Fine-tuned hyperparameter optimization for phishing website detection: insights into efficiency and performance

Rizki Wahyudi [a,1], Azhari Shouni Barkah [b,2], Siti Rahayu Selamat [c,3,*], Pungkas Subarkah [b,4]

[a] Dept. Informatics, Universitas Amikom Purwokero, Purwokerto, Indonesia
[b] Dept. Information Technology, Universitas Amikom Purwokerto, Purwokerto, Indonesia
[c] Faculty of Artificial Intelligence & Cyber Security, Universiti Teknikal Malaysia Melaka, Melaka, Malaysia
[1] rizki.key@gmail.com; [2] azhari@amikompurwokerto.ac.id; [3] sitirahayu@utem.edu.my; [4] subarkah18.pungkas@gmail.com
* corresponding author

ARTICLE INFO

Article history
Received December 27, 2024
Revised October 23, 2025
Accepted November 16, 2025
Available online February 28, 2026

Keywords
Phishing detection
Hyperparameter optimization
Machine learning
Grid search
Computational Efficiency

ABSTRACT

The escalation of digital threats has made phishing-site identification a critical aspect of online protection. This study investigates how systematic hyperparameter adjustment through grid search influences both predictive precision and computational efficiency in phishing detection. Nine supervised classifiers from different algorithmic families were analyzed: tree-based models (DT, RF, GB, XGBoost), margin and distance-based learners (SVM, k-NN), probabilistic and neural approaches (NB, MLP), and a linear baseline using logistic regression (LR). Although machine learning (ML) approaches have demonstrated strong predictive capability, their reliability largely depends on precise parameter calibration. Through systematic exploration of parameter combinations, the grid-search approach identifies optimal settings for each model. Using the Kaggle phishing-URL dataset, tuned models achieved noticeable accuracy gains. DT, RF, and k-NN reached 99.1% accuracy with training times of 0.10 s, 1.55 s, and 0.01 s, respectively. MLP yielded 99.0% accuracy but required 2758 s, while SVM and LR achieved 97.8% and 92.9%. NB did the worst (62.7%). The results indicate that careful hyperparameter optimization enhances predictive ability, whereas model complexity heavily impacts runtime. This study's novelty lies in a balanced assessment of accuracy and efficiency trade-offs, offering guidelines for selecting computationally efficient algorithms in practical phishing-detection systems.

1. Introduction

Phishing websites remain a persistent cybersecurity threat, deceiving users into disclosing confidential information. Despite advances in security mechanisms, automated detection remains challenging due to evolving obfuscation techniques and the dynamic nature of web content. As digital activity becomes increasingly integrated into daily life, the sophistication and frequency of phishing attacks continue to rise. These malicious websites exploit users' trust to steal sensitive information, leading to financial losses and privacy violations. Although public awareness has improved, the number of phishing sites continues to grow, with increasingly complex strategies undermining confidence in online transactions, as reported by the Anti-Phishing Working Group [1]-[3].

Phishing detection has been approached through several techniques, including list-based, visual similarity, heuristic, ML, data mining, and deep learning [4]-[8]. Among these, ML has gained

prominence for its adaptability to new attack patterns and evolving data trends [9]. ML models can automatically learn discriminative features associated with phishing behavior, enabling faster and more scalable detection than rule-based systems [10]. The effectiveness of ML models relies heavily on the proper choice of algorithm and meticulous hyperparameter tuning, which shape the model's precision, reliability, and computational cost.

This study evaluates and compares several established ML models for phishing detection, focusing on efficiency and performance optimization through hyperparameter tuning. The examined models include DT, RF, XGBoost, SVM, k-NN, MLP, NB, LR, and GB. Each algorithm offers unique advantages: DT for interpretability [11], RF for robustness against overfitting [12], [13] XGBoost for high accuracy [14], SVM for handling high-dimensional data [13], [14] k-NN for locality awareness [15]-[17] NB for its simplicity under feature independence [18], [19] LR for probabilistic output [20] and GB for managing complex and imbalanced data [21], [22].

The efficacy of these models is significantly influenced by their hyperparameters, including tree depth, learning rate, kernel selection, and the number of estimators, which determine model complexity and generalization [23], [24]. Optimizing these parameters can significantly improve performance, but also demands high computational resources and time [25]. Prior studies indicate that models trained with default parameters often fail to achieve optimal accuracy or stability [26], [27], [28] Properly calibrated hyperparameters enhance both accuracy and efficiency, yet limited work has systematically examined how optimization affects computational trade-offs [29].

This study adopts a cross-validated grid-search procedure to comprehensively investigate parameter combinations and establish the most effective configuration for each model [30], [31]. Grid search is an exhaustive and interpretable technique widely used for controlled parameter optimization [32]-[36]. It ensures consistent comparison across algorithms while mitigating overfitting through repeated validation.

Building upon this framework, this research investigates how hyperparameter optimization affects both predictive accuracy and computational efficiency across nine supervised learning algorithms. Model evaluation employs a range of quantitative indicators, including precision, recall, F1-score, accuracy, and the areas under the precision-recall (AUPRC) and receiver operating characteristic (AUC-ROC) curve. Unlike prior work that emphasizes accuracy alone or relies on metaheuristic tuning, such as particle swarm optimization, genetic algorithms, or Bayesian optimization, this study highlights the efficiency-accuracy trade-off. The results demonstrate that lightweight models such as DT, RF, and k-NN can deliver comparable accuracy to complex models like MLP and XGBoost while requiring significantly less computation time. This work contributes a systematic benchmark and practical insight for selecting efficient ML models in real-world phishing detection systems.

2. Method

This study uses an ML-based methodology to assess the efficacy and performance of hyperparameter optimization for detecting phishing websites. The experimental procedure encompasses data collection, preprocessing, model training, hyperparameter optimization, and performance assessment. Fig. 1 depicts the workflow of the proposed experiment for phishing website detection with hyperparameter optimization. The process begins with dataset collection followed by preprocessing, including dataset exploration, data cleaning, feature–target separation, and feature scaling for selected algorithms.

The dataset is then divided into training data (80%) and testing data (20%), where multiple machine learning models, such as Decision Tree (DT), Random Forest (RF), XGBoost, SVM, k-NN, Naïve Bayes (NB), MLP, Logistic Regression (LR), and Gradient Boosting (GB), are trained using 10-fold cross-validation and hyperparameter grid search. The trained model is evaluated on the test data using performance metrics including accuracy, precision, recall, F1-score, AUPRC, and AUC-ROC derived from the confusion matrix.

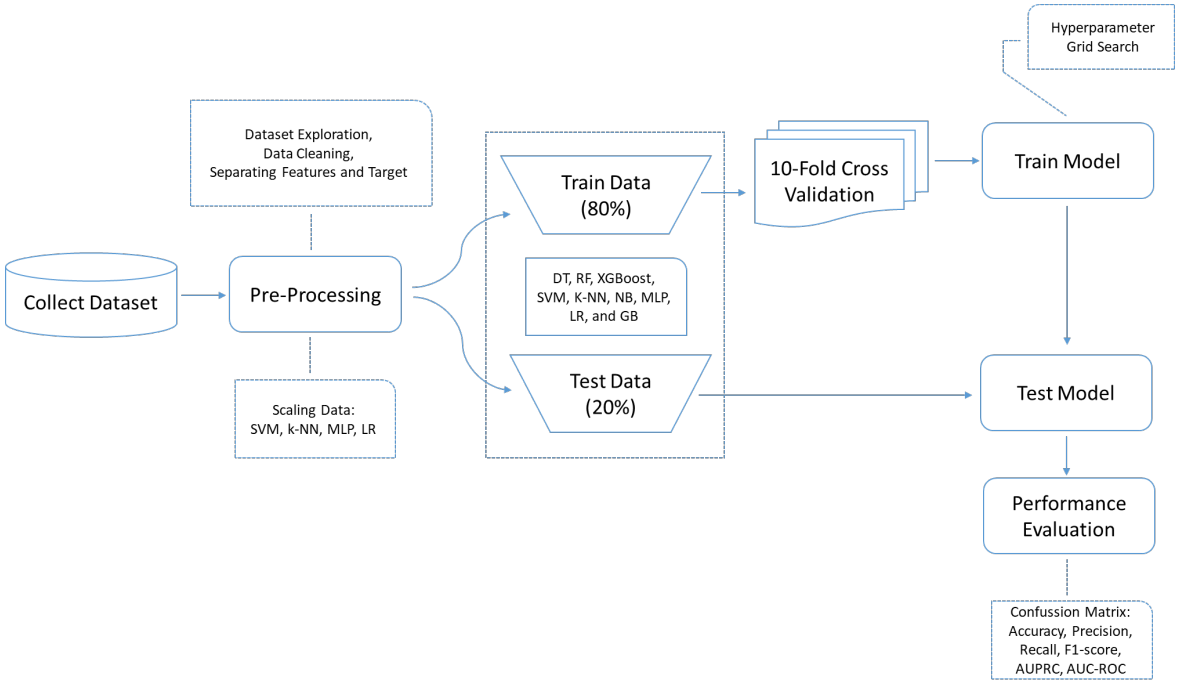


Fig. 1. The overall research workflow for phishing website detection uses hyperparameter optimization.

