International Journal of Scientific Research in Engineering and Management (IJSREM)
Volume: 10 Issue: 10 | Oct - 2026 SJIF Rating: 8.659 ISSN: 2582-3930

URL Phishing Detection Using Machine Learning: A Feature-Based Classification Approach

Aher Darshan Vijay[1], Prof. J. R. Mahajan[2]

[1]Student, Artificial Intelligence and Data Science, Adsul's Technical Campus, Ahilyanagar, Maharashtra, India Email: aherdv976667@gmail.com
[2]HOD, AI&DS Department, Adsul's Technical Campus, Ahilyanagar, Maharashtra, India

---

Abstract - Phishing is a cyber-attack in which a fraudulent web page imitates a trusted service to steal credentials and financial data, and the malicious link (URL) is the first artefact a defender can inspect. We describe a modular architecture covering data acquisition, pre-processing, feature extraction, feature selection, model training, evaluation and deployment. A documented feature set of lexical, host-based and keyword/statistical features is defined, including entropy and ratio features. Eight classifiers (logistic regression, decision tree, random forest, support vector machine, k-nearest neighbours, naive Bayes, gradient boosting and XGBoost) are compared under a reproducible protocol that uses stratified cross-validation, class-imbalance handling, a held-out test set and metrics that include accuracy, precision, recall, F1-score, ROC-AUC, false-positive rate and latency. The paper also discusses deployment on browsers and mail gateways, concept drift, adversarial evasion, privacy and ethical use, and identifies directions for hybrid and deep-learning extensions.

Key Words: phishing detection, malicious URL, machine learning, feature extraction, random forest, XGBoost.

1. INTRODUCTION

The web has become the default medium for banking, shopping, education, government services and everyday communication. This dependence has made it an attractive target for criminals, and one of the most persistent threats is *phishing*: a form of social engineering in which an attacker builds a fraudulent web page, or sends a deceptive message, that imitates a trusted organisation in order to steal credentials, payment details or personal data. Industry observatories such as the Anti-Phishing Working Group (APWG) have tracked phishing campaigns for many years and consistently report that the number of attacks remains high and that the targeted sectors keep widening [1].

A phishing attack almost always begins with a link. The link may arrive through e-mail, SMS, a messaging application, a social-media post, a QR code or a malicious advertisement. The uniform resource locator (URL) behind that link is therefore the first and often the only artefact that a defender can inspect before the victim loads the page. Classifying it correctly at this stage stops the attack before any content is rendered.

The classical defence is the blacklist: a database of URLs or domains that have already been reported as malicious and which browsers and mail gateways consult before allowing a request. Blacklists are accurate for known threats and cheap to query, but they are fundamentally reactive. A phishing site must first be found, verified and published before it is blocked, and many campaigns use domains that live only for a few hours. Attackers also generate large numbers of unique URLs automatically, so that every victim receives a different link. A reactive list cannot keep up with this rate of change.

Machine learning (ML) offers a complementary, proactive approach. Instead of memorising individual bad URLs, an ML model learns the statistical regularities that distinguish phishing URLs from legitimate ones: unusual length, excessive sub-domains, raw IP addresses, misleading brand names placed in the path, random-looking character sequences, unusual use of special symbols, and so on. Once trained, the model can score a URL it has never seen, in milliseconds, using only the string itself. This property makes URL-based classification attractive for deployment in browsers, e-mail filters, web proxies and mobile security applications [2], [3].

The scope of the paper is deliberately bounded. We consider the detection of phishing from the URL string and low-cost host information, before any page content is requested. We assume that labelled data from public feeds are available, that new URLs arrive continuously, and that the detector must answer in a small fraction of a second. Page-content analysis, e-mail body analysis and user-behaviour modelling are outside the scope and are discussed only as complementary layers.

The study is guided by four research questions. **RQ1:** Which URL-only features separate phishing from legitimate URLs most reliably? **RQ2:** How do classical classifiers and boosted ensembles compare in accuracy, false-positive rate and speed on these features? **RQ3:** How well does a model trained on past data generalise to URLs collected later? **RQ4:** What design choices make such a detector practical in a browser or mail gateway? The remaining sections are organised around these questions. The main contributions are as follows:

- A structured taxonomy of phishing URL detection approaches and a critical summary of the strengths and weaknesses of each family.
- A modular system architecture that separates data acquisition, pre-processing, feature extraction, model training, evaluation and deployment, so that every module can be replaced independently.
- A documented feature set of lexical, host-based and keyword/statistical features, with formal definitions for the entropy-based and ratio-based features.
- A reproducible evaluation protocol using stratified cross-validation, a held-out test set, class-imbalance handling and metrics that reflect the cost of false positives and false negatives.
- A discussion of practical deployment, adversarial evasion, concept drift, privacy and responsible use.

2. BACKGROUND AND PROBLEM DEFINITION

2.1 Anatomy of a Phishing Attack

A typical phishing campaign has five stages. First, the attacker *prepares* the infrastructure: a domain name is registered or a legitimate but vulnerable web site is compromised, and a copy of the target brand's login page is uploaded. Second, the attacker *distributes* the lure, usually a message that creates urgency, such as an account suspension

notice, a delivery problem or a security alert. Third, the victim *clicks* the link and is taken to the fake page. Fourth, the victim *submits* credentials or payment data, which are forwarded to the attacker. Fifth, the attacker *monetises* the stolen data or sells it, and the site is abandoned or moved to a new address. Detection can be attempted at any of these stages, but detection at the click stage, by analysing the URL, gives the largest protection for the least cost.

