Inform : Jurnal Ilmiah Bidang Teknologi Informasi dan Komunikasi
Vol.11 No.1, January 2026, P-ISSN : 2502-3470, E-ISSN : 2581-0367

Systematic XGBoost Pipeline for Phishing Website Detection: Hyperparameter Tuning Approach with Nested Cross-Validation

Najma Aura Dias Prameswari[1], Wildanil Ghozi[2], Fauzi Adi Rafrastara[3]

[1,2,3]*Computer Science Department, Universitas Dian Nuswantoro, Semarang, Indonesia*

[1]1112214532@mhs.dinus.ac.id

[2]wildanil.ghozi@dsn.dinus.ac.id (*)

[3]fauziadi@dsn.dinus.ac.id

Received: 2025-11-19; Accepted: 2026-01-27; Published: 2026-02-02

Abstract— Phishing attacks have become increasingly sophisticated and pose a critical threat to cybersecurity, with more than 4.7 million attacks reported in 2023. Traditional blacklist and rule-based detection struggles to keep pace with evolving URL patterns and impersonation techniques. Rather than proposing a new classifier, this study presents a systematic and reproducible XGBoost-based phishing detection pipeline intended as an academic baseline with operationally motivated evaluation (not a production-integrated system). The Mendeley Phishing Websites dataset (58,645 URLs; 30,647 phishing and 27,998 legitimate) with 111 URL- and website-based features. The pipeline applies data cleaning, column-transformer-based pre-processing, and a stratified 80:20 train–test split, with all pre-processing steps fit on the training data only to reduce leakage risk. The final model uses 98 active features after removing 13 constant attributes; quasi-constant features are analyzed and retained. Continuous features are sanitised, log-transformed, and standardised, while binary features are left unchanged. Hyperparameters are tuned via stratified cross-validation using the ROC-AUC metrics, followed by early stopping, probability calibration, and simple threshold tuning. On the hold-out test set, the optimized model, set at a 0.50 decision threshold, achieves 96.34% accuracy, 96.31% precision, 96.70% recall, and 96.51% F1-score, improving over a default XGBoost baseline and yielding fewer false positives and false negatives. These results show that a systematically designed XGBoost pipeline provides a strong and reproducible baseline for URL-based phishing website detection and offers a practical foundation for future work on cost-sensitive learning and temporal validation. This study is limited to tabular URL/website feature-based detection and does not include visual content analysis, HTML/DOM parsing, or deep learning on raw text/images.

Keywords— URL-Based Phishing Detection; XGBoost; Hyperparameter Tuning; Nested Cross-Validation.

I. INTRODUCTION

A phishing website is a counterfeit site designed to resemble an official page, deceiving users into surrendering their credentials and sensitive data [1]. Over the past two years, research on phishing detection based on machine learning and deep learning has resurged, as fraud techniques have become more adaptive and harder to detect manually [2][3]. Recent surveys show that phishing-website detection remains a significant cybersecurity issue and has spurred the emergence of feature-based, graph-based, ensemble, and boosting approaches [2][4][5]. Among these, Gradient Boosting and XGBoost are frequently reported to improve the reliability of identifying fake pages across various URL/website corpora [6]-[8]. In addition, global counts of phishing websites have trended upward over recent quarters [9].

The Anti-Phishing Working Group (APWG) quarterly report underscores the scale and persistence of the threat, noting 1,077,501 phishing attacks in Q4 2023 alone [9]. APWG also recorded that 2023 approached the 2022 record as "the worst year" with more than 4.7 million attacks, indicating that phishing remains a dominant attack vector across sectors [9]. This rising trend is consistent with observations in the literature, which show an increase in phishing activity from 2021 through early 2024 [9]. Beyond growth in quantity, phishing patterns are increasingly sophisticated and adaptive, for example, impersonating the Okta Single Sign-On (SSO) portal and targeting specific agencies or industries such as the Federal Communications Commission (FCC) and cryptocurrency exchanges, complete with CAPTCHA, SMS, and real-time interactions to bypass Multi-Factor Authentication (MFA) [10]. In the crypto domain, seed phrases have also become targets, as Ripple's Chief Technology Officer (CTO) warned following a multimillion-dollar theft that triggered a surge in phishing campaigns disguised as "upgrade/verification" prompts for hardware wallets [11]. Motivated by these real-world patterns, this study focuses on a feature-based detection setting and emphasises systematic tuning, validation, and threshold selection to produce a robust, reproducible baseline. These findings underscore the importance of using appropriate feature representations and rigorous validation to obtain reliable performance estimates and reduce overfitting [2][7][12].

Many anti-phishing systems in the field still rely on blacklists or whitelists, heuristic rules, or third-party services. Such strategies tend to become stale quickly and are less resilient to zero-day threats, motivating a shift to machine learning or deep learning approaches that learn directly from URL/website features [1][2][4][5]. The literature also emphasises that overfitting and ad hoc hyperparameter tuning often bottleneck performance. Consequently, hyperparameter search schemes such as GridSearchCV or more advanced optimization should be treated as core components of a modern phishing-detection pipeline [2][6][13]. Recent reviews further

DOI : https://doi.org/10.25139/inform.v11i1.11221

emphasise the need for feature-based approaches [5] and rigorously validated machine learning models to achieve reliable generalisation within the adopted evaluation setting and to minimise false alarms [14]-[17]. This gap motivates the development of a systematically calibrated and well-tuned machine learning pipeline for phishing detection [2][8][12]. Accordingly, there is urgency to design calibrated, cost-efficient, and well-tuned detection pipelines [7][13].

Recent research on machine-learning-based phishing detection has increasingly positioned the boosting family as the strongest candidates [4][6][7]. Contemporary surveys indicate that XGBoost consistently excels for balanced-data scenarios due to its high precision and fast processing time [4][6][18]. This algorithm is widely used on URL/website corpora sourced from public repositories such as PhishTank, OpenPhish, and Mendeley Data [4]-[6][19].

The study "Deteksi Website Phishing Menggunakan Teknik Machine Learning" compares three algorithms: Decision Tree, Random Forest, and XGBoost, for detecting phishing websites using URL features [18]. The dataset employed is the "Datasets for Phishing Websites Detection" from Mendeley Data [19]. The balanced variant contains 58,645 instances, comprising 30,647 phishing and 27,998 legitimate, with 111 features in six categories [19]. The research process involves three stages: URL feature extraction and model training, starting from a baseline with default parameters, exploring hyperparameter tuning using RandomizedSearchCV, and then targeted refinement via GridSearchCV [18][20]. Evaluation metrics include accuracy, precision, recall, and F1-Score, and XGBoost achieves the best performance with 96.14% accuracy, 96.17% precision, 96.46% recall, and 96.32% F1-Score [14][18]. It is then integrated into a web-based Flask prototype and a Telegram bot [18][21]. The main contribution is an adversarial mechanism for automatic labelling and retraining, enabling the model to adapt to new attack patterns [18]. Its limitations lie in the focus on URL-based detection and an implementation scope confined to those two application channels. Because of the parity in dataset variant and the post-training adaptivity contribution, this study is designated as the primary state-of-the-art [18].

The study [6] focuses on improving phishing detection by integrating XGBoost with various combinations of feature-selection techniques and model interpretability. The authors utilise two data sources, PhishTank 2024 and UCI, comprising a total of 103,055 URLs (67,678 phishing and 35,377 benign), and employ explanatory mechanisms based on Local Interpretable Model-Agnostic Explanations (LIME) and Shapley Additive exPlanations (SHAP) [6][12]. The model achieves 99.80% accuracy, 100% precision, 99.5% recall, and 99.7% F1-Score, with an MCC of 0.996 [6]. The limitations of this study include the differences in datasets and their preprint status, which make it not directly comparable to this research [6].

The study [4] compares seven machine learning algorithms: Gradient Boosting, Random Forest, Decision Tree, KNN, SVM, Logistic Regression, and Naïve Bayes for detecting phishing domains. The dataset used is UCI Phishing Domains, and the results show that Gradient Boosting achieves 97.2% accuracy, 96.8% precision, 97% recall, and 96.9% F1-Score, Random Forest attains 97.1% accuracy with 97.2% precision, 97.4% recall, and 97.3% F1-Score, Decision Tree attains 96.3% accuracy, 96.6% precision, 96.7% recall, and 96.7% F1-Score, KNN attains 95.6% accuracy, 95.7% precision, 96.8% recall, and 96.2% F1-Score, SVM attains 93.9% accuracy, 93.7% precision, 96.4% recall, and 95% F1-Score, Logistic Regression attains 92.7% accuracy, 92.7% precision, 95% recall, and 93.8% F1-Score, while Naïve Bayes attains 60.1% accuracy, 99.2% precision, 29.3% recall, and 45.3% F1-Score [4]. Despite the strong performance, this study is not considered state-of-the-art because it uses a different dataset and does not discuss XGBoost configuration in depth [4].