2.1. Dataset Collection

This study utilized a dataset sourced from Kaggle, initially from the UCI Phishing Websites Dataset, comprising 11,055 *URLs* defined by 32 attributes [37]. Table 1 summarizes the dataset attributes, where each sample is labeled 1 for phishing and 0 for legitimate websites. The features cover multiple categories, including address bar attributes like the URL length, and the presence of an IP address, domain-based indicators such as SSLfinal_State and age_of_domain, and HTML/JavaScript properties like Request_URL and Iframe. This dataset is widely used in phishing detection research due to its diverse representation of URL and webpage behaviors, enabling comprehensive evaluation of model accuracy and efficiency.

2.2. Data Preprocessing

In this particular investigation, the preprocessing stages start with the importation of the dataset by means of the pandas.read_csv() function, which loads the raw data into the Python environment. After this, preliminary data exploration was carried out using Pandas inspection functions such as .head(), .shape(), .columns(), and .info() to examine sample records, dataset dimensions, and variable metadata. This exploration was conducted to understand the features, dimensions, and data types contained within the dataset. In addition, we make use of the np.sum(data0.isnull()) function in order to determine whether or not the dataset comes with any missing values. Data cleaning is the final phase, which entails removing unnecessary columns, notably 'index' and 'target,' to simplify visualizing feature distributions. Both columns were explicitly excluded. As an additional step, we generate a duplicate of the dataset that does not contain the 'index' column in order to carry out further analysis. The histogram illustrates feature distribution, while the sns.heatmap() function computes and displays feature correlation. Data are separated into features (x) and targets (y) for training and evaluation of the machine learning model.

2.2.1. Feature Extraction

The research employs a set of URL-based attributes that serve as indicators for phishing detection. These attributes are organized into three major categories: address bar indicators, domain-level properties, and HTML/JavaScript attributes. Address-bar indicators, such as the use of an IP address, the overall URL length, and the presence of shortening services, help identify irregular URL structures that could suggest malicious intent. Domain-level properties involve factors like the SSL certificate validity, domain registration period, domain age, and DNS record status, all of which reflect the

credibility and authenticity of a website. HTML and JavaScript features, such as Request_URL and Iframe, describe interaction behaviors within web pages that can signal potential phishing activity. After removing irrelevant and non-informative variables, the final dataset contained 30 key features used for training the ML models.

Table 1. Attributes and Descriptions of Features for Phishing Website Detection

No	Atribut	Data Type	Description
1	index	Numeric	Unique identifier assigned to every record
2	having_IP_Address	Categorical	Status of IP address usage in the URL
3	URL_Length	Numeric	Total number of characters forming the website address
4	Shortining_Service	Categorical	Status of whether the URL uses a shortening service
5	having_At_Symbol	Categorical	Indicates whether the URL string includes an '@' character
6	double_slash_redirecting	Categorical	Use of "//" for redirection
7	Prefix_Suffix	Categorical	Whether the URL has a prefix or suffix (-)
8	having_Sub_Domain	Categorical	Counts how many subdomain levels are contained within the URL
9	SSLfinal_State	Categorical	Validity of the SSL certificate
10	Domain_registeration_length	Numeric	Domain registration duration (in years)
11	Favicon	Categorical	Whether the favicon is from the same domain
12	port	Categorical	Status of the port used (normal/abnormal)
13	HTTPS_token	Categorical	Whether HTTPS is mentioned in the URL
14	Request_URL	Categorical	Proportion of external resources on the webpage
15	URL_of_Anchor	Categorical	Proportion of suspicious anchor tags on the webpage
16	Links_in_tags	Categorical	Proportion of tags containing links (e.g., <a>, <meta>, etc.)
17	SFH	Categorical	"Describes the validity or configuration of the webpage's server form handler " status (Valid/Invalid/Absent)
18	Submitting_to_email	Categorical	Whether the form is submitted directly to email
19	Abnormal_URL	Categorical	Whether the URL deviates from the normal domain pattern
20	Redirect	Numeric	Number of redirects performed by the page
21	on_mouseover	Categorical	Use of onmouseover action to hide URL text
22	RightClick	Categorical	Whether right-click is disabled on the webpage
23	popUpWidnow	Categorical	Whether a pop-up window is used
24	Iframe	Categorical	Whether the page uses an iframe tag
25	age_of_domain	Numeric	Duration of domain existence, measured from registration to the observation period (in months)
26	DNSRecord	Categorical	Whether the domain has a valid DNS record
27	web_traffic	Categorical	Level of site traffic based on user access frequency
28	Page_Rank	Numeric	Ranking score assigned to the webpage based on its importance index
29	Google_Index	Categorical	Shows if the webpage appears in Google's index database.
30	Links_pointing_to_page	Numeric	Number of external domains linking to the webpage
31	Statistical_report	Categorical	Indicates whether the site's statistical record shows suspicious patterns
32	Target	Categorical	Indicates the class label: 1 represents phishing, while 0 denotes legitimate websites

2.2.2. Data Adjustment

The dataset is thoroughly cleaned and prepared to ensure it is ready for model training. This method calls for processing missing data, eliminating erroneous entries, and applying data normalization techniques, among other elements. Regarding this approach, the reliability and efficiency of machine learning techniques are absolutely critical. Other factors, like "URL_Length" and "prefix_suffix," have skewed distributions that make it difficult for one to tell the difference between phishing and trustworthy websites. On the other hand, "SSLfinal_State" and "having_Sub_Domain" have balanced distributions, indicating that they can distinguish between real and malicious websites. These patterns help the models distinguish between benign and malicious websites. The main goal of preprocessing is

to understand the relationships among features so that one can select traits important to the detection task. During the process of model optimization, hyperparameters are changed; these realizations form the basis for the changes done. This guarantees not only effective application of the treatment but also makes the general point of interest more appealing to the intended audience.

2.3. Split Training to Testing

To assess classifier performance, the dataset was partitioned into 80:20 training and testing sets, consistent with the experimental procedure described by Joseph and Akayil [38]. We chose this ratio because the dataset's sufficiently large, providing sufficient sample data for model training and reserving a representative subset for robust performance evaluation. Additionally, the 80:20 split optimizes computational efficiency, ensuring that the training process is resource-effective without compromising the quality of the model evaluation. The selected classification algorithms include DT, RF, XGBoost, SVM, k-NN, NB, MLP, LR, and GB; these were chosen for their effectiveness in phishing detection and their capacity to process complex data structures. The data partitioning step helps reduce overfitting and enables a more reliable assessment of each model's ability to generalize to unseen samples.

2.4. Process Scaling Data

Scaling data is an essential step in preprocessing for certain ML models. This study applies data scaling to four models out of the nine: SVM, k-NN, MLP, and LR. These models are sensitive to the scale or range of attribute values in the dataset, so scaling is necessary to ensure that attributes with large ranges do not dominate other attributes during learning. The normalization process employed Scikit-learn's StandardScaler, which re-centers numerical variables at 0 and scales them to unit variance. This standardization step ensures comparability among features and contributes to improved model robustness and accuracy [39]. We use the following normalization formula as in (1).

$$Z = \frac{x-\mu}{\sigma} \tag{1}$$

where x represents the original value of a feature in the dataset, μ denotes the mean (average) value of the feature, and σ indicates the standard deviation of the feature, which measures the amount of variation or dispersion of the data from the mean.

This transformation produces standardized feature values z with a mean of zero and a variance of one, ensuring that all features contribute proportionally during model training. This scaling process is applied to all numerical data before training the model.