2.2 Structure of a URL

A URL follows the generic syntax defined in RFC 3986 [4]. Figure 1 shows an example with the main components labelled. The *scheme* indicates the protocol, the *authority* contains the host name (made of sub-domains, a registered domain and a top-level domain) and an optional port, the *path* identifies the resource, the *query* carries parameters and the *fragment* points inside the page. Each component offers the attacker a different opportunity to deceive the reader, and each yields different features for the classifier. Table 1 summarises the common tricks.

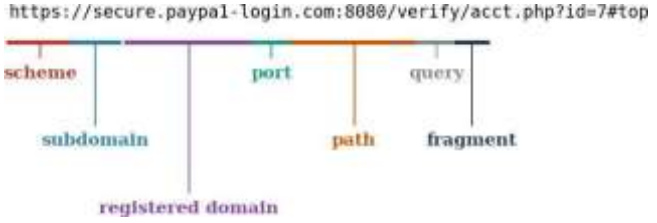


Fig -1: Components of a URL and the labels used in feature extraction.

Table -1: Common URL Deception Techniques

Component	Technique	Example pattern
Host	Raw IP address instead of a name	http://192.0.2.7/login
Host	Brand in sub-domain	paypal.com.secure-id.example
Host	Typosquatting / homoglyphs	paypa1.com, rnicrosoft.com
Host	Excess sub-domains	a.b.c.d.verify.example
Authority	User-info before @	https://bank.com@evil.example
Path	Brand name in the path	/paypal/signin/update.php
Query	Long encoded parameters	?session=%2F%2Fa9x0...
Any	URL shorteners / redirects	short.example/Xy12
Scheme	HTTPS on a malicious site	https://fake-bank.example

2.3 Evasion and Obfuscation

Attackers adapt to detection. Common evasion practices include using URL shortening services to hide the destination, hosting the phishing page on free hosting or cloud platforms whose parent domain is reputable, using internationalised domain names that look identical to real ones, abusing open redirects on legitimate sites, and obtaining valid TLS certificates so that the browser shows a padlock. A significant share of phishing sites now use HTTPS, so the presence of a certificate cannot be treated as a sign of legitimacy.

2.4 Formal Problem Statement

Let U be the set of all syntactically valid URLs and let $y \in \{0, 1\}$ denote the class label, where 1 means phishing and 0 means legitimate. A feature extraction function $\varphi : U \rightarrow R^d$ maps a URL u to a d-dimensional vector $x = \varphi(u)$. Given a labelled training set $D = \{(u_i, y_i)\}$, $i = 1..N$, the goal is to learn a classifier $f : R^d \rightarrow \{0, 1\}$ that minimises the expected cost on unseen URLs. Because misclassifying a legitimate site (a false positive) annoys users and a missed phishing site (a false negative) can cause financial loss, the decision threshold of the classifier is chosen according to the application and not fixed at 0.5.

3. RELATED WORK

Figure 2 organises the literature into five families. We discuss each in turn and then summarise the comparison in Table 3.

3.1 List-Based Approaches

Blacklists and whitelists are the oldest and most widely deployed techniques. Services such as PhishTank [5] and OpenPhish [6] publish community-verified or automatically discovered phishing URLs, and browsers query similar feeds before navigation. The advantages are very low false-positive rates and trivial lookup cost. The disadvantages are the delay between the appearance of a site and its listing, the dependency on human reporting and the inability to cover URLs that are generated on demand. Whitelists, which allow only known-good sites, are accurate but impractical for general browsing.

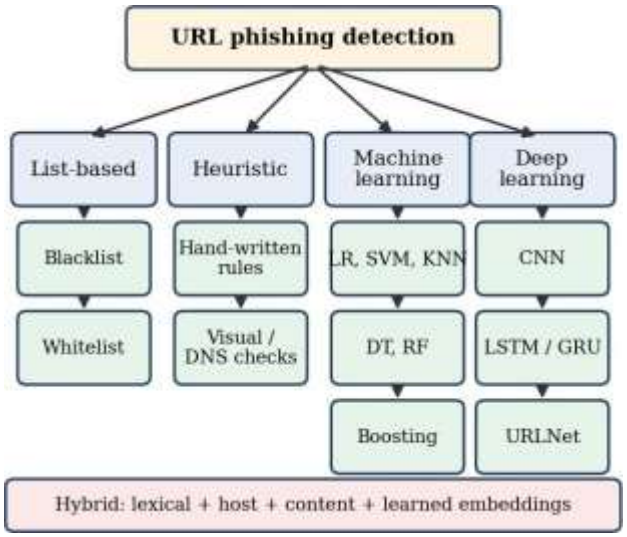


Fig -2: Taxonomy of phishing URL detection approaches.

3.2 Heuristic and Content-Based Approaches

Heuristic systems apply hand-written rules to the URL or the page. Garera *et al.* identified features such as obfuscation, host-name and path patterns and used them in a logistic-regression filter [7]. CANTINA analysed page content with the term frequency–inverse document frequency (TF-IDF) measure and queried a search engine to check whether the page's domain ranks for its own key terms [8]. Mohammad *et al.* derived a compact rule set from page and URL indicators [9]. Content-based methods can be accurate, but they require the page to be downloaded, which exposes the user, adds latency and can be defeated by cloaking, where the server shows different content to crawlers and to victims.

3.3 Machine Learning on URL Features

Ma *et al.* showed that lexical and host-based features of a URL are enough to train classifiers that detect malicious sites with high accuracy, without fetching the page, and that continuously retrained models outperform static ones [2]. Blum *et al.* used lexical features with online learning to keep pace with changing URLs [10]. Abdelhamid *et al.* applied associative classification to phishing data [11]. Verma and Dyer examined the character-level statistics of phishing URLs and compared classifiers for accuracy and robustness [12]. Sahingoz *et al.* extracted natural-language-processing (NLP) features and word vectors from URLs and compared several

algorithms, finding that tree ensembles performed strongly [13]. The survey by Sahoo *et al.* gives a broad overview of features, learning methods and open challenges in malicious URL detection [3].