The study [22] proposes a malicious-website detection framework that addresses high-dimensional features by applying firefly-based feature selection and optimising XGBoost hyperparameters using Particle Swarm Optimisation (PSO), yielding the PSO–XGBoost model. The experiments use a large dataset of more than 36,000 websites with 79 features, covering the categories benign, defacement, malware, phishing, and spam. In the binary evaluation setting (benign vs. malicious), performance is compared across several classification techniques, including J48, Random Tree, Naïve Bayes, Multi-Layer Perceptron (MLP), and Radial Basis Function Network (RBF). J48 achieves 94.91% accuracy, 93.64% kappa, 94.9% average precision, 94.9% average F1, and 94.9% average recall; Random Tree achieves 94.64% accuracy, 93.30% kappa, 94.6% average precision, 94.6% average F1, and 94.6% average recall; Naïve Bayes achieves 56.42% accuracy, 45.14% kappa, 58.4% average precision, 56.4% average F1, and 54.2% average recall; MLP achieves 80.64% accuracy, 75.75% kappa, 80.7% average precision, 80.6% average F1, and 80.6% average recall; and RBF achieves 60.64% accuracy, 50.52% kappa, 61.3% average precision, 60.6% average F1, and 59.4% average recall. Meanwhile, the proposed PSO–XGBoost achieves the best results with 98.42% accuracy, 98.02% kappa, 98.4% average precision, 98.4% average F1, and 98.4% average recall. Although the performance is high and relevant from an optimisation perspective, this study is not selected as the state of the art because it targets general malicious-website detection (benign vs malicious) on a different dataset. In contrast, this study focuses on phishing website detection (phishing vs. legitimate) using the Mendeley Phishing Websites Dataset (balanced variant).

The study [5] maps the phishing-detection landscape across various approaches, including blacklist or whitelist, feature-based, concept-based, machine learning/deep learning, ensemble, and graph-based. The survey also summarizes common corpora such as PhishTank, OpenPhish, UCI, Alexa/legitimate lists, and Mendeley, along with evaluation metrics [5]. The main findings indicate that tree-boosting or ensemble models with informative feature selection tend to excel on lexical URL features. At the same time, deep learning approaches are effective on large data but risk overfitting and incur high training costs [3][5][23]. Highlighted research gaps

DOI : https://doi.org/10.25139/inform.v11i1.11221

include class imbalance and dataset bias, zero-day detection, concept drift, the lack of cross-corpus validation, and inconsistent reporting of hyperparameters [5]. As this is a review without empirical experiments, this journal serves as the conceptual foundation of the present study [5].

Despite advances in boosting-based phishing detection, three research gaps remain [2][5]. First, many studies prioritise accuracy metrics at default thresholds without systematically exploring threshold optimisation to balance precision and recall across varying operational requirements (e.g., minimising false alarms vs. maximising threat detection) [14][15]. Second, hyperparameter tuning often lacks systematic nested validation, risking overfitting to validation sets and overestimating generalization performance [16][17]. Third, computational efficiency and reproducibility details (training time, resource requirements) are rarely reported, limiting practical adoption in resource-constrained environments [5].

To address these gaps, this study aims to develop a systematic and reproducible XGBoost pipeline for URL-based phishing detection. The primary objective is to establish a reproducible academic baseline with operational evaluation considerations. The study systematically optimises decision thresholds to balance precision and recall across varying operational requirements, addressing the static-threshold limitation in prior studies. To prevent hyperparameter tuning bias through nested cross-validation, ensure unbiased performance estimates on a hold-out test set. To improve reproducibility by documenting hardware specifications, training time, complete feature lists, and final model configurations. These objectives directly address the identified research gaps: systematic threshold optimization tackles the static-threshold limitation, nested cross-validation prevents tuning bias and overfitting, and comprehensive documentation improves reproducibility. This study focuses on tabular feature-based detection using URL and website attributes (domain-based, directory-based, file-based, parameter-based) and explicitly excludes visual content analysis, HTML/DOM parsing, and deep learning on raw text/images.

While XGBoost itself is widely used as a classifier, this work's contribution lies in a systematic, reproducible pipeline design. Specifically, the study combines feature cleaning based on statistical analysis, exhaustive GridSearchCV with nested cross-validation to reduce tuning bias, and validation-based threshold optimization for operationally relevant evaluation. This study aims to establish an XGBoost-based pipeline on the Mendeley Phishing Websites Dataset [19][24]. The principal contributions are: rigorous feature cleaning with statistical analysis to remove non-informative features [25], systematic hyperparameter tuning via exhaustive GridSearchCV with nested cross-validation to prevent overfitting [16][26], threshold optimization to maximize F1-Score on validation data [14][15], and comprehensive evaluation with detailed confusion matrix analysis on a hold-out test set [27][28]. This approach provides a strong, reproducible, and computationally documented baseline for URL-based phishing detection, intended for benchmarking and evaluation rather than production integration [29].

II. RESEARCH METHODOLOGY

The methodology for this study is organised into several key stages: experimental setup, dataset preparation, pre-processing, modelling, hyperparameter tuning, model development, model refinement, and evaluation. Each stage plays a crucial role in ensuring comprehensive performance gains for phishing detection using the XGBoost algorithm with a nested cross-validation approach [16][24]. Fig.1 summarises the overall research methodology, and the following subsections describe each stage in detail, including their rationale and implementation.

To complement the workflow diagram, Algorithm 1 provides a concise, step-by-step specification of the full pipeline, from pre-processing and tuning to early stopping, calibration/thresholding (if applicable), and final evaluation, so the procedure can be reproduced.

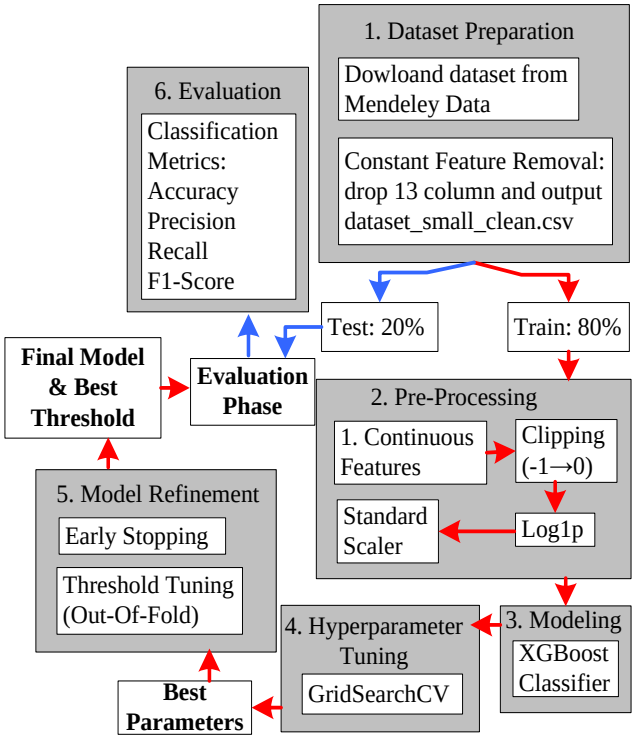


Fig.1. Research Methodology

A. *Experimental Setup*

Experiments were conducted using Python 3.11.7, with the primary libraries scikit-learn 1.3.0, XGBoost 2.0.0, pandas 2.0.3, NumPy 1.24.3, and Matplotlib 3.7.2 for visualization [24][30]. All computations were performed on a Windows 10 64-bit system with an Intel® Core™ i5-3230M processor (2 cores, 4 logical processors), 12 GB of RAM, and a 512 GB SSD. The environment was managed with Anaconda (version 24.11.3). The total GridSearchCV computation time was approximately 33 hours, constituting a one-time offline optimization cost for production deployment. GPU acceleration was not used (CPU-only).

DOI : https://doi.org/10.25139/inform.v11i1.11221

Algorithm 1: Systematic XGBoost Pipeline for URL/Website Feature-Based Phishing Detection

