU time measurements did not include time spent on hyperparameter selection, and the calculations were performed in a local environment, which could affect the results due to hardware limitations. In real-world applications, model performance may differ depending on the platform configuration (local, cloud, containerized, etc.).

Generally, the results obtained are consistent with current trends in the scientific literature (Dener et al., 2022; Louk et al., 2022; Ramesh et al., 2024), where gradient boosting methods are considered effective in detecting harmful processes in the volatile memory of the organization's information system assets. At the same time, differences in datasets, model implementations, hyperparameters, and execution environments may account for some of the discrepancies with other research.

The practical value of this research is to demonstrate the suitability of the HistGB and LightGBM models for integration into fileless attack protection systems, where both accuracy and speed are critical. Their implementation will allow detecting malicious processes in the volatile memory, enabling detection of in-memory threats within a few seconds, making them suitable for near-real-time applications in operational systems.

Further research should consider the full cost cycle, including hyperparameter optimization, before deploying models in production environments. Perspective areas include performance studies in cloud (Condon, 2024; Aqua Security, 2023) or containerized environments (Hanchenko and Gakhov, 2024; Rani et. al., 2023), testing on different datasets or real attack scenarios (Brad, 2024), and comparison with other types of models, such as neural networks or hybrid models.

6. Conclusions

This research has compared four popular gradient boosting methods - HistGB, LightGBM, XGBoost, and AdaBoost - to classify processes in the volatile memory of the asset based on the CIC-MalMem-2022 dataset. All models demonstrated high classification accuracy (accuracy from 99.97% to 99.99%), confirming the suitability of the gradient boosting method for analyzing processes in the volatile memory.

The highest accuracy across all primary metrics was achieved by the optimized HistGB, LightGBM, and XGBoost models, which successfully generalized the behavioral patterns of the software samples. The HistGB model with optimized hyperparameters appeared to be the most balanced in terms of accuracy (accuracy = 99.9943%) and computational performance (CPU time = 3.38 s). This provides a perspective for implementing systems to detect malicious processes in the volatile memory of assets with limited time resources.

An important result of the research is the proposed method for determining the optimal hyperparameters of gradient boosting models via controlled parameter selection (max_depth, learning_rate, n_estimators, min_samples_leaf, etc.) using GridSearchCV in combination with StratifiedKFold cross-validation. This approach ensures the best possible consistency between classification accuracy and time costs, which is critical when models are deployed in real-world information system environments.

Therefore, the research results confirm the feasibility of using HistGB and LightGBM as practical tools for building intelligent cybersecurity systems. They can identify malware based on behavioral features in volatile memory with high accuracy and low computational cost. Further research should focus on evaluating the performance of the models in cloud and containerized environments, as well as on their adaptation to real-time, complex, fileless attack scenarios in corporate infrastructure.

References

Aamir, L. (2022). *Fileless malware: What it is and how it works*. Retrieved June 7, 2025, from https://www.fortinet.com/blog/industry-trends/fileless-malware-what-it-is-and-how-it-works

Afreen, A., Aslam, M., Ahmed, S. (2020). Analysis of fileless malware and its evasive behavior. In *Proceedings of the 2020 International Conference on Cyber Warfare and Security*

(ICCWS) (pp. 1–8). IEEE. Available at https://ieeexplore.ieee.org/abstract/document/9292376

ANY.RUN (2024). *Fileless malware: Everything you need to know*. Retrieved June 7, 2025. Available at https://any.run/cybersecurity-blog/fileless-malware/

Aqua Security (2023). *2023 Cloud Native Threat Report* [White paper]. https://info.aquasec.com/2023-cloud-native-threat-report

Baladram, S. (2024). *AdaBoost classifier explained: A visual guide with code examples*. Medium. Retrieved June 7, 2025, from https://medium.com/data-science/adaboost-classifier-explained-a-visual-guide-with-code-examples-fc0f25326d7b

Barocas, S., Hardt, M., Narayanan, A. (2023). *Fairness and machine learning: Limitations and opportunities*. MIT Press.

Bergstra, J., Bengio, Y. (2012). Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, **13**, 281–305. https://dl.acm.org/doi/abs/10.5555/2188385.2188395

Bertsekas, D. P. (1976). Multiplier methods: A survey. *Automatica*, **12**(2), 133–145.