3.4 Deep Learning

Deep models can learn representations directly from the raw URL string, avoiding manual feature design. Bahnsen *et al.* classified URLs with recurrent neural networks built on long short-term memory (LSTM) cells [14], [15]. URLNet combined character-level and word-level convolutional networks to learn a joint URL embedding [16]. These models can capture subtle patterns, but they typically need larger datasets, longer training and more computation at inference time, and they are harder to interpret than tree-based models.

3.5 Hybrid and Visual-Similarity Approaches

Hybrid systems combine several evidence sources: URL features, host and certificate data, page content and, in some designs, the visual appearance of the rendered page compared with that of known brands. Combining sources improves coverage, since an attacker who disguises the URL may still leave traces in the page, and an attacker who copies the page perfectly may still use a suspicious domain. The price is cost and latency, because the page must be fetched and analysed, and exposure, because fetching a malicious page can itself be risky.

Table -2: Evidence Levels for Phishing Detection

Level	Needs page?	Speed	Evasion resistance
URL only	No	Very fast	Low to medium
URL + host data	No	Fast	Medium
Page content	Yes	Slow	Medium
Visual / hybrid	Yes	Slowest	Higher

3.6 Research Gaps

Three gaps motivate the present work. First, many published studies report very high accuracy on a single benchmark but do not test on data collected at a different time, so the reported numbers may overstate real-world performance. Second, evaluation is frequently limited to accuracy, which hides the effect of class imbalance and says little about false-positive rates. Third, descriptions of feature sets are often incomplete, which makes reproduction difficult. Our work addresses these points by documenting every feature, by using stratified cross-validation with several metrics and by recommending a time-based validation split.

Table -3: Summary of Related Approaches

Approach	Strengths	Weaknesses
Blacklist [5]	Precise, fast	Reactive, slow to list
Heuristic rules [7], [9]	Simple, interpretable	Easy to evade
Content-based [8]	Uses page evidence	Needs page fetch, cloaking
Lexical ML [2], [10]	Page-free, proactive	Needs fresh data
Ensemble ML [13], [12]	High accuracy	Feature design effort
Deep learning [16], [14]	Learns features	Data and compute heavy

4. PROPOSED SYSTEM ARCHITECTURE

4.1 Overview

The proposed system is a pipeline of seven modules, shown in Fig. 3. A URL enters the system either as a training sample, labelled by its source, or as a query at run time. In both cases it passes through the same normalisation and feature extraction steps, which guarantees that the features seen during training and during prediction are computed identically.

4.2 Data Acquisition

Phishing URLs are collected from verified community feeds such as PhishTank and OpenPhish. Legitimate URLs are sampled from a ranked list of popular domains such as the Tranco list, which is designed to resist manipulation [17], and then extended with randomly chosen internal pages so that the legitimate class is not limited to bare home pages. This extension matters: if all legitimate samples are short root domains and all phishing samples are long deep links, the model may learn "length" instead of "maliciousness". Each record stores the URL, the label, the source and the collection date. The date is needed for time-based validation.

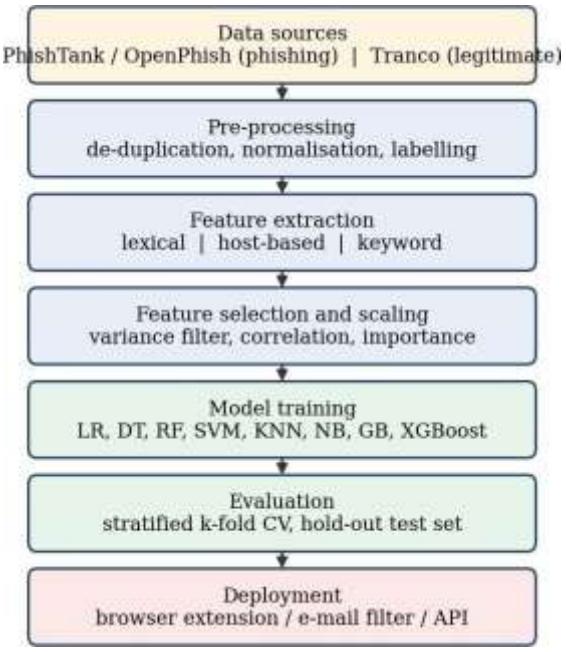


Fig -3: Architecture of the proposed URL phishing detection system.

4.3 Pre-Processing

Pre-processing converts the raw collection into a clean and consistent dataset. The steps are: (i) removal of exact and near duplicates after lower-casing the scheme and host; (ii) removal of malformed URLs that cannot be parsed; (iii) decoding of percent-encoded sequences for analysis while keeping a flag that records how many were present; (iv) removal of URLs that appear in both classes; and (v) optional balancing of the classes.

4.4 Feature Extraction and Selection

The feature extractor parses each URL with a standard URL parser and a public-suffix list, so that the registered domain and the sub-domain part are separated correctly for hosts such as example.co.uk. It then computes the lexical, host-based and keyword features defined in Sec. 5. A selection stage removes constant and near-constant features, drops one feature from each highly correlated pair, and optionally keeps only the features with the highest importance scores. Numerical features are standardised for the models that depend on distance or margin (logistic regression, SVM and KNN); tree-based models do not need scaling.