Input: Tabular dataset *D* with URL/website features (raw: 111 features) and *labels* $y \in \{0,1\}$; *(0 = legitimate, 1 = phishing)*
1: Dataset preparation and cleaning
2: Load dataset *D* (Mendeley Phishing Websites Dataset, balanced variant).
3: Identify and remove constant features (drop 13 constant columns).
4: Obtain cleaned feature matrix Xclean with 98 active features (92 continuous, 6 binary).
5: Hold-out split
6: Perform stratified train/test split: *(Xtrain, ytrain) = 80%, (Xtest, ytest) = 20%, random_state = 42.*
7: Pre-processing definition (applied without test leakage)
8: Define pre-processing pipeline *P*: *Continuous features (92): sanitize (-1 → 0) → log1p → StandardScaler; Binary features (6): passthrough (no scaling)*
Where, *P* is always fitted using only training data.
9: Hyperparameter selection (training only)
10: Run GridSearchCV on training set (*Xtrain, ytrain*): *Inner CV: StratifiedKFold(n_splits=10, shuffle=True, random_state=42); Selection metric: mean ROC-AUC.* Obtain the best hyperparameters θbest for XGBoost.
11: Model refinement (training only)
12: Early stopping to determine effective boosting rounds.
13: Create internal validation split from training: *(Xtr2, ytr2) = 80% of training, (Xval2, yval2) = 20%* of training (stratified).
14: Fit preprocessing *P on (Xtr2, ytr2),* then transform *Xtr2* and *Xval2*.
15: Train XGBoost with *θbest* and: *n_estimators_max = 5000, early_stopping_rounds = 100, eval_metric = "aucpr"; eval_set = (Xval2, yval2).*
Record *best_iteration = Nbest (effective n_estimators)*.
16: Final refit on full training set
17: Construct final model *M* with *θbest and n_estimators = Nbest* (early stopping disabled in this final refit).
18: Fit pipeline (*P + M*) on full training data (*Xtrain, ytrain*).
19: Probability calibration (training only)
20: Apply isotonic calibration to the refit pipeline using: *CalibratedClassifierCV(method="isotonic", cv=3) on (Xtrain, ytrain)*. Obtain a calibrated model Mcal
21: Threshold optimization (training only, out-of-fold)
22: Generate out-of-fold (OOF) probabilities on training set: *probaOOF = cross_val_predict(Mcal, Xtrain, ytrain, cv=3, method="predict_proba")[:,1].*
23: Define threshold grid T = linspace(0.20, 0.80, 121) (step = 0.005).
24: For each t in T: *compute F1(t)* using *ytrain vs 1(probaOOF ≥ t)*.
25: Select tuned threshold: *t* = argmax_t F1(t).*
26: Final evaluation on hold-out test set
27: Compute test probabilities: *probaTest = Mcal.predict_proba(Xtest)[:,1].*
28: Evaluate predictions at two operating points: *Default threshold 0.50; Tuned threshold t**
29: Report confusion matrices and classification metrics: Accuracy, Precision, Recall, and F1-Score for both thresholds.
Output: Final calibrated XGBoost model Mcal; Tuned decision *threshold t**; Test-set evaluation metrics at *thresholds 0.50 and t**

B. Dataset Preparation

The dataset used in this study is sourced from the Mendeley Data Repository under the title "Mendeley Phishing Websites Dataset" [19]. Its features represent URL and website characteristics spanning six main categories: URL-based features, domain-based features, directory-based features, file-based features, parameter-based features, and resolving data/external metrics features [19]. These categories include lexical attributes such as URL length, counts of specific special characters, indicators of IP address presence in the URL, domain age information, and other external attributes [12][19]. Before splitting the dataset into training and test sets, a data-cleaning stage was performed to remove non-informative features that could introduce noise into model learning [25].

This cleaning stage comprised three primary analyses: constant-feature removal to eliminate features with identical values across all samples; duplicate-feature analysis to identify features with identical value patterns; and quasi-constant analysis to detect features with extremely high single-value dominance [25]. Separating cleaning from modelling is important to ensure that cleaning decisions are not influenced by model performance and can be reproduced independently [31].

1) Initial Dataset Characteristics: The initial dataset consists of 58,645 rows and 112 columns, including 111 features and 1 binary target column (phishing vs. legitimate) [19]. The class distribution is relatively balanced, with 30,647 phishing (52.3%) and 27,998 legitimate (47.7%) instances [19]. The complete pre-cleaning characteristics are presented in Table I.

TABLE I
PRE-CLEANING DATASET CHARACTERISTICS

Rows	Cols	Feature	Target
58,645	112	111	1

2) Constant Feature Removal (13 features): The first analysis identified constant features, i.e., features whose values are identical across all 58,645 samples [25]. For each column $\boldsymbol{X_j}$, let $\boldsymbol{n_{unique}}(\boldsymbol{X_j})$ denote the number of distinct values observed in that column and $\boldsymbol{dom}(\boldsymbol{X_j})$ denote the maximum relative frequency of any values in the column. A column is classified as constant when $\boldsymbol{n_{unique}}(\boldsymbol{X_j}) = \mathbf{1}$ and $\boldsymbol{dom}(\boldsymbol{X_j}) = \mathbf{1.0}$, meaning that all samples share exactly the same value. The analysis identified 13 constant features, all of which took the value 0 and belonged to the domain-based category. These features represent special character counts in the domain component:

a) slash (qty_slash_domain)
b) question mark (qty_questionmark_domain)
c) equal sign (qty_equal_domain)
d) ampersand (qty_and_domain)
e) exclamation mark (qty_exclamation_domain)
f) space (qty_space_domain)
g) tilde (qty_tilde_domain)
h) comma (qty_comma_domain)

DOI : https://doi.org/10.25139/inform.v11i1.11221

i) plus sign (qty_plus_domain)
j) asterisk (qty_asterisk_domain)
k) hashtag (qty_hashtag_domain)
l) dollar sign (qty_dollar_domain)
m) per cent sign (qty_percent_domain).

Because constant features provide no discriminative information for classification, they only increase dimensionality without making a predictive contribution and may slow down computation and increase the risk of overfitting [25]. Therefore, all 13 constant features were removed [25]. After removing the 13 constant features, the dataset was reduced from 111 features to 98 active features. The class distribution remained unchanged because only feature columns were removed, not rows.

3) Statistical Analyses: Following constant-feature removal, a duplicate-features analysis was conducted to identify groups of features with identical value patterns across all samples [25]. To detect duplication, each column was hashed and grouped by identical hash values, then row-wise equality was verified: $\boldsymbol{h_j = hash(X_j)}$; a duplicate is flagged if $\boldsymbol{h_j = h_k}$ $\boldsymbol{and}$ $(\boldsymbol{x_{ij} = x_{ik}}\ \forall_i)$. The analysis found one duplicate group of six features:

a) qty_dollar_file
b) qty_hashtag_directory
c) qty_hashtag_file
d) qty_questionmark_directory
e) qty_questionmark_file
f) qty_slash_file

These features have identical values across all rows, indicating that the special-character indicators occur in the same pattern in the file and directory components of the URL. Although these features were detected as duplicates, this study decided not to remove them for two main reasons. First, the scope focuses only on cleaning constant features that truly exhibit no variation in value, whereas duplicate features still exhibit variation, even when they are identical across features [25]. Second, tree-based algorithms, such as XGBoost, are relatively robust to multicollinearity and redundancy because of their internal feature-importance and regularisation mechanisms [24]. XGBoost naturally assigns lower weights to redundant features during training; therefore, duplicate features should not significantly impair performance [12][24]. This finding is documented for transparency and may inform follow-up studies focusing on dimensionality reduction. The final analysis concerns quasi-constant features, defined as columns with extremely high dominance of a single value while still exhibiting some variation [25]: $\boldsymbol{dom\,X_j \geq 0.995}$ and $\boldsymbol{n_{unique}\,(x_j) > 1}$, meaning the most frequent value (top-value frequency) covers more than 99.5% of all samples. The analysis identified 15 quasi-constant features across multiple categories:

a) qty_at_domain
b) qty_hashtag_url
c) qty_underline_domain
d) qty_space_url
e) qty_dollar_url
f) qty_asterisk_url
g) url_google_index
h) qty_comma_url
i) qty_plus_url
j) domain_google_index
k) qty_exclamation_url
l) domain_in_ip
m) server_client_domain
n) qty_tilde_ur
o) url_shortened

Unlike constant and duplicate features, these quasi-constant features were retained in the dataset for several reasons. First, despite the very high dominance of a single value, they still have variation ($n_{unique} > 1$) that may carry discriminative information; small variations can be strong indicators for certain classes [12][25]. Second, this approach aligns with related studies that utilize implicit feature selection through tree-based feature importance [12][24]. XGBoost can automatically evaluate each feature's contribution during training and assign appropriate weight; if a quasi-constant feature is uninformative, the model will naturally assign it a low importance score without manual removal [24]. Third, retaining these features preserves the model's flexibility to explore the full feature space; their actual contributions can be examined post hoc via feature-importance analysis once the final model is trained [12].

4) Final Dataset for Model Development: After completing all cleaning stages, constant feature removal, duplicate feature analysis, and quasi-constant analysis, the final dataset was saved as dataset_small_clean.csv. This dataset contains 58,645 samples with 98 active features (from the original 111 minus 13 constant features) plus 1 target column. The complete post-cleaning characteristics are presented in Table II.

TABLE II
POST-CLEANING DATASET CHARACTERISTICS

Rows	Cols	Feature	Target
58,645	99	98	1

5) Train-Test Split: Before pre-processing begins, dataset_small_clean.csv, which contains 58,645 samples, is split into two sets at an 80:20 ratio [26]. This yields 46,916 samples for the training set (80%) and 11,729 samples for the test set (20%), with stratified sampling to preserve class proportions [26]. The dataset is relatively balanced, though it shows a slight imbalance: 30,647 phishing samples (52.3%) and 27,998 legitimate samples (47.7%). Although this imbalance is not extreme, a mechanism is still required to prevent the model from biasing toward the majority class (phishing) during learning [24]. Complete class-distribution characteristics are shown in Table III.

TABLE III
DATASET LABEL DISTRIBUTION

Subset	Label	Count	Total
Train	0	22,398	46,916
Train	1	24,518	
Test	0	5,600	11,729
Test	1	6,129	

C. Pre-processing

In this study, all pre-processing is performed exclusively on the training set (80%) to prevent data leakage into the test set [31]. Transformation parameters, such as the mean and standard deviation for StandardScaler, are fit on the training set and then applied to the test set without refitting [30][31]. This approach follows the principle of nested cross-validation, in which the test set is fully held out until the final evaluation stage [16][31].