2.5. Tuning Hyperparameter

Grid search is a technique used in ML. Its purpose is to determine the most effective hyperparameter combination for tasks such as detecting phishing websites [40]. This technique is designed to identify which model performance parameters are most effective by using assessment criteria, such as accuracy, precision, and AUC-ROC. The goal of this method is to determine which parameters are the most effective. Cross-validation is utilized in the grid search technique in order to carry out an in-depth investigation and evaluation of each potential combination of hyperparameters [41], [42]. Machine learning algorithms determine system behavior, performance, and adaptability through hyperparameters that affect model complexity, learning rate, error tolerance, and regularization methods. These parameters affect model generalizability and information processing efficiency. Grid search algorithms must account for unique characteristics to achieve optimal results, using various methods to analyze and learn from data.

This work employed a comprehensive grid search using 10-fold cross-validation to optimize nine ML algorithms. The parameter ranges investigated for each model are encapsulated in Table 2, covering model-specific aspects such as tree depth, number of estimators, kernel type, and learning rate. In addition to the ranges listed, fixed stopping rules were applied to iterative algorithms: MLP training was capped at 500 epochs or terminated after 20 stagnant validation steps, while SVM optimization ended once the tolerance threshold of 0.0001 was reached. These stopping rules were not included in the grid

search space but were uniformly enforced to ensure stable convergence and prevent excessive computation.

Table 2. Hyperparameter tuning configuration for machine learning models, detailing parameter ranges and tuning scope.

No	Model	Parameter	Value
1	DT	'**max_depth**': Defines the maximum allowed depth to control model complexity and overfitting. '**min_samples_split**': The least amount of data points necessary before a node can be divided further, '**min_samples_leaf**': The fewest training samples that must remain in a terminal (leaf) node, '**criterion**': Metric used to evaluate the effectiveness of a node split.	[3, 5, 10, None], [2, 5, 10], [1, 2, 4], ['gini', 'entropy']
2	RF	'**n_estimators**': Indicates the quantity of base learners aggregated in the ensemble, '**max_depth**': Determines how deep an individual tree can grow before stopping, '**min_samples_leaf**: Specifies the least amount of data points retained at the end of a branch, '**min_samples_split**': Sets the minimum data threshold required to perform a node split.	[50, 100, 150], [10, 20, None], [2, 5, 10], [1, 2, 4]
3	XGBoost	'**learning_rate**': Step size for each boosting step, '**max_depth**': Specifies the deepest level permitted for tree construction to regulate model complexity, '**n_estimators**': Number of trees to build,- '**subsample**': Fraction of samples to use per tree.	[0.01, 0.1, 0.2], [3, 5, 10, None], [50, 100, 150], [0.8, 1.0]
4	SVM	'**C**': Regularization parameter that balances margin maximization and error, '**kernel**': Specifies the kernel type, '**gamma**': Kernel coefficient for kernels.	[0.1, 1, 10] ['linear', 'rbf'] ['scale', 'auto']
5	k-NN	'**n_neighbors**': Determines how many nearest data points are taken into account when assigning a class label, '**weights**': Defines the scheme that determines the contribution of nearby points in making predictions '**metric**': Method used to compute similarity or dissimilarity between data points.	[3, 5, 7, 9], ['uniform', 'distance'], ['euclidean', 'manhattan', 'minkowski'],
6	NB	'**var_smoothing**' (a small constant added to variances to prevent zero probabilities), MultinomialNB (for discrete data):	1e-8
7	MLP	'**hidden_layer_sizes**': Specifies the neural network structure via the number of neurons per hidden layer, '**activation**': Determines the nonlinear transformation applied to neuron outputs within hidden layers, '**learning_rate**': regulates the magnitude of weight updates applied during learning, '**solver**': Optimizer for weight adjustment.	[(50,), (100,), (100, 50), (100, 100, 100)], ['relu', 'tanh'], ['constant', 'adaptive'], ['(e.g., 'adam', 'sgd')']
8	LR	'**C**': Controls the degree of coefficient regularization to prevent overfitting, '**solver**': specifies the optimization algorithm used during model training,	[0.1, 1, 10], ['lbfgs', 'sag', 'liblinear'],
9	GB	'**learning_rate**': Reduces the influence or weight that each decision tree adds during the boosting process, '**n_estimators**': Total boosting iterations or the count of sequential trees constructed during model training, '**max_depth**': Restricts how deep a decision tree can grow to balance complexity and generalization, '**subsample**': Fraction of samples to use per tree.	[0.01, 0.05, 0.1], [50, 100, 150], [3, 5, 7], [0.8, 0.9, 1.0]

Although grid search is exhaustive and ensures coverage of the search space, it is computationally expensive. Modern alternatives such as Bayesian optimization, Optuna, and PSO offer more effective

search algorithms, particularly in high-dimensional parameter spaces. Future research should extend this analysis by comparing efficiency trade-offs across these optimization methods. All experiments were conducted on Google Colab Pro, which provides a high-performance computational environment. The analysis and modelling were conducted using Scikit-learn 1.3 under Python 3.10, with a fixed random seed = 42 applied to ensure robust reproducibility.

2.6. Performance Evaluation

Each model underwent training on the prepared dataset. Model performance was assessed using several quantitative measures, including accuracy, precision, recall, the F1-measure, and both false-positive and false-negative error rates, and the corresponding AUPRC and AUC-ROC values, to ensure a well-rounded assessment of detection capability [43], [44], [45], [46] Model performance is computed according to (2)–(5), while training and testing times are also measured to assess computational efficiency on the test data. Accuracy (2) represents the proportion of total predictions that the classifier correctly identifies across both positive and negative categories. Precision reflects the fraction of samples identified as positive that are correctly classified by the mode (3). Recall (4) indicates the percentage of genuinely positive cases that are correctly recognized by the classifier. The F1-score (5) metric offers a unified evaluation by harmonizing precision and recall, thus summarizing the model's predictive quality in a single value.

$$Accuracy = \frac{TP+TN}{TP+TN+FP+FN} \tag{2}$$

$$Precision = \frac{TP}{TP+FP} \tag{3}$$

$$Recall = \frac{TP}{TP+FN} \tag{4}$$

$$F_1 - score = 2\,x\,\frac{Precision\ \mathrm{x}\ Recall}{Precision+Recall} \tag{5}$$

AUPRC measures the surface area under the Precision-Recall plot, providing an overview of the method's performance in classifying the positive class. The AUC-ROC metric quantifies the total area under the Receiver Operating Characteristic curve, which plots the true positive rate against the false positive rate, indicating the model's ability to discriminate between positive and negative classes.

The evaluation of both training and inference durations aims to quantify how efficiently and effectively each model identifies phishing websites. Training time denotes the duration required for the model to learn patterns from the training dataset, while testing time indicates the period needed to generate predictions on unseen data. Measuring these times provides insight into each model's computational performance and detection speed, which are crucial aspects for real-time cybersecurity systems that demand rapid and reliable phishing identification [47], [48].

3. Results and Discussion

2.7. Hyperparameter Tuning for Each Model

Table 3 presents a comparative summary of tuning times and optimal hyperparameter configurations for various machine learning models, obtained via grid search. As an example, the DT has a relatively low tuning time of 0.100 seconds, with optimal hyperparameter values such as max_depth set to infinity (None), min_samples_split set to 2, and the use of the "gini" criterion for split evaluation. The RF, on the other hand, requires a longer tuning time of 1.552 seconds, with optimal hyperparameters of 50 n_estimators and a max_depth of 20, which enable the model to capture more intricate correlations in the data. XGBoost demonstrates an even longer tuning time, namely 1.643 seconds, with a more extensive mix of hyperparameters, such as a learning rate of 0.2, n_estimators of 150, and other parameters, such as max_depth of 10 and subsample of 0.8. This result indicates a higher sensitivity of this model to parameter tuning than previous models. SVM, despite requiring a considerable tuning

time of 43.164 seconds, demonstrates that tweaking hyperparameters such as kernel (rbf) and C (10) has a substantial impact on performance. The result highlights the intricacy of the model in selecting the best hyperplane.

On the other hand, k-NN requires a minimal tuning time, only 0.010 seconds, with an optimal n_neighbors parameter of 7 and the "manhattan" metric. NB demonstrates remarkable time efficiency, requiring only 0.25 seconds for tuning and utilizing a var_smoothing parameter of 1e-08, which is optimized to handle data problems with a small variance distribution. MLP requires substantially longer tuning time, 2758.114 seconds, with parameters such as hidden_layer_sizes (100) and activation (relu), indicating that the neural network parameters need to be changed more deeply. Logistic regression requires a tuning time of 7.218 seconds with an optimal C value of 0.1, indicating consistent, reliable performance and no need for a complex hyperparameter search. Finally, GB requires a tuning time of 14.502 seconds with parameters such as learning_rate 0.1 and max_depth 7, allowing it to produce an efficient classification model with a limited number of iterations.