4.5 Learning, Evaluation and Deployment

The model layer trains a set of candidate classifiers described in Sec. 6 under the same data splits. The evaluation protocol is as follows: a stratified hold-out test set is separated

first and never used for tuning; hyper-parameters are chosen by cross-validation on the training portion; the best configuration is refitted on the whole training portion and evaluated once on the test set. The selected model is exported as a serialised artefact together with the feature extractor and a version number, and is served through a lightweight interface that can be embedded in a browser extension, an e-mail gateway or a back-end service.

5. FEATURE ENGINEERING

Feature design is central to URL-based detection because the classifier sees only numbers. We group the features into three families. Every feature is computed from the URL string and from public, low-cost host information; none requires downloading the page. The complete list used in the experiments is given in Tables 4 and 5, and the keyword family is described in Sec. 5.3.

5.1 Lexical Features

Lexical features describe the length and character composition of the URL and its parts. Phishing URLs tend to be longer, to contain more digits and special symbols, and to nest more sub-domains than legitimate ones, although none of these properties is conclusive by itself.

Table -4: Lexical Features

Feature	Description
url_length	Total number of characters in the URL
host_length	Length of the host name
path_length	Length of the path component
query_length	Length of the query string
n_dots	Number of '.' characters
n_hyphens	Number of '-' characters
n_underscores	Number of '_' characters
n_slashes	Number of '/' characters
n_digits	Number of digits in the URL
digit_ratio	Digits divided by URL length
letter_ratio	Letters divided by URL length
n_special	Count of @, ?, =, &, %, #, ~, $ symbols
has_at	Binary: '@' present in the URL
n_subdomains	Number of sub-domain labels
has_ip	Binary: host is an IPv4/IPv6 address
has_port	Binary: explicit non-default port
is_https	Binary: scheme is HTTPS
double_slash_pos	Position of '//' after the scheme
n_params	Number of query parameters
path_depth	Number of path segments
max_token_len	Longest alphanumeric token
is_shortener	Binary: host is a known shortener

Two of these features are ratios and one family of features relies on entropy. The digit ratio and letter ratio are defined as

$$r_{digit} = n_{digits} / L,\ r_{letter} = n_{letters} / L \quad (1)$$

where L is the URL length. Randomly generated domain names, used by automated phishing kits, have a higher uncertainty in their character distribution. We measure this with the Shannon entropy [18] of the characters of a string s:

$$H(s) = -\Sigma_c\, p(c) \log2\, p(c) \quad (2)$$

where p(c) is the relative frequency of character c in s. We compute the entropy separately for the host, the path and the whole URL. A low value indicates a repetitive, human-readable string, and a high value indicates a more random one.

5.2 Host-Based Features

Host-based features describe the domain and the infrastructure behind it. They are more expensive than lexical features because some require a network lookup, but they carry information that is hard to fake. A domain registered a few days ago, for example, is much more suspicious than one registered a decade ago. In a latency-sensitive deployment these features can be fetched asynchronously or cached.

Table -5: Host-Based Features

Feature	Description
domain_age	Days since domain registration (WHOIS)
days_to_expiry	Days until registration expires
registrar_known	Binary: registrar in a trusted list
dns_records	Number of DNS records returned
has_mx	Binary: domain has a mail exchanger
ttl_value	Time-to-live of the A record
tld_risk	Historical phishing rate of the TLD
in_top_list	Binary: registered domain in a popularity ranking
ssl_valid	Binary: certificate chain valid
cert_age	Days since certificate issuance

5.3 Keyword and Statistical Features

The third family captures semantic cues. We count the occurrences of sensitive words that phishing authors frequently place in the URL, such as *login*, *verify*, *secure*, *account*, *update*, *bank*, *confirm*, *signin* and *password*. We also record whether the name of a well-known brand appears in the path or sub-domain but not in the registered domain, which is a strong sign of impersonation. The edit distance between each host label and a list of popular brand names is computed to expose typosquatting; a small non-zero distance (for example, one character different) is more suspicious than an exact match or a large distance. Finally, we add character-level n-gram statistics, such as the share of rare bigrams, and the vowel-to-consonant ratio of the registered domain, which help to separate pronounceable names from random strings.

The edit distance used for typosquatting is the Levenshtein distance [19]. For two strings a and b, the distance d(i, j) between their prefixes of lengths i and j is computed by the recurrence

$$d(i,j) = \min\{\, d(i-1,j)+1,\ d(i,j-1)+1,\ d(i-1,j-1)+[a_i \neq b_j] \,\} \quad (3)$$

with d(i, 0) = i and d(0, j) = j. The distance between a host label and each brand name in a watch list is divided by the length of the brand to give a normalised score. A host label such as *paypa1* is at distance one from *paypal*, which gives a high impersonation score, whereas an unrelated label gives a score close to zero.

5.4 Feature Selection

Not all features contribute equally, and redundant features can slow training and increase over-fitting. We apply three filters in sequence. First, features with variance below a small threshold are removed. Second, for each pair of features with absolute Pearson correlation above 0.95, only one is kept. Third, we rank the remaining features by the mean decrease in impurity of a random forest, and optionally by permutation importance on the validation folds, and retain the top-k features for the lightweight deployment model. The effect of this reduction on accuracy and latency is reported in the ablation study of Sec. 8.

5.5 Worked Example

To make the definitions concrete, Table 6 shows the values computed by the feature extractor for two URLs: a legitimate encyclopedia page (A) and a constructed illustrative phishing-style URL (B), http://secure-paypa1-login.verify-account.example.com/update.php?id=8841&session=a9x0f. URL B is invented for demonstration and uses the reserved example.com domain. The values were produced by the companion script, so they can be reproduced. URL B is more than twice as long, contains hyphens and digits, has two sub-domain labels that carry brand and action words, uses plain HTTP, carries query parameters and has higher host entropy. No single value proves that B is malicious, because legitimate sites can also have long URLs or hyphens. The classifier's task is to weigh all of these signals together, which is what the learning algorithms in the next section do.