DOI : https://doi.org/10.25139/inform.v11i1.11221

1) Feature Classification: The first step in pre-processing is to classify features based on their value characteristics. After dataset cleaning, the 98 active features are divided into two categories: continuous and binary [30]. This classification is conducted by analyzing the number of unique values (n_unique) and the value composition of each feature using a helper function is_binary_series() that checks whether a column contains only (0, 1).

a) Continuous Features: Continuous features exhibit positively skewed distributions, with most values concentrated in the lower ranges and a few extremely high [32]. Exploratory analysis revealed three issues: (1) the presence of a special value −1 that is incompatible with logarithmic transforms; (2) highly skewed distributions with extreme outliers; (3) non-uniform scales that can cause large-scale features to dominate learning. To address these, a three-stage pre-processing pipeline was designed: clipping ($-1 \rightarrow 0$) for special-value sanitization, log1p transformation to reduce skewness, and StandardScaler for scale normalization [18][30][32]. The first stage is special-value sanitisation to handle the −1 value in the logarithmic transformation. This step preserves mathematical consistency, semantic coherence, and domain uniformity [32]. Sanitization uses a clipping operation that transforms all negative values, including -1, to 0, as defined in Equation (1). The second stage involves the log1p transformation. This is applied to all continuous features to reduce skewness, handle 0 values without mathematical errors, stabilize variance, and mitigate the influence of outliers [18][32], defined in Equation (2) [32]. The third stage is normalisation using StandardScaler to standardise all features. This step accelerates Gradient Boosting convergence, prevents large-scale features from dominating the learning process, improves numerical stability, and facilitates fair regularization between features [24][30]. Although the log1p transformation has reduced skewness, different features still have varying means and standard deviations. StandardScaler standardises each feature by setting its mean to 0 and its standard deviation to 1, also known as z-score normalisation using Equation (3) [30]. The parameters μ and σ are computed only from the training set using *.fit()* or *.fit_transform()*. This approach is crucial for preventing data leakage, as calculating parameters from a combination of training and test sets can inadvertently incorporate test-set information into the training process, leading to overly optimistic performance estimates [31].

$$x_{clipped} = \max(x, 0) = \begin{cases} 0, & if\ x < 0 \\ x, & if\ x \geq 0 \end{cases} \quad (1)$$

$$x' = log(1 + x) \quad (2)$$

$$z = \frac{x - \mu}{\sigma} \quad (3)$$

b) Binary Features: Binary features take only two unique values, 0 and 1, representing the presence or absence of specific attributes in the URL/website. Because they are already in an optimal format for machine-learning algorithms, binary features require no additional transformation [24]. In the pre-processing pipeline, binary features are handled via a passthrough mechanism in a ColumnTransformer, meaning they are forwarded from input to output without modification [30]. Transforms such as log1p or standardization are not applied because they could distort boolean (0, 1) values, while tree-based models like XGBoost naturally handle such features via deterministic splits at a 0.50 threshold [24]. This approach also improves computational efficiency without sacrificing feature information [24].

2) ColumnTransformer Pipeline: To ensure consistency and reproducibility of pre-processing, all transformation steps are combined into a single integrated pipeline using scikit-learn's ColumnTransformer [14][30]. The ColumnTransformer is fit only on the training set and applied to the test set to prevent data leakage [31]. It enables different transformers to be applied to different column subsets in a single operation, making it well-suited for scenarios with both binary and continuous features that require distinct treatments [30]. These two branches run in parallel and are then recombined into a single feature matrix totalling 98 active features: 6 binary and 92 continuous, ready for model training.

3) Feature-Target Correlation Analysis: After the full pre-processing pipeline was applied to the training set (46,916 samples), a correlation analysis was conducted to understand the strength of association between transformed features and the target label (0 = legitimate, 1 = phishing) [12]. Correlations were computed using the point-biserial correlation, a special case of Pearson's correlation that applies to a continuous and a binary variable [14]. The analysis shows that the top 10 continuous features most correlated with the phishing target are those related to the URL's directory and file components [12]. The five strongest positive correlations ($r > 0.6$), all indicating strong associations with the phishing label, are:

a) qty_slash_directory (0.69)
b) directory_length (0.685)
c) qty_slash_url (0.674)
d) qty_dot_file (0.643)
e) qty_dot_directory (0.635)

The analysis reveals three characteristic patterns of phishing URLs: complex directory structures used to obscure true intent or mimic legitimate sites, increased use of special characters for obfuscation or credential phishing, and path structure being more informative for phishing detection than the domain alone, as attackers may use a legitimate-looking domain with complex paths [1][2][12]. These findings are consistent with the literature, which shows that phishing URLs often exhibit complex, abnormal paths to evade blacklists or mask the true domain with long subdomains/paths [2][5][7]. With the pre-processing pipeline in place, the dataset is ready for the modelling stage, using the XGBoost Classifier with comprehensive hyperparameter tuning [24].

D. Modeling

For this study, the XGBoost (Extreme Gradient Boosting) algorithm was selected as the classification model for phishing detection [24]. XGBoost is a gradient-boosted decision tree (GBDT) implementation optimized for speed and performance on tabular data [24]. The model builds an ensemble of trees

DOI : https://doi.org/10.25139/inform.v11i1.11221

sequentially, where each new tree is trained to reduce the residual errors of the previous trees [24]. XGBoost minimises a regularisation objective function that combines a loss function and a complexity penalty, using first- and second-order information (gradients and Hessians) to efficiently determine optimal splits [24].

Formally, XGBoost constructs a predictive ensemble by sequentially adding the sum of *K* decision trees, as shown in Equation (4) [24]. the space of a regression tree (CART). Where, q represents the structure of each tree that maps an example to the corresponding leaf index, T is the number of leaves in the tree, and each f_k corresponds to an independent tree structure q and leaf weights w. Unlike decision tree, each regression tree contains a continuous score on each of the leaf; we use w_i to represent score on i-th leaf [24].

$$\hat{y}_i = \phi(x_i) = \sum_{k=1}^{K} f_k\ (x_i),\ f_k\ \in F \quad (4)$$

Where $F = \{f\ (x) = w_{q(x)}\}\ (q : \mathbb{R}^m \rightarrow T,\ w \epsilon \mathbb{R}^T)$ is

The model minimises a regularised objective function that combines the loss function ℓ with a regularisation term Ω (Equation (5)), balancing prediction accuracy and model complexity [24]. Where, $\Omega(f) = \gamma T + \frac{1}{2}\lambda \|w\|^2$. Here γ controls the complexity cost per leaf and λ is the L2 regularization parameter on leaf weights. The model is trained in an additive manner. At iteration t, we add f_t to minimize the loss shown in Equation (6) [24].

$$\mathcal{L}(\phi) = \sum_{i=1}^{n} \ell\ (\hat{y}_i, y_i) + \sum_k \Omega\ (f_k) \quad (5)$$

$$\mathcal{L}^{(t)} = \sum_{i=1}^{n} \ell\left(y_i, \hat{y}_i^{(t-1)} + f_t(x_i)\right) + \Omega(f_t) \quad (6)$$

XGBoost uses a second-order Taylor approximation for efficient optimization, as shown in Equation (7) [24]. Where $g_i = \partial_{\hat{y}}^{(t-1)} \ell\left(y_i, \hat{y}^{(t-1)}\right)$ and $h_i = \partial^2_{\hat{y}_{(t-1)}} \ell(y_i, \hat{y}^{(t-1)})$ are first and second order gradient statistics on the loss function. For a fixed tree structure $q(x)$, the optimal weight of leaf j is computed using Equation (8) [24]. Where $I_j = \{i|q(x_i) = j\}$ is the instance set of leaf j. The corresponding tree quality score is calculated using Equation (9) [24].

$$\mathcal{L}^{(t)} \approx \sum_{i=1}^{n} \left[g_i\ f_t\ (x_i) + \frac{1}{2}\ h_i\ f_t^2(x_i)\right] + \Omega(f_t) \quad (7)$$

$$w_j^* = -\frac{\sum i \epsilon I_j g_i}{\sum i \epsilon I_j^{h_i + \lambda}} \quad (8)$$

$$\check{\mathcal{L}}^{(t)}(q) = -\frac{1}{2}\sum_{j=1}^{T} \frac{(\sum i \epsilon I_j g_i)^2}{\sum i \epsilon I_j^{h_i + \lambda}} + \gamma T \quad (9)$$

This score measures the quality of a tree structure q and is used for split evaluation. The gain from splitting a leaf into left (I_L) and right (I_R) children's evaluation using Equation (10) [24].

$$\mathcal{L}_{split} = \frac{1}{2}\left[\frac{(\sum i \epsilon I_j g_i)^2}{\sum i \epsilon I_j^{h_i + \lambda}} + \frac{(\sum i \epsilon I_j g_i)^2}{\sum i \epsilon I_j^{h_i + \lambda}} - \frac{(\sum i \epsilon I_j g_i)^2}{\sum i \epsilon I_j^{h_i + \lambda}}\right] - \gamma \quad (10)$$

XGBoost offers several advantages for this task. First, it has built-in *L1* (alpha) and *L2* (lambda) regularisation that curbs model complexity and helps prevent overfitting, improving robustness when deployed to unseen data [24]. Second, its computational efficiency is high through histogram-based split finding, parallelization, and native handling of missing/sparse inputs, keeping training time short even with many features [24]. Third, granular control of tree structure, such as max_depth, min_child_weight, and gamma, helps maintain generalization [24]. Fourth, for class imbalance, parameters such as scale_pos_weight, subsample, and colsample_bytree help balance the contribution of the minority class without extensive feature engineering [24].