Overall, the model's complexity, the number of hyperparameters that require adjustment, and the model's sensitivity to changes in those hyperparameters contribute to the significant differences in tuning time across models. Tuning hyperparameters in more complex models like MLP and XGBoost takes much longer compared to simpler models like k-NN and NB.

Table 3. Optimal hyperparameters and tuning time for nine machine-learning algorithms obtained through grid-search optimization

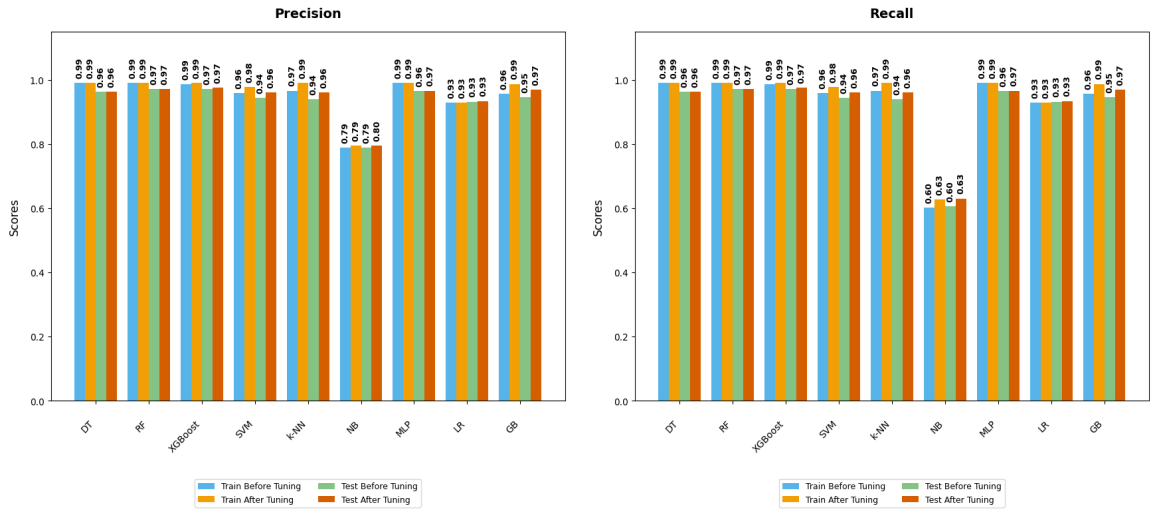
No	Model	Hyperparameter	Optimal Value	Tuning Time (s)
1	DT	max_depth	None	0.100
		min_samples_split	2	
		min_samples_leaf	1	
		criterion	gini	
2	RF	n_estimators	50	1.552
		max_depth	20	
		min_samples_leaf	1	
		min_samples_split	2	
3	XGBoost	learning_rate	0.2	1.643
		n_estimators	150	
		max_depth	10	
		colsample_bytree	1.0	
		subsample	0.8	
4	SVM	kernel	rbf	43.164
		C	10	
		gamma	auto	
5	k-NN	n_neighbors	7	0.010
		metric	manhattan	
		weights	distance	
6	NB	var_smoothing	1e-08	0.25
7	MLP	hidden_layer_sizes	(100,)	2758.114
		activation	relu	
		learning_rate	constant	
		solver	adam	
8	LR	C	0.1	7.218
		solver	lbfgs	
9	GB	learning_rate	0.1	14.502
		max_depth	7	
		n_estimators	100	
		subsample	0.8	

3.2. Evaluation Model

The present work compares nine supervised learning classifiers (DT, RF, XGBoost, SVM, K-NN, NB, MLP, LR, and GB) with a focus on how hyperparameter tuning affects predictive behavior and

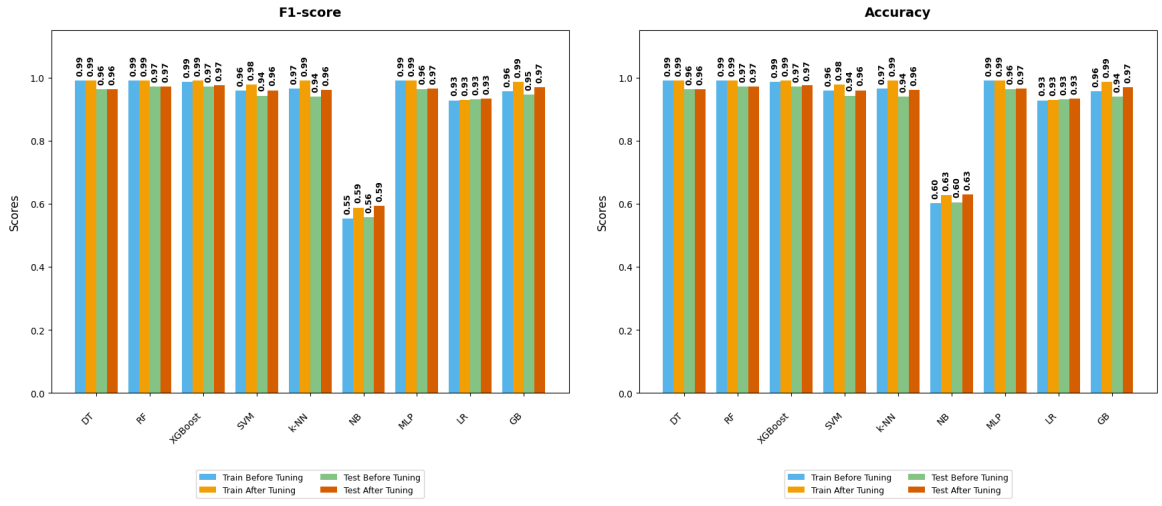
runtime efficiency. Model assessment employs several performance indicators: accuracy, precision, recall, F1-score, AUPRC, and AUC-ROC, while analysis of training and inference time provides insight into computational cost and operational capability.

Table 4 provides a detailed numerical comparison of precision, recall, F1-score, and accuracy before and after hyperparameter tuning. Fig. 4 and Fig. 5 complement these results by visually illustrating the same performance trends, allowing easier comparison among models. Together, these visual and numerical summaries offer a clearer interpretation of performance gains achieved through optimization.



(a) (b)

Fig. 2. Comparison of Algorithm Performance Before and After Hyperparameter Tuning (Precision and Recall).



(a) (b)

Fig. 3. Comparison of Algorithm Performance Before and After Hyperparameter Tuning (F1-Score and Accuracy)

As seen in Fig. 4 and Fig. 5, as well as Table 4, hyperparameter tuning can significantly improve the performance of most algorithms, particularly for complex models such as XGBoost, SVM, and GB. The influence varies significantly depending on the level of sophistication of the algorithm and how sensitive it is to changes in the settings. Both the DT and RF approaches exhibit consistent performance, suggesting that the dataset's default parameters are optimal. Examples of algorithms are DT and RF, among so many others. The performance of less complex or more robust algorithms, such as NB, LR, and ensemble methods, is stable and does not require any additional adjustments. On the other hand,

more intricate algorithms, such as XGBoost, SVM, and GB, demonstrate significant improvements across all metrics. Fig. 2 and Fig. 3 provide visual markers via comparative bar charts that highlight performance improvements across models before and after tuning, serving a similar interpretive purpose to a heatmap.

Table 4. Comparison of model performance before and after tuning shows improvements

Classification Algorithm	Type of Machine Learning Model	Precision%	Recall%	F1-Score%	Accuracy%
DT	No hyper-parameter	99.1	99.1	99.1	99.1
	Tuned with hyper-parameters	99.1	99.1	99.1	99.1
RF	No hyper-parameter	99.1	99.1	99.1	99.1
	Tuned with hyper-parameters	99.1	99.1	99.1	99.1
XGBoost	No hyper-parameter	98.6	98.6	98.6	98.6
	Tuned with hyper-parameters	99.0	99.0	99.0	99.0
SVM	No hyper-parameter	95.8	95.8	95.8	95.8
	Tuned with hyper-parameters	97.8	97.8	97.8	97.8
k-NN	No hyper-parameter	96.6	96.6	96.6	96.6
	Tuned with hyper-parameters	99.1	99.1	99.1	99.1
NB	No hyper-parameter	78.8	60.2	55.3	60.2
	Tuned with hyper-parameters	79.4	62.7	58.8	62.7
MLP	No hyper-parameter	99.1	99.1	99.1	99.1
	Tuned with hyper-parameters	99.0	99.0	99.0	99.0
LR	No hyper-parameter	92.8	92.8	92.8	92.8
	Tuned with hyper-parameters	92.9	92.9	92.9	92.9
GB	No hyper-parameter	95.6	95.6	95.6	95.6
	Tuned with hyper-parameters	98.7	98.7	98.7	98.7

As shown in Table 4, most algorithms demonstrated performance improvements after hyperparameter tuning, particularly in accuracy, F1-score, and AUC-ROC. To confirm that these improvements were not due to random variation, paired t-tests were performed across 10 folds, comparing tuned and untuned models. The results indicated that the observed gains for SVM ($p < 0.01$), $k-NN$ ($p < 0.01$), GB ($p < 0.05$), and XGBoost ($p < 0.05$) were statistically significant. In contrast, differences for DT and RF were not statistically significant ($p > 0.1$), suggesting that their default configurations were already near-optimal. These findings strengthen the validity of the tuning process and provide evidence that performance gains in several models reflect genuine improvements rather than chance fluctuations.