Bhimani, D. (2022). *Histogram Boosting Gradient Classifier*. Analytics Vidhya. Retrieved June 7, 2025, from https://www.analyticsvidhya.com/blog/ 2022/01/histogram-boosting-gradient-classifier/

Brad, L. (2024). *Fileless Malware Will Beat Your EDR*. Retrieved June 7, 2025, from https://blog.morphisec.com/fileless-malware-attacks

Brownlee, J. (2020). *Histogram-Based Gradient Boosting Ensembles for Machine Learning*. Machine Learning Mastery. Retrieved June 7, 2025, from https://machinelearningmastery.com/histogram-based-gradient-boosting-ensembles/

Canadian Institute for Cybersecurity (2022). *CIC-MalMem-2022 Dataset*. University of New Brunswick. Retrieved June 7, 2025, from https://www.unb.ca/cic/datasets/malmem-2022.html

Carrier, T., Victor, P., Tekeoglu, A., Lashkari, A. (2022). Detecting obfuscated malware using memory feature engineering. In *Proceedings of the 8th International Conference on Information Systems Security and Privacy* (pp. 177–188). SciTePress. https://doi.org/10.5220/0010908200003120

Chen, T., Guestrin, C. (2016). *XGBoost documentation*. Read the Docs. Retrieved June 7, 2025, from https://xgboost.readthedocs.io/

Condon, S. (2024). *Fileless attacks surge as cybercriminals evade cloud security defenses*. CSO Online. Retrieved June 7, 2025, from https://www.csoonline. com/article/643356/fileless-attacks-surge-as-cybercriminals-evade-cloud-security-defenses.html

Dener, M., Ok, G., Orman, A. (2022). Malware Detection Using Memory Analysis Data in Big Data Environment. *Applied Sciences, 12*(17), 8604. https://doi.org/10.3390/app12178604

Fortinet (2025). *Fileless malware*. Retrieved June 7, 2025, from https://www.fortinet.com/resources/cyberglossary/fileless-malware

Hanchenko, M., Gakhov, S. (2024, April 2026). Analysis of methods for detecting fileless malware in the energy-dependent memory of an organisation's information system assets. In *V International Scientific and Practical Conference «Ricerche Scientifiche e Metodi Della Loro Realizzazione: Esperienza Mondiale e Realtà Domestiche»: Collection of Scientific Papers «ΛΌΓΟΣ»* (pp. 262–265). European Historical Studies. https://doi.org/10.36074/logos-26.04.2024.054

Hasan, S. M. R., Dhakal, A. (2023). Obfuscated malware detection: Investigating real-world scenarios through memory analysis. In *2023, the IEEE International Conference on Telecommunications and Photonics (ICTP)*. IEEE. https://ieeexplore.ieee.org/abstract/document/10490701

Ke, G., Meng, Q., Finley, T., Wang, T., Chen, W., Ma, W., Ye, Q., Liu, T.-Y. (2017). LightGBM: A highly efficient gradient boosting decision tree. In *Advances in Neural Information Processing Systems,* **30**, 3146–3154. Neural Information Processing Systems Foundation.

https://papers.nips.cc/paper_files/paper/2017/hash/6449f44a102fde848669bdd9eb6b76fa-Abstract.html

Khalid, O., Ullah, S., Ahmad, T., Saeed, S., Alabbad, D. A., Aslam, M., Buriro, A., Ahmad, R. (2022). An insight into the machine-learning-based fileless malware detection. *Sensors,* **23**(2), 612. https://www.mdpi.com/1424-8220/23/2/612

Louk, M. H. L., Tama, B. A. (2022). Tree-Based Classifier Ensembles for PE Malware Analysis: A Performance Revisit. *Algorithms,* **15**(9), 332. https://doi.org/10.3390/a15090332

Microsoft (2025, February 15). *LightGBM*. GitHub. Retrieved May 18, 2025, from https://github.com/Microsoft/LightGBM

Microsoft (2025, February 14). *LightGBM parameters* [GitHub documentation]. GitHub. Retrieved June 7, 2025, from https://github.com/microsoft/LightGBM/blob/master/docs/Parameters.rst

Naeem, H., Dong, S., Falana, O. J., Ullah, F. (2023). Development of a deep stacked ensemble with process-based volatile memory forensics for platform-independent malware detection and classification. *Expert Systems with Applications,* **223,** 119952. https://www.sciencedirect.com/science/article/abs/pii/S0957417423004542