The model is integrated into a scikit-learn pipeline that unifies all pre-processing steps with the XGBoost classifier [30]. The pipeline design ensures that all transformations applied to the training set are consistently applied during validation and prediction, minimizing data leakage and facilitating deployment [30][31]. The final configuration is determined through a structured tuning procedure using GridSearchCV with early stopping to optimize the balance between n_estimators and learning_rate [20][30]. Evaluation is based on core classification metrics (accuracy, precision, recall, and F1) and optimal threshold selection, as discussed in the evaluation section [14][15].

E. Hyperparameter Tuning

Hyperparameter tuning was performed via GridSearchCV to systematically explore the hyperparameter space and identify the optimal configuration for the XGBoost model [20][30]. This tuning procedure constitutes the inner loop of the nested validation strategy, in which candidate configurations are evaluated exclusively on the training set, and the hold-out test set is never used [16][17]. GridSearchCV employed StratifiedKFold (*n_splits = 10*, *shuffle = True*, *random_state = 42*), training on nine folds and validating on the remaining fold for each configuration [26]. The selection criterion was the mean ROC-AUC across the 10 folds, providing a robust estimate of generalization performance for hyperparameter selection [33][34]. During this search, *n_estimators* was treated as an upper bound; after selecting the best hyperparameters, early stopping on an internal validation split of the training set was used to determine the effective number of boosting rounds (*best_iteration*) for the final refit [24]. Total hyperparameter combinations: 76,800; each evaluated using 10-fold stratified cross-validation with ROC-AUC scoring.

F. Model Development Phase

Fig. 2 summarises the proposed model development pipeline. All model development was conducted strictly on the training set (46,916 samples), while the test set (11,729 samples) was reserved as a hold-out for final evaluation [16].

DOI : https://doi.org/10.25139/inform.v11i1.11221

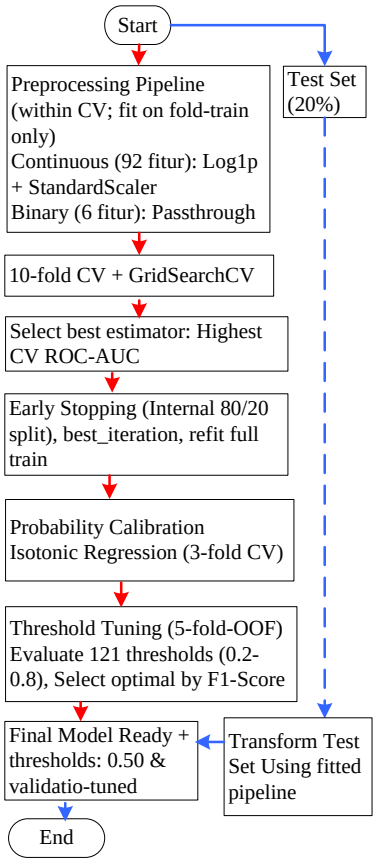


Fig.2. Model Development Phase

The dataset was first split into training and test sets using a stratified 80:20 split with a fixed random state of 42 to ensure reproducibility [26]. On the training path, within the inner 10-fold stratified cross-validation used for hyperparameter tuning, pre-processing was applied fold-wise as follows: continuous features (92 features) were sanitized ($-1 \rightarrow 0$), transformed with log1p to reduce skewness and normalized using StandardScaler (fit only on the fold-train data), while binary features (6 features) were passed through unchanged (passthrough) [26][30][32].

Hyperparameter tuning was performed using GridSearchCV with an exhaustive search over the parameter grid defined in Table IV [20][30]. Each hyperparameter configuration was evaluated using 10-fold stratified cross-validation, with the configuration achieving the highest mean ROC-AUC across the 10 folds selected as the optimal model [26][33]. After selecting the best hyperparameters, early stopping was performed using a stratified internal split of the training set, where 20% of the training data served as the validation set (eval_set) to monitor AUC-PR (Area Under Precision-Recall Curve) with a maximum of 5,000 trees and a patience of 100 rounds [24].

TABLE IV
GRIDSEARCHCV SPACE

Parameter	Values Tested
learning_rate	[0.08, 0.10, 0.15, 0.20, 0.25]
max_depth	[6, 8, 10, 12, 14]
n_estimators	[100, 200, 500, 800]
subsample	[0.7, 0.8, 0.9, 1.0]
colsample_bytree	[0.7, 0.8, 0.9, 1.0]
min_child_weight	[1, 2, 3, 5]
gamma	[0.0, 0.1, 0.3]
reg_lambda	[0.5, 1.0, 2.0, 3.0]

The best iteration obtained from this internal validation was used to set the effective number of trees, after which the final model was refit on the full training set with n_estimators fixed to this value (without using the test set at any stage) [24][31]. Probability calibration was then performed using Isotonic Regression (3-fold CV), followed by threshold optimisation on the training set using 3-fold cross-validated out-of-fold predicted probabilities, with 121 candidate thresholds evaluated in the range 0.20-0.80. The threshold maximizing F1-Score was selected (the hold-out test set was not used) [14][35][36]. Final evaluation on the test set was conducted at two operating points: the default threshold of 0.50 and the validation-tuned threshold [14][15]. This internal validation split was created independently from the 10-fold cross-validation used for hyperparameter tuning, ensuring that early stopping decisions were based on a hold-out portion of the training data without reusing cross-validation folds.

G. Model Refinement

Following hyperparameter tuning, the refinement stage aims to finalise the model before evaluation on the hold-out test set [24][36]. First, the XGBoost model was refit using the selected hyperparameters and an internal stratified validation split of the training set (20%) to apply early stopping and determine the effective number of boosting rounds (best_iteration) [24]. The final model was then refit on the full training set with n_estimators fixed to best_iteration [24]. Second, probability calibration was performed using Isotonic Regression with 3-fold cross-validation to improve the reliability of probabilities for threshold-based decision-making [35][36]. Third, threshold tuning was conducted on the training set using 3-fold out-of-fold predicted probabilities to select the threshold that maximized the F1-Score [14][15]. The output of this stage is a calibrated model and an optimal operating threshold ready for final evaluation on the test set [35][36].

H. Evaluation

Model evaluation is conducted on the hold-out test set (20%), which is never used during training, tuning, or refinement [16]. The test set is transformed using the pre-processing pipeline trained on the training set, without refitting, to avoid data leakage [31]. Evaluation is performed at two operating points: the default threshold of 0.50 and the validation-tuned optimal threshold, allowing a comparison of standard versus optimized performance [14][15].

Decision-based metrics used to evaluate performance are derived from the confusion matrix [27][28]. A confusion matrix presents the distribution of predictions in a 2×2 table, containing True Positive (TP), True Negative (TN), False Positive (FP), and False Negative (FN) [27][28]. From the confusion matrix, we calculate the four key performance

DOI : https://doi.org/10.25139/inform.v11i1.11221

metrics using Equations (11)-(14) [14][15]. Accuracy (Equation 11) measures the proportion of correct predictions out of all predictions [14][15]. Where, *N* denotes the total number of samples. Precision (Equation 12) measures the proportion of predicted positives that are correct [14][15]. Recall (Equation 13) measures the proportion of actual positives that are correctly detected [14][15]. F1-Score (Equation 14) measures the harmonic mean of Precision and Recall [14][15].

$$Accuracy = \frac{TP+TN}{N} \quad (11)$$

$$Precision = \frac{TP}{TP+FP} \quad (12)$$

$$Recall = \frac{TP}{TP+FN} \quad (13)$$

$$F1 = 2 \times \frac{Precision \times Recall}{Precision+Recall} \quad (14)$$

These metrics provide a comprehensive view of model performance. Accuracy measures overall correctness, Precision indicates reliability of positive predictions (important for minimizing false alarms), Recall measures completeness of positive detection (critical in cybersecurity to minimize missed phishing attacks), and F1 balances Precision and Recall [14][15][28].

I. Nested Cross-Validation Approach

This study employs a two-loop nested validation strategy to prevent data leakage and obtain an unbiased final evaluation [16][17]. The outer loop is implemented via an initial stratified 80:20 train–test split, with the test set held out until final evaluation [16][26]. The inner loop is implemented via 10-fold stratified cross-validation for hyperparameter tuning, operating only on the training set [17][26]. Model refinement steps (early stopping validation, calibration, and threshold selection) are also performed exclusively on the training set. In contrast, the hold-out test set is used only once for final reporting [16][17][31].