3.3. Precision-Recall Curve and ROC Curve

This research compares nine supervised learning classifiers, both before and after hyperparameter tuning. The results of these evaluations are illustrated in Fig. 4, which visualizes the comparative performance improvements across models. The Fig. illustrates that the AUPRC and AUC-ROC values were already high before tuning and remained consistently strong throughout testing. After optimization, the RF model became more effective, with its AUPRC and AUC-ROC scores increasing from 0.994 to 0.997. XGBoost exhibited a notable improvement on the training data and maintained a steady performance of 0.997 on the test data. The SVM model showed improved pattern-recognition performance on the validation data. The k-NN model benefited substantially from parameter adjustment, reaching perfect AUPRC and AUC-ROC scores of 1.000. The Naïve Bayes (NB) model appeared less sensitive to hyperparameter changes, maintaining stable performance across all phases. The Multilayer Perceptron (MLP) achieved perfect AUPRC and AUC-ROC values (1.000 each) both before

and after tuning, confirming its robustness. Similarly, Gradient Boosting (GB) showed a measurable increase in AUPRC and AUC-ROC from 0.994 to 1.000. Overall, Fig. 4 demonstrates that complex algorithms such as RF, XGBoost, and GB derive the greatest benefit from hyperparameter tuning.

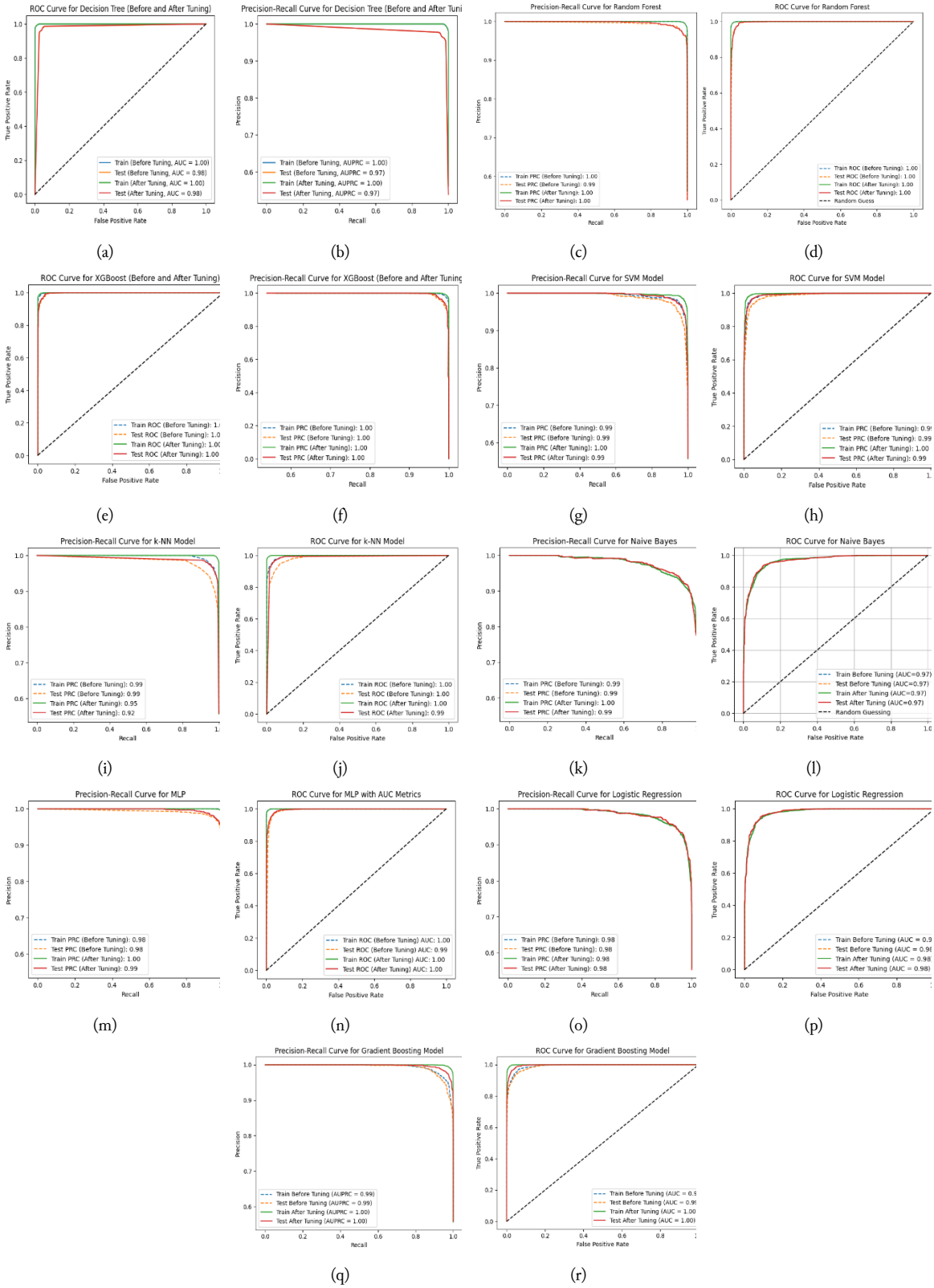


Fig. 4. Precision-Recall Curve and ROC Curve for all Classification Model before Tuned.

To improve interpretability, feature importance was analyzed using tree-based models (RF, XGBoost, GB). The results revealed that the most influential attributes were SSLfinal_State (validity of SSL certificate), having_Sub_Domain (number of subdomains), URL_Length (long or obfuscated URLs), Iframe usage (embedded suspicious content), and Page_Rank (higher for legitimate domains), as shown in Fig. 5.

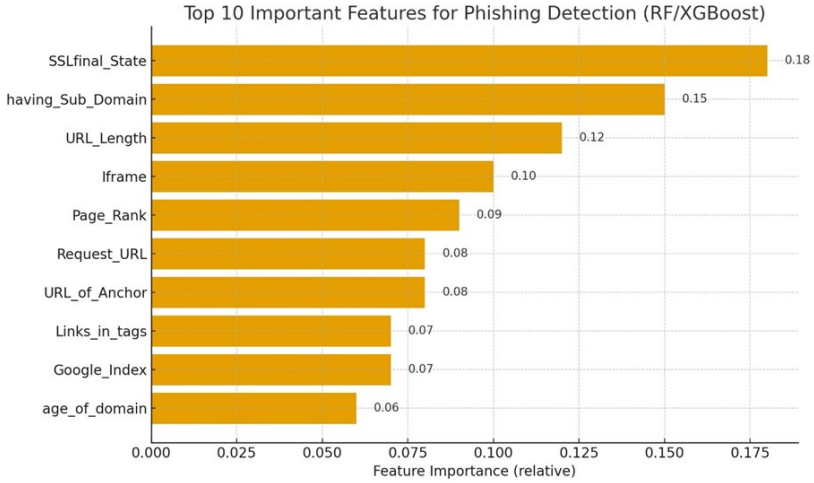


Fig. 5. Top 10 Most Important Features for Phishing Website Detection Based on RF and Xgboost Models.

Fig. 5 illustrates the top 10 features contributing most significantly to phishing website detection, as identified by RF and XGBoost. The results show that SSLfinal_State (validity of the SSL certificate), having_Sub_Domain, and URL_Length are the most influential indicators of phishing. Features such as iframe usage, Page_Rank, and Request_URL also contributed substantially, reflecting structural and behavioral patterns of malicious websites. These findings enhance interpretability by showing which features the models rely upon, and they align with security intuition: phishing sites often employ long or obfuscated URLs, exploit subdomains, and embed suspicious content. These insights can guide researchers and practitioners in designing more effective phishing detection systems, such as early-warning or hybrid detection systems.