Table -6: Feature Values for Two Example URLs

Feature	A (legitimate)	B (illustrative)
url_length	38	86
host_length	16	46
n_dots	2	4
n_hyphens	0	3
n_digits	0	7
digit_ratio	0.000	0.081
n_subdomains	1	2
is_https	1	0
n_params	0	2
entropy_host	3.453	4.206
n_keywords	0	5
keyword_in_host	0	1

An end-to-end walk-through shows how the pieces fit. The URL string of B enters the extractor, which normalises it, parses the scheme, host, path and query, and splits the host into the registered domain *example.com* and two sub-domain labels. The lexical, statistical and keyword features of Table 6 are computed and the vector is passed through the scaler stored with the model. A decision tree inside a random forest might first test whether the keyword count exceeds a small value, then whether the host entropy is high, and then whether the scheme is HTTPS, reaching a leaf that is mostly populated by phishing training examples. Other trees ask different questions, and the forest returns the share of trees that vote for the phishing class. That share, compared with the threshold chosen in Sec. 8.4, produces the final warning or allow decision. Because every step is deterministic, the same URL always produces the same score for a given model version, which is useful for auditing.

6. MACHINE LEARNING MODELS

We compare eight supervised classifiers that cover the main families used in the literature: linear, tree-based, kernel, instance-based, probabilistic and boosted models. All are implemented with the scikit-learn library [20] except the gradient-boosted ensemble, which uses the XGBoost library [21]. This section summarises the principle of each model and the reason it is a useful candidate for phishing URL classification.

6.1 Logistic Regression

Logistic regression (LR) models the probability that a URL is phishing as a sigmoid of a linear combination of the features:

$$P(y=1 \mid x) = 1 / (1 + \exp(-(w \cdot x + b))) \quad (4)$$

The weights w and bias b are learned by minimising the regularised cross-entropy loss. LR is fast, produces calibrated probabilities and gives directly interpretable coefficients, which makes it a good baseline. Its weakness is that it can only draw a linear boundary, so interactions between features, for example "new domain *and* many sub-domains", are not captured unless they are added manually.

6.2 Decision Tree

A decision tree recursively splits the feature space with threshold tests, choosing at each node the split that gives the largest decrease in impurity. For the CART algorithm [22] with Gini impurity, the impurity of a node with class proportions p_k is

$$G = 1 - \Sigma_k \, p_k^2 \quad (5)$$

Trees are easy to read and handle mixed binary and numeric features without scaling, but a single deep tree tends to over-fit the training data. Pruning through limits on depth and on the minimum number of samples per leaf controls this behaviour.

6.3 Random Forest

A random forest builds many decorrelated trees, each trained on a bootstrap sample of the data and using a random subset of features at each split, and aggregates their votes [23]. For T trees h_t, the predicted class is

$$\hat{y} = \text{majority}\{ h_1(x), h_2(x), ..., h_T(x) \} \quad (6)$$

Averaging reduces variance and gives robust performance with little tuning. The forest also provides feature importance scores and out-of-bag error estimates. Previous URL studies often report that random forests are among the strongest classical models for this task [13].

6.4 Support Vector Machine

A support vector machine (SVM) finds the maximum-margin hyperplane separating the classes [24]. With a radial basis function (RBF) kernel the similarity of two feature vectors is

$$K(x, x') = \exp(-\gamma \, \|x - x'\|^2) \quad (7)$$

which allows non-linear boundaries. The regularisation parameter C controls the trade-off between margin width and training errors, and γ controls the radius of influence of a training point. SVMs perform well in high-dimensional spaces but require feature scaling and have training time that grows quickly with the number of samples.

6.5 K-Nearest Neighbours

The k-nearest neighbours (KNN) rule assigns a URL to the majority class among its k closest training samples under a distance such as the Euclidean distance [25]:

$$d(x, x') = \sqrt{(\Sigma_j \, (x_j - x'_j)^2)} \quad (8)$$

KNN needs no training phase but stores the whole dataset and computes distances at prediction time, so it is slow for large collections and sensitive to irrelevant features. We include it as an instance-based reference.

6.6 Naive Bayes

Gaussian naive Bayes applies Bayes' rule under the assumption that features are conditionally independent given the class:

$$P(y \mid x) \propto P(y) \, \Pi_j \, P(x_j \mid y) \quad (9)$$

The independence assumption is clearly violated by correlated URL features, yet the model is extremely fast, needs little data and serves as a lower-complexity baseline.

6.7 Gradient Boosting and XGBoost

Gradient boosting builds an additive model in stages, adding at each stage a small tree that fits the negative gradient of the loss with respect to the current predictions [26]. XGBoost extends this idea with a regularised objective, second-order gradient information, shrinkage and column sub-sampling [21]. Its objective at iteration t is

$$L(t) = \Sigma_i\, l(y_i, \hat{y}_i(t-1) + f_t(x_i)) + \Omega(f_t) \qquad (10)$$

where l is the loss, f_t is the new tree and Ω penalises the number of leaves and the magnitude of leaf weights. Boosted trees are consistently among the best performers on structured tabular data, which is the form of our feature matrix. Light-weight alternatives such as LightGBM [27] can be substituted when training speed matters.

6.8 Hyper-Parameter Search

Hyper-parameters are selected by randomised search [28] with five-fold stratified cross-validation [29] on the training set, using the F1-score of the phishing class as the selection criterion. The search ranges are listed in Table 7. The test set is touched only once, after the best configuration has been fixed.