III. Result and Discussion

This chapter presents the experimental results of phishing detection using the XGBoost pipeline designed in Chapter 2 [24]. The primary objective is to assess the effectiveness of the nested scheme (a hold-out test as the outer loop, K-fold CV as the inner loop) for hyperparameter tuning and threshold selection on the Mendeley Phishing Websites dataset [16][19][26]. The evaluation focuses on decision-based metrics (accuracy, precision, recall, and F1) derived from confusion matrix analysis at two operating points: the default 0.50 and the tuned threshold 0.45 [14][15][27].

A. GridSearchCV Results

The exhaustive grid search with finer resolution. Due to its comprehensive nature, computation took 33 hours,12 minutes, and 0.9 seconds [20]. The results increased CV ROC-AUC to 99.29%, a marginal yet consistent improvement over the exploratory stage (+0.0124%), reinforcing the value of fine-grained tuning around the top candidate [33][34]. The optimal hyperparameter obtained from GridSearchCV is summarised in Table V.

Table V
Optimal Hyperparameters from GridSearchCV Space

Parameters	Value Tested
learning_rate	0.08
max_depth	8
n_estimators	800

B. Model Training and Refinement

After obtaining the optimal hyperparameter configuration from the grid stage, the model was refit on the entire training set and refined via three mechanisms: early stopping, probability calibration, and threshold tuning [24][35][36]. All steps were performed solely on the training/validation data (without touching the test set) to prevent data leakage [31].

1) Early Stopping: The optimal configuration from GridSearchCV (*learning_rate=0.08, max_depth=8, n_estimators=800*) was then trained with early stopping to determine the effective number of trees in a data-driven manner [24]. Early stopping was applied with a maximum of 5,000 trees and patience of 100 under an internal validation scheme [24]. Early stopping triggered at best_iteration = 950, well below the maximum of 5,000 trees, indicating convergence without unnecessary complexity. The difference from the initial GridSearchCV estimate (*n_estimators=800*) is expected: GridSearchCV provides an initial estimate based on fixed tree counts, while refitting with early stopping on the full training data allows the effective number of trees to adjust dynamically in response to the complete data signal [20][24]. The final configuration demonstrates several key characteristics for robust phishing detection [24]: (1) a moderate learning rate (0.08), which supports stable, incremental learning and prevents overshooting optimal solutions; (2) a relatively shallow tree depth (8) that helps curb overfitting by limiting individual tree complexity; (3) a sufficient tree count (950) that provides adequate model capacity while maintaining computational efficiency; (4) ensemble diversity: subsample=0.9 and colsample_bytree=0.7 introduce randomness in training data and feature selection; (5) local complexity controls: min_child_weight=1 and gamma=0.0 regulate individual split decisions. The final training configuration after early stopping is reported in Table VI. Following early stopping, probability calibration was performed using Isotonic Regression (3-fold CV) to ensure probability reliability for threshold-based decision making [35][36].

Table VI
Final Model Training Configuration With Early Stopping

Parameters	Value Tested
learning_rate	0.08
max_depth	8
n_estimators	950

2) Threshold Tuning: Threshold tuning evaluated 121 candidate thresholds (*0.20 to 0.80, step=0.005*) using out-of-fold predictions from the calibration process, with F1-score as the optimization criterion on validation data [14][15].

DOI : https://doi.org/10.25139/inform.v11i1.11221

Validation results indicated an optimal threshold of 0.45 with an F1-score of 96.35%. However, evaluation on the hold-out test set revealed that the default threshold of 0.50 achieved higher performance on most metrics (F1=96.51%), indicating slight validation optimism at the tuned threshold of 0.45 [17]. This finding underscores the importance of the nested cross-validation approach, where the test set remains fully isolated until final evaluation to provide unbiased performance estimates [16][17].

C. Model Evaluation

The final evaluation was conducted on the hold-out test set of 11,729 samples, which were never used during training, tuning, or refinement [16]. Evaluation was performed at two comparative operating points: the default threshold (0.50) and the tuned threshold (0.45) [14][15].

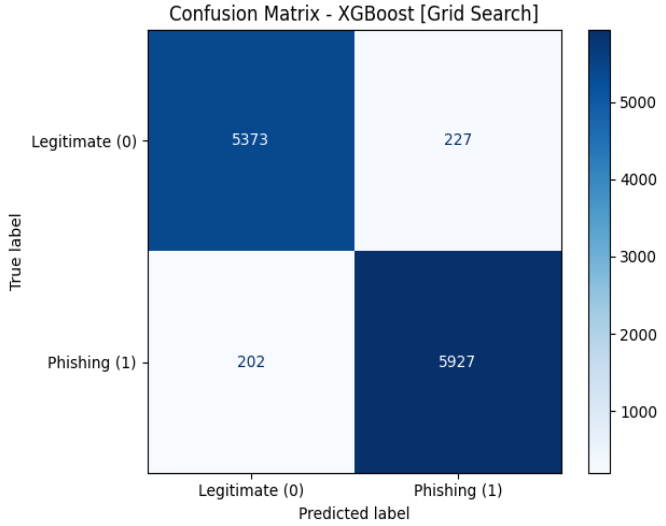


Fig.3. Confusion Matrix of Final Model

The confusion matrix in Fig.3 provides a detailed breakdown of the model's predictions across all test samples. Each cell represents a specific classification result [27][28]: (1) True Positives (TP = 5,927), where the model correctly detected 5,927 phishing URLs out of 6,129 actual phishing samples. This demonstrates the model's strong capability in identifying phishing threats [1][14]; (2) True Negatives (TN = 5,373), where the model correctly identified 5,373 legitimate URLs out of 5,600 actual legitimate samples. This indicates reliable recognition of genuine websites [14]; (3) False Positives (FP = 227), where 227 legitimate URLs were incorrectly flagged as phishing. This representing 4.05% of all legitimate samples. While false positives can cause user inconvenience by blocking access to safe websites, the relatively low rate indicates that the model maintains good precision [14][15]; (4) False Negatives (FN = 202), where the model failed to detect 202 phishing URLs, accounting for 3.30% of all phishing samples (202/6,129 = 0.0330). False negatives represent the most critical error type in cybersecurity applications, as they allow malicious sites to evade detection and potentially compromise users [1][2][8]. The low rate demonstrates the model's strong defensive capability [1][8].

1) Classification Metrics Calculation: For the confusion matrix presented above, the four key performance metrics – accuracy, precision, recall, and F1-Score – were computed according to the formulas defined in Section H (Evaluation). These metrics quantify how effectively the model distinguishes phishing from legitimate URLs. First, accuracy measures the proportion of correct predictions across all samples. On the test set, the model achieved an accuracy of 96.34%, indicating high overall classification performance across both classes [14][15]. Second, precision measures the reliability of positive predictions by calculating the proportion of correctly predicted phishing URLs among all URLs flagged as phishing. A precision of 96.31% means that when the model flags a URL as phishing, it is correct 96.31% of the time. This high precision is important for maintaining user trust and minimizing false alarms that could lead to alert fatigue [8][14][15]. Third, recall, also known as sensitivity or true positive rate, measures the completeness of phishing detection by calculating the proportion of actual phishing URLs that were successfully identified. A recall of 96.70% indicates that the model successfully detected 96.70% of all actual phishing URLs in the test set. High recall is critical in cybersecurity applications, as it minimizes the risk of malicious sites bypassing detection [1][2][8][14][15]. Finally, the F1-Score provides a single metric that balances precision and recall using their harmonic mean. The F1-Score of 96.51% confirms that the model achieves a good trade-off between precision and recall. Unlike the arithmetic mean, the harmonic mean used in the F1-Score penalizes extreme imbalances, ensuring that both precision and recall must be high for a strong overall score [14][15].

2) Performance Summary: The confusion matrix shows a balanced error distribution, with false positives (227 samples, 4.05%) and false negatives (202 samples, 3.30%) both at low levels [27][28]. This balance is particularly important for operational deployment, where both error types carry distinct costs [8][29]. The low false-negative rate (3.30%) is critical in cybersecurity contexts, as it minimises the risk of malicious sites evading detection [1][2][29]. Despite this emphasis on threat detection (high recall), the model maintains strong precision (96.31%), ensuring minimal false alarms [8][14]. This balance, reflected in the F1-Score of 96.51%, demonstrates the effectiveness of the systematic hyperparameter tuning and threshold optimization approach [14][15].

3) Threshold Comparison: On the test set, the default threshold of 0.50 achieved higher overall performance than the tuned threshold of 0.45. Specifically, threshold 0.50 achieved accuracy = 96.34%, precision = 96.31%, recall = 96.70%, and f1 = 96.51%, outperforming threshold 0.45 on accuracy (+0.15%), recall (+0.31%), and f1-score (+0.16%), while maintaining nearly identical precision (−0.01%). This performance advantage indicates slight validation optimism during threshold tuning on out-of-fold predictions, confirming that the default threshold of 0.50 provides more robust generalization to unseen data [17]. The superior performance of the default threshold validates the effectiveness of the

DOI : https://doi.org/10.25139/inform.v11i1.11221

developed pipeline. It demonstrates that systematic hyperparameter tuning with early stopping can yield models that perform reliably without requiring extensive threshold manipulation in production deployment [16][24][29].