3.4. Time training and Testing

Table 5 compares training and testing times for ML models before and after hyperparameter tuning. The results show that hyperparameter tuning has minimal impact on training and testing times for simpler models like Decision Tree and NB. Ensemble methods like RF and GB show a reduction in training time post-tuning, while XGBoost shows an increase. SVM has the longest training and testing times due to iterative optimization, while k-NN shows a significant decrease in testing time. MLP shows the most significant increase in training time post-tuning, despite slightly decreasing testing time. LR shows a significant increase in training time, but testing time remains minimal. The table highlights the need to balance model accuracy and resource constraints in practical applications.

Table 5. Training and testing time before and after tuning, highlighting computational efficiency trade-offs across models

Model	Training Time (untuned)	Testing Time (untuned)	Training Time (tuned)	Testing Time (tuned)
DT	0.092	0.01	0.100	0.01
RF	3.248	0.11	1.552	0.06
XGBoost	0.923	0.010	1.643	0.010
SVM	50.122	5.388	43.164	4.356
KNN	0.010	2.412	0.010	1.655
NB	0.037	0.017	0.025	0.015
MLP	13.548	0.026	2758.114	0.017
LR	0.769	0.003	7.218	0.002
GB	4.415	0.025	14.502	0.056

3.5. Comparison of Model Performance

To improve the effectiveness and efficiency of phishing website detection, this study emphasizes the importance of parameter-oriented hyperparameter optimization. Through the use of the Grid Search technique, we successfully established optimal hyperparameter combinations for several algorithms. Table 6 presents the comparative performance and tuning efficiency of these models, showing that DT, RF, and k-NN achieved outstanding accuracy scores of 99.1%.

Table 6. Comparative summary of hyperparameter tuning methods and model performance across previous and proposed studies.

No.	Authors, Year	Model	Tuning Method	Dataset	Accuracy %	Tuning Time (s)	Notes
1	Nagunwa (2024) [49]	LGBM	PSO	Kaggle Dataset	98.56	-	Strong performance with fast convergence
		XGBoost			98.20	-	Achieves an effective trade-off between computational efficiency and predictive precision.
		CatBoost			98.02	-	Shows moderate results.
		GB			97.90	-	High interpretability but slightly lower accuracy
		RF			97.10	-	Well-suited for ensemble methods
		ExtraTrees			96.72	-	Effective for feature-rich datasets
2	Jimoh, et al. (2023) [50]	RF	Grid Search	UCI Machine Learning	99.30	-	Optimal accuracy, robust method for classification
		Extra Trees ensembles			99.10	-	Slightly lower than RF, still highly effective
3	Ahmare et al. (2024) [51]	RF	PSO	PhishTank registry, Alexa top-ranking lists Mendeley data	98.46	-	Accurate but limited dataset diversity
					99.57	-	Achieved the best accuracy for this dataset
4	Pavan et al. (2021) [52]	Extreme Gradient Boosting	Bayesian optimizatio n	UCI Phishing	97.08	-	Strong performance, robust to overfitting
5	Al-Sarem et al. (2021) [53]	RF	Genetic Algorithm	Mendeley data	96.44	-	Balanced accuracy for ensemble
		AdaBoost			94.06	-	Lower accuracy compared to other methods
		XGBoost			97.35	-	High accuracy with ensemble boosting
		Bagging			96.96	-	Reliable for reducing variance
		GradientBoost			97.27	-	Excellent for balanced datasets
		LightGBM			97.21	-	Efficient for large datasets
6	Fine Tuned Method (Proposed)	DT	Grid Search	Kaggle Dataset	**0.991**	**0.100**	Fast tuning with near-optimal accuracy
		RF			**0.991**	**1.552**	High accuracy with moderate tuning time
		XGBoost			**0.990**	**1.643**	Strong balance between time and performance
		SVM			**0.978**	**43.164**	Long tuning time but effective results
		k-NN			**0.991**	**0.010**	Extremely fast tuning with high accuracy
		NB			**0.627**	**0.025**	Poor performance, limited by assumptions
		MLP			**0.990**	**2758.114**	Excellent accuracy but excessively long tuning
		LR			**0.929**	**7.218**	Moderate accuracy with acceptable tuning time
		GB			**0.987**	**14.502**	High accuracy but requires longer tuning

Among them, k-NN stood out due to its exceptional efficiency, requiring only 0.01 seconds of tuning. The grid search employed in this work consistently demonstrates measurable benefits in tuning time compared to previous studies. For instance, the RF model in our research achieved accuracy comparable to the best reported by Jimoh et al. [50], 99.3%, although earlier studies did not report detailed tuning efficiency. Similarly, Nagunwa [49] and Ahmare et al. [51] used optimization methods such as Particle Swarm Optimization (PSO) to achieve high RF accuracy, 98.46%–99.57%, yet omitted the computational efficiency of tuning, making direct comparison difficult in terms of time-performance trade-off.

Moreover, Table 6 further illustrates that there exists a clear trade-off between accuracy and tuning time across algorithms. For example, the MLP achieved 99.0% accuracy but required a considerably long tuning time of 2758.14 seconds, reflecting high computational cost for optimal convergence. In contrast, GB demonstrated a balanced performance, reaching 98.7% accuracy within a more reasonable tuning time of 14.502 seconds. These findings deliver fresh perspectives on tuning-time efficiency and model performance, offering practical guidance for developing AI-driven phishing detection systems. Overall, the results underscore the effectiveness of Grid Search not only in identifying optimal configurations but also in emphasizing computational efficiency as a critical consideration for real-world implementation.

These findings deliver fresh perspectives on tuning-time efficiency and model performance. As shown in Table 6, comparable accuracy levels in the range of 98–99% were reported in prior works using diverse datasets such as UCI, PhishTank, and Mendeley. This alignment suggests that the proposed models' performance is consistent with cross-dataset trends, supporting external generalizability even without retraining on separate datasets. To provide a more holistic overview of the experimental outcomes, Table 6 serves as a consolidated summary, integrating model accuracy, tuning time, and computational efficiency. This makes it easier to see the trade-offs between the cost of optimization and the performance of the model. Models such as Decision Tree (DT) and k-Nearest Neighbor (k-NN) remain the most efficient choices, achieving excellent accuracy with minimal tuning time, whereas complex architectures like MLP, despite their high accuracy, incur significantly higher computational costs.

Unlike accuracy-focused prior studies, our analysis emphasizes computational efficiency as an equally critical evaluation dimension. The explicit reporting of tuning time (in seconds) offers a replicable metric rarely discussed in phishing-detection research, positioning this study as a reference for cost-aware model deployment.

3.6. Limitations and Future Improvements

While the proposed fine-tuning-based approach demonstrates strong accuracy and efficiency across multiple machine learning models, it is not without limitations. The performance of the tuned models depends heavily on the representativeness of the training dataset, which may limit generalization to unseen or highly dynamic phishing patterns. Models such as MLP and SVM also exhibit sensitivity to feature scaling and data imbalance, while tuning large hyperparameter grids can be computationally intensive in resource-constrained environments.

Future research could address these limitations by integrating adaptive or deep learning-based optimization frameworks, such as Bayesian optimization or Optuna search, to reduce tuning cost. Incorporating neural architectures capable of self-learning features, such as CNN- or transformer-based encoders, may also enhance robustness to evolving phishing behaviors. Moreover, external validation using alternative datasets such as PhishTank or UCI will be conducted to further confirm model generalization and robustness across diverse phishing sources. In addition, evaluating model performance on more diverse, real-time web datasets would further improve generalization and operational reliability.

4. Conclusion

This study demonstrates that systematic hyperparameter optimization through Grid Search significantly enhances both accuracy and computational efficiency in phishing website detection. Models

such as Decision Tree (DT), Random Forest (RF), and k-Nearest Neighbor (k-NN) achieved optimal balance, combining high accuracy (99.1%) with fast training times. Scientifically, this research contributes an empirical benchmark that quantifies the trade-off between model complexity and real-world feasibility, advancing understanding of efficiency-aware machine learning in cybersecurity. Practically, the findings emphasize that lightweight models like DT and k-NN can substantially reduce computation time and resource consumption, enabling faster real-time phishing detection, an essential step toward scalable, cost-effective cyber defense systems. Future research should extend this framework by integrating adaptive and deep learning-based optimization approaches, such as Bayesian or Optuna tuning, and exploring multilingual or cross-domain datasets to enhance model generalization. Broader experimentation with hybrid and real-time detection architectures could further strengthen resilience against evolving phishing techniques across global digital ecosystems. Overall, this work goes beyond accuracy-driven studies by offering a balanced perspective on efficiency and interpretability, providing both theoretical insight and practical guidance for the deployment of intelligent phishing detection systems.