Neo, G. K. (2023). *Malware detection in memory images using machine learning* [Final Year Project]. Nanyang Technological University, Singapore. https://dr.ntu.edu.sg/handle/10356/165974

Prashant, B. (2020 July 15). *A guide on XGBoost hyperparameter tuning* [Kaggle Notebook]. Kaggle. Retrieved June 7, 2025, from https://www.kaggle.com/code/prashant111/a-guide-on-xgboost-hyperparameters-tuning

Prashant, B. (2020, June 30). *Bagging vs boosting* [Kaggle Notebook]. Kaggle. Retrieved June 7, 2025, from https://www.kaggle.com/code/prashant111/bagging-vs-boosting

Prashant, B. (2020, July 21). *LightGBM Classifier in Python* [Kaggle Notebook]. Kaggle. Retrieved June 7, 2025, from https://www.kaggle.com/code/prashant111/lightgbm-classifier-in-python

Prashant, B. (2020, December 8). *XGBoost K-Fold CV & feature importance* [Kaggle Notebook]. Kaggle. Retrieved June 7, 2025, from https://www.kaggle.com/code/prashant111/xgboost-k-fold-cv-feature-importance/notebook

Project Jupyter (2014). *Project Jupyter*. Retrieved June 7, 2025, from https://jupyter.org/

Prokhorenkova, L., Gusev, G., Vorobev, A., Dorogush, A. V., Gulin, A. (2017). CatBoost: Unbiased boosting with categorical features. In *Advances in Neural Information Processing Systems,* **30**. Neural Information Processing Systems Foundation. https://papers.nips.cc/paper_files/paper/2017/hash/6449f44a102fde848669bdd9eb6b76fa-Abstract.html

Ramesh, S. P., Anand, S. Raj, Karthikeyan V. G. (2024). Machine Learning Approach for Malware Detection Using Malware Memory Analysis Data. *International Conference on Applications and Techniques in Information Security* (pp 135–145). Springer. https://doi.org/10.1007/978-981-97-9743-1_10

Rani, O., Amit, S., Lena, F., Erin S., Jose, I. (2023). *What are fileless attacks?* Aqua Cloud Native Academy. Retrieved June 7, 2025, from https://www.aquasec.com/cloud-native-academy/application-security/fileless-attacks/

Rathi, R. (2019). *All you need to know about the Gradient Boosting algorithm — Part 2 (Classification)*. Medium. Retrieved June 7, 2025, from https://medium.com/data-science/all-you-need-to-know-about-gradient-boosting-algorithm-part-2-classification-d3ed8f56541e

ReliaQuest (2023). *Fileless malware accounts for 86.2% of all detections*. Global Security Mag. Retrieved June 7, 2025, from https://www.globalsecuritymag.de/fileless-malware-accounts-for-86-2-of-all-detections-reliaquest.html

scikit-learn developers (2012). *sklearn.preprocessing.LabelEncoder*. In *Scikit-learn Documentation*. Retrieved June 7, 2025, from https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.LabelEncoder.html#sklearn.preprocessing.LabelEncoder scikit-learn developers. (2025, January 10). *API Reference.* In *Scikit-learn Documentation.* Retrieved May 18, 2025, from https://scikit-learn.org/stable/api/sklearn.ensemble.html

scikit-learn developers. (2025, June 5). *Glossary of Common Terms and API Elements.* In *Scikit-learn Documentation*. Retrieved June 7, 2025, from https://scikit-learn.org/stable/glossary.html#term-n_jobs

scikit-learn developers. (2025). *Preprocessing — Normalization*. In *Scikit-learn Documentation*. Retrieved June 7, 2025, from https://scikit-learn.org/stable/modules/preprocessing.html#normalization

Trotta, F. (2017). *Understanding L1 and L2 regularization*. Towards Data Science. Retrieved June 7, 2025, from https://towardsdatascience.com/understanding-l1-and-l2-regularization-93918a5ac8d0/

XGBoost Contributors (2021). *XGBoost* [GitHub repository]. GitHub. Retrieved June 7, 2025, from https://github.com/dmlc/xgboost/

Received June 7, 2025, revised November 22, 2025, accepted December 19, 2025