TABLE VII
THRESHOLD PERFORMANCE COMPARISON ON TEST SET

Metrics	Threshold 0.45	Threshold 0.50	(0.50 – 0.45)
Accuracy	96.18%	96.34%	+ 0.16%
Precision	96.31%	96.31%	0.00%
Recall	96.39%	96.70%	+ 0.31%
F1-Score	96.35%	96.51%	+ 0.16%

D. Comparison with Baseline and State-of-the-Art

To assess the advantages of the proposed pipeline, we compare its test-set performance on the Mendeley Phishing Websites Dataset against reported XGBoost results from prior work (Default, RandomizedSearchCV, and GridSearchCV) in [18]. Table VIII reports the metrics at the final operating threshold (0.50), with the first three columns taken from the research [18] and the last column from this study. The findings are based on representative prior studies with different datasets and experimental settings. To provide broader context, Table IX summarizes these studies and highlights differences in data sources, evaluation protocols, and reporting practices.

TABLE VIII
COMPARISON BETWEEN THE PROPOSED PIPELINE AND REPORTED XGBOOST RESULTS IN PRIOR WORK ON THE MENDELEY PHISHING WEBSITES DATASET

Model	XGBoost (Default) [18]	XGBoost (Randomized) [18]	XGBoost (Grid) [18]	Proposed XGBoost (Grid)
Feature	111	111	111	98
Accuracy	96.14%	96.09%	96.14%	96.34%
Precision	96.17%	96.08%	96.17%	96.31%
Recall	96.46%	96.46%	96.46%	96.70%
F1-Score	96.32%	96.27%	96.32%	96.51%

TABLE IX
SUMMARY OF REPRESENTATIVE PREVIOUS STUDIES ON PHISHING WEBSITE DETECTION (DATASETS AND SETTINGS VARY)

Study	Dataset	Method/Focus	Result (Acc/P/R/F1)	Notes
Enhancing Phishing Detection: Integration of XGBoost with Feature Selection Techniques *(preprint)* [6]	PhishTank 2024 + UCI (103,055 URLs)	XGBoost + feature selection; interpretability (LIME/SHAP)	99.80% / 100% / 99.5% / 99.7%	Different datasets; preprint → not directly comparable.
Comparative Study of Machine Learning Algorithms for Phishing Website Detection [4]	UCI Phishing Domains	Compare 7 ML models (GB, RF, DT, KNN, SVM, LR, NB)	Best Random Forest: 97.1% / 97.2% / 97.4% / 97.3%	Different dataset; not focused on XGBoost configuration.
Safeguarding cyberspace: Enhancing malicious website detection with PSO optimized XGBoost and firefly-based feature selection [22]	Benign → Alexa top websites Spam → WEBSPAM-UK2007 Phishing → OpenPhish Malware → DNS-BH Defacement → Alexa	Firefly feature selection + PSO hyperparameter optimization + XGBoost	PSO–XGBoost: 98.42% / 98.4% / 98.4% / 98.4%	Reports as weighted averages (Avg Prec/Rec/F1); includes Kappa (98.02%); Evaluates benign vs malicious on a different dataset, so it is not directly comparable to binary phishing vs legitimate in this study.
A high-accuracy phishing website detection method based on machine learning [37]	Phishing Websites Dataset (58,000 legit; 30,647 phish; 112 attributes = 111 features + 1 label phishing)	ML comparison + SMOTEENN balancing; drop constant features; scenario-based evaluation	99.2% / 99.1% / 99.4% / 99.2%	Same source dataset; different balancing or evaluation setup.

TABLE X
FINAL REPRODUCIBILITY SUMMARY

Component	Final Setting Used In This Study
Dataset	Mendeley Phishing Websites. Total of 58,645 rows with 112 columns consisting of 111 features and 1 binary target column (phishing vs. legitimate).
Cleaning Dataset	Drop 13 constant features. After cleaning: 98 features (92 continuous + 6 binary); see Table II for post-cleaning dataset characteristics.
Split	Stratified train-test split: 80/20; random_state = 42
Pre-processing	Sanitization: replace -1 → 0 (continuous features); log transform: log1p on continuous; scaling: StandardScaler on continuous; binary features passed through unchanged
Hyperparameter Tuning	GridSearchCV; inner cross-validation: StratifiedKFold (n_splits=10, shuffle=True, random_state=42); selection metric: mean ROC-AUC
Selected XGBoost hyperparameters (from GridSearchCV)	learning_rate=0.08, max_depth=8, n_estimators=800, colsample_bytree=0.7, subsample=0.9, min_child_weight=1, gamma=0.0, reg_lambda=0.5, eval_metric=aucpr, tree_method=hist, random_state=42, objective=binary:logistic, n_jobs= -1

DOI : https://doi.org/10.25139/inform.v11i1.11221

Component	Final Setting Used In This Study
Early Stopping (effective boosting rounds)	Applied after GridSearchCV selection: n_estimators_max=5000, early_stopping_rounds=100, eval_metric=aucpr, eval_set=20% stratified split from training (test set not used), best_iteration=950 (effective n_estimators=950, final refit on full training with early stopping disabled)
Calibration	probability calibration: isotonic regression (CalibratedClassifierCV(method="isotonic", cv=3)) on the training set, final calibrated model fitted on full training data
Threshold Optimization	Threshold search: grid 0.20-0.80 (121 points, step=0.005), objective=maximize F1 on training OOF (Out-Of-Fold) probabilities (3-fold CV via cross_val_predict on calibrated model, test not used for tuning), tuned threshold t*=0.45, test evaluation reported at 0.50 and t*
Hardware & Software	See Experimental Setup section for complete specifications

As shown in Table VIII, the proposed pipeline (98 selected features after constant-feature removal and systematic GridSearchCV in this work) achieves slightly higher performance than the XGBoost (GridSearchCV, 111 features) results reported in [18]. At the same operating threshold (0.50), the proposed model improves accuracy by 0.20%, precision by 0.14%, recall by 0.24%, and f1-score by 0.19%. Although the gains are modest, they indicate a more favourable balance between phishing detection and false-alarm control under the selected decision rule. In practical terms, on the 11,729-sample hold-out test set used in this study, a 0.20% increase in accuracy corresponds to approximately 24 additional correct predictions, along with fewer false positives and false negatives. From an operational perspective, the approximately 33 hours GridSearchCV cost can be treated as a one-time offline optimization step; once selected, the final configuration supports consistent deployment where stable performance and controlled false alarms are critical.

Table IX complements the comparison in Table VIII by summarizing representative phishing-detection studies with varying degrees of comparability. Several works report strong performance but rely on different datasets (e.g., PhishTank/UCI or UCI-only) or different problem settings (e.g., general malicious-website detection rather than binary phishing vs legitimate). In addition, one study uses the same source dataset but applies different balancing and evaluation procedures, potentially affecting the reported metrics. Therefore, Table IX is presented to contextualize the proposed study rather than to claim strict, apples-to-apples superiority over all prior work.

For convenience of replication, Table X consolidates all final settings used in this study, including the effective n_estimators after early stopping, the calibration configuration, and the tuned threshold.

IV. CONCLUSION

Phishing websites remain a persistent cybersecurity threat, deceiving users into disclosing sensitive information through counterfeit pages. Rather than proposing a new classifier, this study establishes a systematic, reproducible academic baseline for URL/website-feature-based (tabular) phishing detection using an XGBoost pipeline on the Mendeley Phishing Websites dataset (58,645 samples). Starting from 111 URL and website-based features, the pipeline removes 13 constant features (resulting in 98 active features). It performs rigorous data cleaning, ColumnTransformer-based pre-processing, and grid-based hyperparameter optimization using stratified cross-validation, followed by early stopping and decision-threshold selection. To reduce leakage risk, all pre-processing and model-selection steps are fitted on training data only, with evaluation performed on a hold-out test set.

On the hold-out test set (11,729 samples) at a 0.50 decision threshold, the optimised model achieved an accuracy of 96.34%, a precision of 96.31%, a recall of 96.70%, and an F1-Score of 96.51%, with a balanced error profile (false-positive rate 4.05% and false-negative rate 3.30%). Compared with a default XGBoost configuration, the proposed pipeline improves accuracy by 0.20%, precision by 0.14%, recall by 0.24%, and F1-Score by 0.19%, corresponding to approximately 24 additional correct predictions and 15 additional detected phishing URLs on the test set. Feature correlation analysis indicated that directory/path-related attributes (e.g., qty_slash_directory, directory_length, qty_slash_url) are among the features most strongly associated with the phishing label, which aligns with prior observations that attackers often employ complex URL path structures to increase deception.

Methodologically, the primary contributions include a fully reproducible evaluation protocol and an end-to-end experimental pipeline that integrates systematic feature cleaning, nested hyperparameter tuning, and operationally motivated threshold selection with confusion-matrix-based reporting. Although threshold tuning identified 0.45 as optimal on internal validation data, the default threshold of 0.50 is reported for comparability and achieved superior test performance, suggesting mild validation optimism and supporting the stability of the selected configuration without extensive threshold manipulation. However, this study has several limitations that should be acknowledged. First, the scope is limited to tabular URL/website feature-based detection and does not include visual content analysis, HTML/DOM parsing, or deep learning on raw text/images. Second, evaluation is conducted on a static dataset without temporal validation to assess performance degradation as phishing tactics evolve. Third, while the pipeline is systematically documented for reproducibility, the trained model and complete codebase are not publicly released, which may limit exact replication by other researchers.