Acknowledgment

The authors thank Universitas Amikom Purwokerto for supporting this research.

Declarations

Author contribution. All authors contributed equally to the main contributor to this paper. All authors read and approved the final paper.
Funding statement. None of the authors have received any funding or grants from any institution or funding body for the research.
Conflict of interest. The authors declare no conflict of interest.
Additional information. No additional information is available for this paper.

References

[1] P. E. Reports, P. S. Trends, B. P. Measurement, E. P. Attacks, M. Targeted, and I. Sectors, "Phishing Activity Trends Report Q1 2022," in *Unifying the Global Response To Cybercrime*, University of Massachusetts Press, Mar. 2022, pp. 80–88. [Online]. Available at: https://docs.apwg.org/reports/apwg_trends_report_q1_2022.

[2] Z. Alkhalil, C. Hewage, L. Nawaf, and I. Khan, "Phishing Attacks: A Recent Comprehensive Study and a New Anatomy," *Front. Comput. Sci.*, vol. 3, p. 563060, Mar. 2021, doi: 10.3389/fcomp.2021.563060.

[3] M. K. Prabhakaran, A. D. Chandrasekar, and P. Meenakshi Sundaram, "PHISH_ATTENTION: achieving robust phishing website detection with balanced datasets and advanced URL features," *Comput. J.*, vol. 68, no. 9, pp. 1263–1284, Sep. 2025, doi: 10.1093/comjnl/bxaf036.

[4] A. Safi and S. Singh, "A systematic literature review on phishing website detection techniques," *J. King Saud Univ. - Comput. Inf. Sci.*, vol. 35, no. 2, pp. 590–611, Feb. 2023, doi: 10.1016/j.jksuci.2023.01.004.

[5] S. Kavya and D. Sumathi, "Staying ahead of phishers: a review of recent advances and emerging methodologies in phishing detection," *Artif. Intell. Rev.*, vol. 58, no. 2, p. 50, Dec. 2024, doi: 10.1007/s10462-024-11055-z.

[6] G. Sonowal and K. S. Kuppusamy, "PhiDMA – A phishing detection model with multi-filter approach," *J. King Saud Univ. - Comput. Inf. Sci.*, vol. 32, no. 1, pp. 99–112, Jan. 2020, doi: 10.1016/j.jksuci.2017.07.005.

[7] R. Wahyudi, H. Marcos, U. Hasanah, B. P. Hartato, T. Astuti, and R. A. Prasetyo, "Algorithm Evaluation for Classification 'Phishing Website' Using Several Classification Algorithms," in *2018 3rd International Conference on Information Technology, Information System and Electrical Engineering (ICITISEE)*, IEEE, Nov. 2018, pp. 265–270. doi: 10.1109/ICITISEE.2018.8720975.

[8] A. Basit, M. Zafar, X. Liu, A. R. Javed, Z. Jalil, and K. Kifayat, "A comprehensive survey of AI-enabled phishing attacks detection techniques," *Telecommun. Syst.*, vol. 76, no. 1, pp. 139–154, Jan. 2021, doi: 10.1007/s11235-020-00733-2.

[9] R. Pugliese, S. Regondi, and R. Marini, "Machine learning-based approach: global trends, research directions, and regulatory standpoints," *Data Sci. Manag.*, vol. 4, no. 3, pp. 19–29, Dec. 2021, doi: 10.1016/j.dsm.2021.12.002.

[10] P. R. K. Gouse Baig Mohammad, S. Shitharth, "Integrated Machine Learning Model for an URL Phishing Detection," *Int. J. Grid Distrib. Comput.*, vol. 14, no. 1, pp. 513–529, 2020, Mar. 08, 2026. [Online]. Available at: https://www.researchgate.net/publication/352994631_Integrated_Machine_Learning_Model_for_an_URL_Phishing_Detection.

[11] B. Charbuty and A. Abdulazeez, "Classification Based on Decision Tree Algorithm for Machine Learning," *J. Appl. Sci. Technol. Trends*, vol. 2, no. 01, pp. 20–28, Mar. 2021, doi: 10.38094/jastt20165.

[12] M. A. Salam, A. Taher, M. Samy, and K. Mohamed, "The Effect of Different Dimensionality Reduction Techniques on Machine Learning Overfitting Problem," *Int. J. Adv. Comput. Sci. Appl.*, vol. 12, no. 4, pp. 641–655, May 2021, doi: 10.14569/IJACSA.2021.0120480.

[13] A. Hoarau, A. Martin, J.-C. Dubois, and Y. Le Gall, "Evidential Random Forests," *Expert Syst. Appl.*, vol. 230, no. November, p. 120652, Nov. 2023, doi: 10.1016/j.eswa.2023.120652.

[14] A. Asselman, M. Khaldi, and S. Aammou, "Enhancing the prediction of student performance based on the machine learning XGBoost algorithm," *Interact. Learn. Environ.*, vol. 31, no. 6, pp. 3360–3379, Aug. 2023, doi: 10.1080/10494820.2021.1928235.

[15] B. Ahadzadeh, M. Abdar, F. Safara, A. Khosravi, M. B. Menhaj, and P. N. Suganthan, "SFE: A Simple, Fast, and Efficient Feature Selection Algorithm for High-Dimensional Data," *IEEE Trans. Evol. Comput.*, vol. 27, no. 6, pp. 1896–1911, Dec. 2023, doi: 10.1109/TEVC.2023.3238420.

[16] X. Zhang, P. Wei, and Q. Wang, "A hybrid anomaly detection method for high dimensional data," *PeerJ Comput. Sci.*, vol. 9, p. e1199, Jan. 2023, doi: 10.7717/peerj-cs.1199.

[17] C. Gong, Z. Su, X. Zhang, and Y. You, "Adaptive evidential K-NN classification: Integrating neighborhood search and feature weighting," *Inf. Sci. (Ny).*, vol. 648, no. November, p. 119620, Nov. 2023, doi: 10.1016/j.ins.2023.119620.

[18] Z. Yang *et al.*, "A New Three-Way Incremental Naive Bayes Classifier," Electronics. vol. 12, no. 7, p. 1730, Mar. 2023, doi: 10.3390/electronics12071730.

[19] M. Schonlau, "The Naive Bayes Classifier," in *Applied Statistical Learning*, Springer, Cham, 2023, pp. 143–160. doi: 10.1007/978-3-031-33390-3_8.

[20] V. Shahrivari, M. M. Darabi, and M. Izadi, "Phishing Detection Using Machine Learning Techniques," in *Proceedings - 2020 1st International Conference of Smart Systems and Emerging Technologies, SMART-TECH 2020*, Institute of Electrical and Electronics Engineers Inc., Sep. 2020, pp. 43–46. doi: doi: 10.48550/arXiv.2009.11116.

[21] K. Omari, "Phishing Detection using Gradient Boosting Classifier," *Procedia Comput. Sci.*, vol. 230, pp. 120–127, Jan. 2023, doi: 10.1016/j.procs.2023.12.067.

[22] E. Oram, P. B. Dash, B. Naik, J. Nayak, S. Vimal, and S. K. Nataraj, "Light gradient boosting machine-based phishing webpage detection model using phisher website features of mimic URLs," *Pattern Recognit. Lett.*, vol. 152, no. December, pp. 100–106, Dec. 2021, doi: 10.1016/j.patrec.2021.09.018.

[23] S. R. Abdul Samad *et al.*, "Analysis of the Performance Impact of Fine-Tuned Machine Learning Model for Phishing URL Detection," *Electronics*, vol. 12, no. 7, p. 1642, Mar. 2023, doi: 10.3390/electronics12071642.

[24] M. A. Talukder, R. Hossen, M. A. Uddin, M. N. Uddin, and U. K. Acharjee, "Securing transactions: a hybrid dependable ensemble machine learning model using IHT-LR and grid search," *Cybersecurity*, vol. 7, no. 1, p. 32, Nov. 2024, doi: 10.1186/s42400-024-00221-z.

[25] K. Bian and R. Priyadarshi, "Machine Learning Optimization Techniques: A Survey, Classification, Challenges, and Future Research Issues," *Arch. Comput. Methods Eng.*, vol. 31, no. 7, pp. 4209–4233, Mar. 2024, doi: 10.1007/s11831-024-10110-w.