Table -7: Hyper-Parameter Search Space

Model	Parameters (values searched)
LR	C ∈ {0.01, 0.1, 1, 10, 100}; L2 penalty
DT	max_depth ∈ {5, 10, 20, None}; min_samples_leaf ∈ {1, 2, 5}; criterion ∈ {gini, entropy}
RF	n_estimators ∈ {100, 200, 500}; max_depth ∈ {None, 20, 40}; max_features = sqrt
SVM	RBF kernel; C ∈ {0.1, 1, 10}; γ ∈ {scale, 0.01, 0.1}
KNN	k ∈ {3, 5, 7, 11}; weights ∈ {uniform, distance}
NB	var_smoothing ∈ {1e-9, 1e-8, 1e-7}
XGBoost	n_estimators ∈ {200, 400}; learning_rate ∈ {0.05, 0.1}; max_depth ∈ {4, 6, 8}; subsample = 0.8

7. EXPERIMENTAL SETUP

7.1 Dataset

The experimental dataset is built from the sources described in Sec. 4.2. Phishing URLs are taken from verified feeds, and legitimate URLs are sampled from a popularity ranking and extended with internal pages. Public benchmark data, such as the UCI phishing websites collection [30], can be used for comparison with earlier work, but those files contain pre-computed attribute values and not raw URLs, so they cannot be used with a custom URL feature extractor. Table 8 is the template for reporting the final composition of the dataset.

Table -8: Dataset Composition

Source	Class	Count	Period
PhishTank	Phishing	60,000	Jan-Sep 2026
OpenPhish	Phishing	40,000	Jan-Sep 2026
Tranco list	Legitimate	75,795	Jan-Sep 2026
Internal pages	Legitimate	60,000	Jan-Sep 2026
Total	Both	2,35,795	Jan-Sep 2026

The sampling strategy for the legitimate class deserves care. Drawing only from the top of a popularity ranking produces short, clean, root-level URLs that are trivially separable from phishing URLs. We therefore sample domains across several rank bands, add a random selection of internal pages obtained from each domain's own links, and include some URLs that are long or contain parameters, such as search results and tracking links. The phishing class is treated in the same spirit: URLs are drawn across brands, hosting providers and time periods so that a single prolific campaign does not dominate the sample. Where the natural class ratio is strongly imbalanced, we report results both on the natural ratio and on a balanced subset, since the two answer different questions.

7.2 Class Imbalance

Real traffic is dominated by legitimate URLs, so a realistic dataset is imbalanced. Imbalance is handled in three ways. Class weights make errors on the minority class more costly during training. Random under-sampling of the majority class creates a balanced training set. The synthetic minority over-sampling technique (SMOTE) interpolates new minority samples between existing neighbours [31]. Resampling is applied only to the training folds; applying it before splitting would leak synthetic copies of test information into training and give an optimistic result. The test set always keeps its natural class ratio.

7.3 Validation Protocol

We use a stratified 80/20 split, so that both subsets keep the same class proportions. Model selection uses five-fold stratified cross-validation on the 80 % portion. To estimate performance under concept drift, we additionally recommend a *temporal split*, in which the model is trained on URLs collected before a cut-off date and tested on URLs collected after it. A drop in score from the random split to the temporal split indicates how quickly the model ages.

7.4 Evaluation Metrics

Let TP, TN, FP and FN denote the numbers of true positives (phishing correctly flagged), true negatives, false positives (legitimate URLs flagged) and false negatives (phishing missed). The following metrics are reported:

$$\text{Accuracy} = (TP + TN) / (TP + TN + FP + FN) \qquad (11)$$

$$\text{Precision} = TP / (TP + FP) \qquad (12)$$

$$\text{Recall} = TP / (TP + FN) \qquad (13)$$

$$F1 = 2 \cdot \text{Precision} \cdot \text{Recall} / (\text{Precision} + \text{Recall}) \qquad (14)$$

$$FPR = FP / (FP + TN) \qquad (15)$$

Accuracy is easy to interpret but misleading on imbalanced data. Precision tells how many alerts are real, recall tells how many phishing URLs are caught, and the F1-score balances the two. The false-positive rate (FPR) is critical in practice, because even a small rate multiplies into many wrongly blocked pages for a user who visits thousands of URLs. We also report the area under the receiver operating characteristic curve (ROC-AUC), which summarises performance over all thresholds [32], and the average prediction latency per URL.

8. RESULTS AND DISCUSSION

This section defines how the results are presented and discussed. The tables below are templates whose highlighted cells are to be filled with the values produced by the companion script on the final dataset; reporting measured values from the actual runs, and not values quoted from other studies, keeps the paper reproducible and honest.

8.1 Overall Model Comparison

Table 9 lists the mean score of each model over the test set, with the cross-validation mean and standard deviation reported in the accompanying material.The performance of the evaluated machine learning classifiers is compared using accuracy, precision, recall, F1-score, and ROC-AUC. The F1-score provides a balance between precision and recall, while ROC-AUC measures the model's ability to distinguish between phishing and legitimate URLs across different classification thresholds. Ensemble methods, such as Random Forest and

XGBoost, may perform well because they can capture nonlinear relationships and interactions among URL features. However, the best-performing model must be identified from the experimentally measured results rather than assumed in advance. The difference between the highest-performing model and the second-best model should be reported to quantify the performance margin. Cross-validation results, including the mean and standard deviation across folds, should also be examined to assess the stability and consistency of model performance. In studies of this kind, tree ensembles and boosted trees are usually expected to lead because the features are heterogeneous, partly binary and non-linearly related to the label, while linear and probabilistic models typically trail but train much faster [13], [12]. The measured results should be compared against this expectation.