Despite these limitations, implementing this pipeline in real-world applications presents additional challenges, including severe class imbalance in operational datasets, concept drift, and robustness against sophisticated evasion techniques. Future research will explore cost-sensitive learning, temporal validation for drift-aware evaluation, more efficient hyperparameter optimisation (e.g., Bayesian optimisation or

DOI : https://doi.org/10.25139/inform.v11i1.11221

evolutionary strategies), and the integration of content-based signals to improve resilience against visually deceptive attacks. Overall, these extensions can further strengthen the proposed pipeline to inform future deployment-oriented studies under realistic constraints.

REFERENCES

[1] Q. E. u. Haq, M. H. Faheem, and I. Ahmad, "Detecting phishing URLs based on a deep learning approach to prevent cyber-attacks," *Applied Sciences*, vol. 14, no. 22, pp. 10086, Nov. 2024, doi: 10.3390/app142210086.

[2] N. F Almujahid *et al*., "Comparative evaluation of machine learning algorithms for phishing website detection," *PeerJ Comput. Sci.*, vol. 10, e2131, 2024, doi: 10.7717/peerj-cs.2131.

[3] S. Khan, B. Khan, S. Jan, S. Ullah, and Aiman, "Empirical analysis of neural networks-based models for phishing website classification using diverse datasets," *Journal of Cyber Security*, vol. 5, no. 1, pp. 47–66, 2023, doi: 10.32604/jcs.2023.045579.

[4] K. Omari *et al., "Comparative study of machine learning algorithms for phishing website detection," International Journal of Advanced Computer Science and Applications*, vol. 14, no. 9, pp. 417–425, 2023, doi: 10.14569/IJACSA.2023.0140945.

[5] T. Srinivasa, P. K. Srivastava, and M. Sharma, "A comprehensive survey on phishing website detection techniques," *SGVU International Journal of Convergence of Technology and Management*, vol. 11, no. 2, pp. 52–59, 2025.

[6] M. R. T. Utami, M. H. Hilman, and S. Yazid, "Enhancing phishing detection: Integrating XGBoost with feature selection techniques," *SSRN* preprint, Jan. 2025, doi: 10.2139/ssrn.5087049.

[7] K. Omari *et al*., "Phishing detection using gradient boosting classifier," *ICECMSN*, vol. 230, pp. 120–127, 2023, doi: 10.1016/j.procs.2023.12.067.

[8] S. S. M. Aldaham, O. Ouda, and A. A. Abd El-Aziz, "Improved detection of phishing websites using machine learning," *International Journal of Intelligent Systems and Applications in Engineering*, vol. 12, no. 21s, pp. 4619–4633, 2024.

[9] Anti-Phishing Working Group (APWG), "Phishing activity trends report, 4th quarter 2023," Anti-Phishing Working Group, Feb. 2024.

[10] B. Toulas, "Hackers target FCC, crypto firms in advanced Okta phishing attacks," *BleepingComputer*, Mar. 2024

[11] M. Hernandez, "Ripple CTO warns of huge phishing surge as seed phrases become targets," *The Currency Analytics*, Oct. 2025.

[12] A. Fajar *et al*., "Enhancing phishing detection through feature importance and selection," *arXiv* preprint, 2024, arXiv:2411.06860, doi: 10.48550/arXiv.2411.06860.

[13] T. Nagunwa, "Comparative analysis of nature-inspired metaheuristic techniques for optimizing phishing website detection," *AI*, vol. 3, no. 3, pp. 344–367, 2024.

[14] M. Sokolova and G. Lapalme, "A systematic analysis of performance measures for classification tasks," *Information Processing & Management*, vol. 45, no. 4, pp. 427–437, 2009, doi: 10.1016/j.ipm.2009.03.002.

[15] D. M. W. Powers, "Evaluation: From precision, recall and F-measure to ROC, informedness, markedness & correlation," *Journal of Machine Learning Technologies*, vol. 2, no. 1, pp. 37–63, 2011.

[16] G. C. Cawley and N. L. C. Talbot, "On over-fitting in model selection and subsequent selection bias in performance evaluation," *Journal of Machine Learning Research*, vol. 11, pp. 2079–2107, 2010.

[17] S. Varma and R. Simon, "Bias in error estimation when using cross-validation for model selection," *BMC Bioinformatics*, vol. 7, no. 91, 2006.

[18] Lukito and W. B. T. Handaya, "Deteksi website phishing menggunakan teknik machine learning," *Jurnal Informatika Atma Jogja*, vol. 6, no. 1, pp. 69–80, May 2025, doi: 10.24002/jiaj.v6i1.11538.

[19] G. Vrbančič, I. Fister Jr., and V. Podgorelec, "Datasets for phishing websites detection," *Data in Brief*, vol. 33, art. 106438, 2020, doi: 10.1016/j.dib.2020.106438.

[20] J. Bergstra and Y. Bengio, "Random search for hyper-parameter optimization," *Journal of Machine Learning Research*, vol. 13, pp. 281–305, 2012.

[21] H. A. K. Afandi, M. L. F. Al-Dzaki, N. Qomariasih, and R. A. Wildana, "GuardSurfing: Ekstensi browser sebagai alat bantu deteksi website phishing dengan metode klasifikasi XGBoost untuk deteksi URL phishing berbasis Flask framework," *Info Kripto*, vol. 19, no. 2, pp. 73–85, 2025, doi: 10.56706/ik.v19i2.124.

[22] S. Sheikhi and P. Kostakos, "Safeguarding cyberspace: Enhancing malicious website detection with PSO-optimized XGBoost and firefly-based feature selection," *Computers & Security*, vol. 142, p. 103885, Jul. 2024, doi: 10.1016/j.cose.2024.103885.

[23] H. Ghalechyan *et al*., "Phishing URL detection with neural networks: An empirical study," *Scientific Reports*, vol. 14, no. 1, art. 25134, 2024, doi: 10.1038/s41598-024-74725-6.

[24] T. Chen and C. Guestrin, "XGBoost: A scalable tree boosting system," in *Proc. ACM SIGKDD Int. Conf. Knowledge Discovery and Data Mining*, 2016.

[25] I. Guyon and A. Elisseeff, "An introduction to variable and feature selection," *Journal of Machine Learning Research*, vol. 3, pp. 1157–1182, 2003.

[26] R. Kohavi, "A study of cross-validation and bootstrap for accuracy estimation and model selection," in *Proc. IJCAI*, 1995.

[27] S. Sathyanarayanan and B. R. Tantri, "Confusion matrix-based performance evaluation metrics," *African Journal of Biomedical Research*, vol. 27, no. 4S, pp. 4023–4031, Nov. 2024, doi: 10.53555/AJBR.v27i4S.4345.

[28] M. Grandini, E. Bagli, and G. Visani, "Metrics for multi-class classification: An overview," *arXiv Preprint*, arXiv:2008.05756, 2020.

[29] A. F. Tjahjono, H. Hasan, R. P. Putera, D. M. P. Indranto, and A. T. Hermawan, "Klasifikasi URL phishing untuk SIEM: Perbandingan model machine learning XGBoost dan deep learning TabNet dalam deteksi ancaman siber," *Sains Data: Jurnal Studi Matematika dan Teknologi*, vol. 3, no. 2, pp. 62–71, Jul. 2025, doi: 10.52620/sainsdata.v3i2.227.

[30] F. Pedregosa *et al*., "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[31] S. Kaufman, S. Rosset, C. Perlich, and O. Stitelman, "Leakage in data mining: Formulation, detection, and avoidance," *ACM Transactions on Knowledge Discovery from Data*, vol. 6, no. 4, art. 15, 2012, doi: 10.1145/2382577.2382579.

[32] C. Feng, H. Wang, and N. Lu, "Log transformation and its implications for data analysis," *Shanghai Archives of Psychiatry*, vol. 26, no. 2, pp. 105–109, 2014, doi: 10.3969/j.issn.1002-0829.2014.02.009.

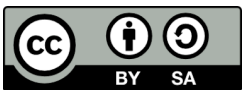
[33] T. Fawcett, "An introduction to ROC analysis," *Pattern Recognition Letters*, vol. 27, no. 8, pp. 861–874, 2006.

[34] J. Davis and M. Goadrich, "The relationship between precision-recall and ROC curves," in *Proc. ICML*, 2006.

[35] B. Zadrozny and C. Elkan, "Transforming classifier scores into accurate multiclass probability estimates," in *Proc. ACM SIGKDD Int. Conf. Knowledge Discovery and Data Mining*, 2002.

[36] A. Niculescu-Mizil and R. Caruana, "Predicting good probabilities with supervised learning," in *Proc. ICML*, 2005.

[37] M. Bahaghighat, M. Ghasemi, and F. Ozen, "A high-accuracy phishing website detection method based on machine learning," *Journal of Information Security and Applications*, vol. 77, p. 103553, Sep. 2023, doi: 10.1016/j.jisa.2023.103553



DOI : https://doi.org/10.25139/inform.v11i1.11221