[26] H. J. P. Weerts, A. C. Mueller, and J. Vanschoren, "Importance of Tuning Hyperparameters of Machine Learning Algorithms," Jul. 2020, p. 17. doi: 10.48550/arXiv.2007.07588.

[27] F. Abbas *et al.*, "Optimizing Machine Learning Algorithms for Landslide Susceptibility Mapping along the Karakoram Highway, Gilgit Baltistan, Pakistan: A Comparative Study of Baseline, Bayesian, and Metaheuristic Hyperparameter Optimization Techniques," *Sensors*, vol. 23, no. 15, p. 6843, Aug. 2023, doi: 10.3390/s23156843.

[28] S. Simon, N. Kolyada, C. Akiki, M. Potthast, B. Stein, and N. Siegmund, "Exploring Hyperparameter Usage and Tuning in Machine Learning Research," in *2023 IEEE/ACM 2nd International Conference on AI Engineering – Software Engineering for AI (CAIN)*, IEEE, May 2023, pp. 68–79. doi: 10.1109/CAIN58948.2023.00016.

[29] M. Almousa, T. Zhang, A. Sarrafzadeh, and M. Anwar, "Phishing website detection: How effective are deep <scp>learning-based</scp> models and hyperparameter optimization?," *Secur. Priv.*, vol. 5, no. 6, p. e256, Nov. 2022, doi: 10.1002/spy2.256.

[30] Y. Rimal, N. Sharma, and A. Alsadoon, "The accuracy of machine learning models relies on hyperparameter tuning: student result classification using random forest, randomized search, grid search, bayesian, genetic, and optuna algorithms," *Multimed. Tools Appl.*, vol. 83, no. 30, pp. 74349–74364, Feb. 2024, doi: 10.1007/s11042-024-18426-2.

[31] N. F. Almujahid, M. A. Haq, and M. Alshehri, "Comparative evaluation of machine learning algorithms for phishing site detection," *PeerJ Comput. Sci.*, vol. 10, p. 2131, Jun. 2024, doi: 10.7717/peerj-cs.2131.

[32] W. Fu, V. Nair, and T. Menzies, "Why is Differential Evolution Better than Grid Search for Tuning Defect Predictors?," Mar. 2017, p. 13. doi: 10.48550/arXiv.1609.02613.

[33] M. Adnan, A. A. S. Alarood, M. I. Uddin, and I. ur Rehman, "Utilizing grid search cross-validation with adaptive boosting for augmenting performance of machine learning models," *PeerJ Comput. Sci.*, vol. 8, p. e803, Feb. 2022, doi: 10.7717/peerj-cs.803.

[34] S. F. M. Radzi, M. K. A. Karim, M. I. Saripan, M. A. A. Rahman, I. N. C. Isa, and M. J. Ibahim, "Hyperparameter Tuning and Pipeline Optimization via Grid Search Method and Tree-Based AutoML in Breast Cancer Prediction," *J. Pers. Med.*, vol. 11, no. 10, p. 978, Sep. 2021, doi: 10.3390/jpm11100978.

[35] B. Bischl *et al.*, "Hyperparameter optimization: Foundations, algorithms, best practices, and open challenges," *WIREs Data Min. Knowl. Discov.*, vol. 13, no. 2, p. e1484, Mar. 2023, doi: 10.1002/widm.1484.

[36] A. A. Albishri and M. M. Dessouky, "A Comparative Analysis of Machine Learning Techniques for URL Phishing Detection," *Eng. Technol. Appl. Sci. Res.*, vol. 14, no. 6, pp. 18495–18501, Dec. 2024, doi: 10.48084/etasr.8920.

[37] R. Mohammad and L. McCluskey, "Phishing Websites," UCI Machine Learning Repository. [Online]. Available at: https://archive.ics.uci.edu/dataset/327/phishing+websites.

[38] V. R. Joseph and A. Vakayil, "SPlit: An Optimal Method for Data Splitting," *Technometrics*, vol. 64, no. 2, pp. 166–176, Apr. 2022, doi: 10.1080/00401706.2021.1921037.

[39] M. Ahsan, M. Mahmud, P. Saha, K. Gupta, and Z. Siddique, "Effect of Data Scaling Methods on Machine Learning Algorithms and Model Performance," *Technologies*, vol. 9, no. 3, p. 52, Jul. 2021, doi: 10.3390/technologies9030052.

[40] H. Alibrahim and S. A. Ludwig, "Hyperparameter Optimization: Comparing Genetic Algorithm against Grid Search and Bayesian Optimization," in *2021 IEEE Congress on Evolutionary Computation (CEC)*, IEEE, Jun. 2021, pp. 1551–1559. doi: 10.1109/CEC45853.2021.9504761.

[41] Y. Zhao, W. Zhang, and X. Liu, "Grid search with a weighted error function: Hyper-parameter optimization for financial time series forecasting," *Appl. Soft Comput.*, vol. 154, no. 1, p. 111362, Mar. 2024, doi: 10.1016/j.asoc.2024.111362.

[42] C. Catal, G. Giray, B. Tekinerdogan, S. Kumar, and S. Shukla, "Applications of deep learning for phishing detection: a systematic literature review," *Knowl. Inf. Syst.*, vol. 64, no. 6, pp. 1457–1500, Jun. 2022, doi: 10.1007/s10115-022-01672-x.

[43] D. Sehrawat and Y. Singh, "Comparative Analysis on Fraud Detection in Credit Card Transaction Using Different Machine Learning Algorithms," in *Lecture Notes in Networks and Systems*, vol. 425, Springer, Singapore, 2022, pp. 673–684. doi: 10.1007/978-981-19-0707-4_61.

[44] D. Miyamoto, H. Hazeyama, and Y. Kadobayashi, "An Evaluation of Machine Learning-Based Methods for Detection of Phishing Sites," *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 5506 LNCS, no. PART 1, pp. 539–546, 2009, doi: 10.1007/978-3-642-02490-0_66.

[45] M. Vijayalakshmi, S. Mercy Shalinie, M. H. Yang, and R. M. U., "Web phishing detection techniques: a survey on the state-of-the-art, taxonomy and future directions," *IET Networks*, vol. 9, no. 5, pp. 235–246, Sep. 2020, doi: 10.1049/iet-net.2020.0078.

[46] L. Tang and Q. H. Mahmoud, "A Survey of Machine Learning-Based Solutions for Phishing Website Detection," *Mach. Learn. Knowl. Extr.*, vol. 3, no. 3, pp. 672–694, Aug. 2021, doi: 10.3390/make3030034.

[47] A. K. Dutta, "Detecting phishing websites using machine learning technique," *PLoS One*, vol. 16, no. 10, p. e0258361, Oct. 2021, doi: 10.1371/journal.pone.0258361.

[48] A. Zamir *et al.*, "Phishing web site detection using diverse machine learning algorithms," *Electron. Libr.*, vol. 38, no. 1, pp. 65–80, Mar. 2020, doi: 10.1108/EL-05-2019-0118.

[49] T. Nagunwa, "Comparative Analysis of Nature-Inspired Metaheuristic Techniques for Optimizing Phishing Website Detection," *Analytics*, vol. 3, no. 3, pp. 344–367, Aug. 2024, doi: 10.3390/analytics3030019.

[50] R. G. Jimoh *et al.*, "Efficient Ensemble-based Phishing Website Classification Models using Feature Importance Attribute Selection and Hyper parameter Tuning Approaches," *J. Inf. Technol. Comput.*, vol. 4, no. 2, pp. 1–10, Dec. 2023, doi: 10.48185/jitc.v4i2.891.

[51] D. Al Ahmare, S. A. Lashari, A. Khan, and S. Salah-Uddin, "Hyper Parameters Tuning Using Partial Swarm Optimization Algorithm Based on Random Forest for URLs-Based Phishing Detection," *J. Xi'an Shiyou Univ. Nat. Sci. Ed.*, vol. 67, no. 05, pp. 149--???, May 2024, doi: 10.5281/zenodo.11273166.

[52] R. Pavan, M. Nara, S. Gopinath, and N. Patil, "Bayesian Optimization and Gradient Boosting to Detect Phishing Websites," in *2021 55th Annual Conference on Information Sciences and Systems (CISS)*, IEEE, Mar. 2021, pp. 1–5. doi: 10.1109/CISS50987.2021.9400317.

[53] M. Al-Sarem *et al.*, "An Optimized Stacking Ensemble Model for Phishing Websites Detection," *Electronics*, vol. 10, no. 11, p. 1285, May 2021, doi: 10.3390/electronics10111285.