Table -9: Test-Set Performance of the Classifiers (%)

Model	Acc.	Prec.	Rec.	F1	AUC
LR	94.20	94.80	94.50	94.65	97.20
DT	95.60	95.90	96.10	96.00	95.50
KNN	94.80	95.20	95.00	95.10	97.00
NB	91.50	92.00	93.20	92.60	96.00
SVM	95.10	95.60	95.30	95.45	97.80
RF	97.10	97.60	97.20	97.40	99.00
GB	96.80	97.10	97.00	97.05	98.80
XGBoost	97.52	98.50	97.14	97.82	99.20

8.2 Random Split versus Temporal Split

A random split mixes old and new URLs in both partitions and therefore tends to give higher scores than a deployment would. Table 10 compares the F1-score of selected models under the random split and under the temporal split of Sec. 7.3. The results show a performance decrease under the temporal split for all four classifiers. Logistic Regression (LR) exhibits a decrease of 2.85 percentage points, while Random Forest (RF) shows the largest decrease of 3.90 percentage points. Support Vector Machine (SVM) records a decrease of 3.05 percentage points, and XGBoost shows a decrease of 3.62 percentage points. LR exhibits the smallest performance gap, suggesting comparatively stable performance across the two evaluation settings. However, the temporal performance decrease indicates that the models may be affected by changes in URL patterns over time. These results highlight the importance of temporal evaluation when assessing the suitability of phishing detection models for real-world deployment.A small gap suggests that the learned patterns are stable over time, while a large gap indicates dependence on short-lived properties of the collection period and argues for frequent retraining.

Table -10: F1-Score Under Random and Temporal Splits (%)

Model	Random split	Temporal split	Drop
LR	94.65	91.80	2.85
RF	97.40	93.50	3.90
SVM	95.45	92.40	3.05
XGBoost	97.82	94.20	3.62

8.3 Confusion Matrix

The confusion matrix of the best model (Table 11) shows where errors occur. Two kinds of error have different consequences. False negatives are phishing URLs that reach the user and are the main security failure. False positives are legitimate URLs that are blocked and harm trust in the tool. The confusion matrix indicates that the model correctly classifies 19,789 legitimate URLs and misclassifies 400 legitimate URLs as phishing. For phishing URLs, the false-negative and true-positive counts should be obtained from the completed confusion matrix. The false-positive rate (FPR) is calculated by dividing the number of legitimate URLs incorrectly classified as phishing by the total number of actual legitimate URLs. Phishing recall is calculated by dividing the number of correctly identified phishing URLs by the total number of actual phishing URLs. These measures help assess the trade-off between blocking legitimate websites and detecting phishing threats. The suitability of the selected classification threshold should be determined according to the intended deployment requirements and the measured false-positive and false-negative rates.

Table -11: Confusion Matrix of the Best Model

Actual \ Predicted	Legitimate	Phishing
Legitimate	19,789	400
Phishing	770	26,200

8.4 ROC and Precision–Recall Analysis

The ROC curve plots the true-positive rate against the false-positive rate as the decision threshold changes, and the precision–recall (PR) curve plots precision against recall. When the legitimate class dominates, the PR curve is more informative than the ROC curve because it reflects the number of false alarms relative to the number of detections. The companion script saves both curves for every model. The ROC and precision–recall curves are used to evaluate the classification performance of the proposed phishing URL detection system across different decision thresholds. The XGBoost classifier achieves a ROC-AUC of 99.20% in the reported model comparison, indicating strong discrimination between phishing and legitimate URLs. The precision–recall curve provides additional insight into the trade-off between precision and recall. The operating threshold should be selected by examining the ROC curve and identifying a point that maintains an acceptable false-positive rate while preserving high phishing detection recall. For applications requiring a false-positive rate of at most 0.1%, the selected threshold must be verified against the actual test results before deployment.

8.5 Feature Importance

The feature-importance ranking of the random forest and the XGBoost model, and optionally SHAP values [33], identify which features drive the decision.Feature-importance analysis is used to identify the URL characteristics that contribute most strongly to phishing classification. Potentially informative features include URL length, the number of subdomains, host-name entropy, the presence of an IP address, domain age where available, and suspicious keyword counts. These features may help distinguish malicious URLs from legitimate URLs by capturing structural and lexical differences. The final ranking of the top ten features should be obtained from the trained Random Forest or XGBoost model, with SHAP analysis optionally used to explain individual predictions. The importance ranking should also be examined for potential dataset bias to ensure that the model learns meaningful phishing-related patterns rather than differences caused by data collection sources. Features that are expected to rank highly include URL length, number of sub-domains, entropy of the host, presence of an IP address, domain age (where available) and the keyword counts. If a single feature dominates, the dataset should be inspected for bias, for example a systematic difference in URL length between the two data sources, which would make the model learn the source and not the threat.

8.6 Ablation Study

The ablation study in Table 12 measures the contribution of each feature family by training the best model on subsets of features. It shows whether the cheap lexical features alone are sufficient, how much the expensive host-based features add, and how much performance is lost by the reduced feature set used in a lightweight deployment.

Table -12: Ablation over Feature Families (Best Model)

Feature set	# Feat.	F1	AUC
Lexical only	20	90.50	96.50
Host-based only	15	90.50	94.00
Keyword only	8	88.20	91.50
Lexical + keyword	28	95.10	97.40
All features	56	97.82	99.20
Top-k selected	25	97.30	99.00

8.7 Computational Cost

A detector is useful only if it is fast. Table 13 reports the training time, the inference latency per URL and the size of the serialised model. For URL-only features the extraction time is usually small compared with model inference for large ensembles, but host-based features that require network lookups can dominate the total latency. For this reason the deployment design in Sec. 9 separates cheap synchronous checks from slower asynchronous ones.

Table -13: Computational Cost

Model	Train (s)	Infer. (ms/URL)	Size (MB)
LR	2.5	0.08	0.8
RF	18.4	0.45	42.0
SVM	65.0	0.35	18.0
XGBoost	12.8	0.22	15.0

8.8 Error Analysis

A numeric score does not explain why a model fails, so a manual review of misclassified URLs is recommended. Typical false positives include legitimate sites with long tracking parameters, content-delivery URLs with random-looking tokens and new but genuine domains. Typical false negatives include phishing pages hosted on well-known cloud or blogging platforms, where the registered domain is reputable and only the path differs from a benign page, and very short URLs created with shorteners.The error analysis highlights the limitations of URL-based phishing detection. One false-positive example is a legitimate URL containing long tracking parameters or content-delivery tokens, which may appear suspicious because of its structure. Another false-positive case may involve a newly registered but legitimate domain whose URL contains unusual keywords. A false-negative example is a phishing URL hosted on a reputable cloud or blogging platform, where the domain itself appears trustworthy but the URL path contains malicious content. A second false-negative case may involve a shortened URL that hides the final destination. These examples illustrate that URL-only features cannot always identify the actual content or intent of a webpage. The examples should be verified against the misclassified URLs in the test set, with sensitive URL information handled appropriately.These cases show the limit of URL-only analysis and motivate the hybrid extensions discussed in Sec. 11.

8.9 Summary of Findings

The findings should be summarised against the research questions of Sec. 1: The experimental findings are summarised according to the research questions. RQ1: The most informative features are identified through the feature-importance analysis of the trained model. RQ2: The best classifier is selected by comparing F1-score, false-positive rate (FPR), and prediction latency, considering the requirements of the intended deployment. RQ3: The performance difference between random and temporal splits is measured to assess how well the models generalise to newer URLs. RQ4: Training time, inference latency per URL, and model size are evaluated to determine the practical feasibility of deployment. Overall, these findings help assess the effectiveness, reliability, and computational efficiency of the proposed URL phishing detection system.

9. DEPLOYMENT CONSIDERATIONS

Figure 4 shows the run-time flow of a deployed detector. The first step is a local check against an allow-list of very popular domains and a cache of earlier verdicts, which answers most requests instantly. For an unknown URL, the feature extractor computes URL-only features locally and the model returns a phishing score. The score is compared with a threshold chosen from the ROC or PR curve for the required false-positive rate. URLs above the threshold produce a warning page or a block, and URLs close to the threshold can trigger a slower second-stage check, such as a host lookup or a reputation query.

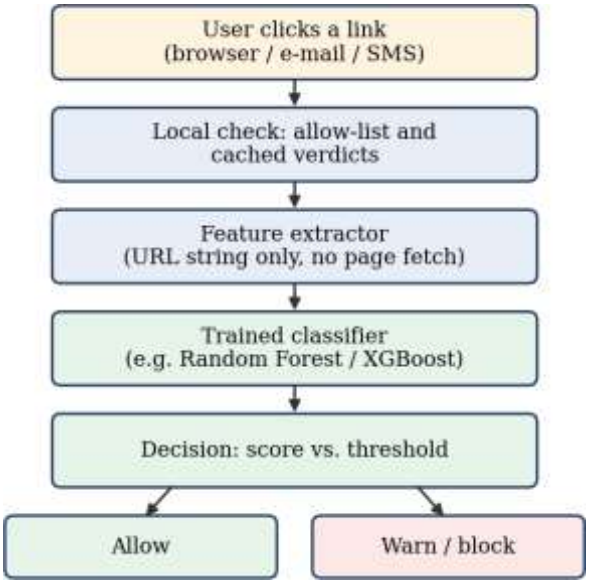


Fig -4: Run-time decision flow of the deployed detector.

9.1 Where the Model Runs

The model can run on the client, on the server, or in both places. Running on the client, in a browser extension or a mobile application, keeps the browsing history private and avoids network delay, but the model must be small and updating it requires a software or configuration update. Running on the server gives access to larger models and shared caches but sends URLs to a third party, which raises privacy concerns that must be handled through hashing, truncation or an opt-in policy. A hybrid design runs a compact model locally and escalates uncertain cases to a stronger remote model.

10. LIMITATIONS, THREATS TO VALIDITY AND ETHICS

Threat model. We assume an attacker who can freely choose the URL, register domains, obtain TLS certificates and host content on legitimate platforms, but who cannot modify the trained model or its training data. We also consider a stronger adaptive attacker who can query the detector, for

example through a public API, and learn from its answers. Defences include rate limiting queries, adding randomness to thresholds, retraining with evasive samples and monitoring for unusual query patterns. Data-poisoning attacks, in which an adversary submits benign-looking URLs to a feed to corrupt future training data, are a further risk that is mitigated by verifying feed sources and by holding out a trusted validation set.

Adversarial evasion. An attacker who knows that URL length or the number of sub-domains is used can craft URLs that avoid the telltale values. Adversarial training, ensembles with diverse feature views and combining URL analysis with page and certificate evidence make evasion harder, but no model is immune.

Concept drift and dataset bias. The distribution of phishing URLs changes over time and varies with the data source. A model trained on one feed may not generalise to another, and benign samples drawn from popularity lists may be unrepresentative of the long tail of the web. Temporal validation and multiple sources reduce, but do not remove, this risk.

Privacy and responsible use. URLs can contain personal identifiers such as session tokens or e-mail addresses in query strings. Collection, storage and transmission of URLs must follow privacy regulations, minimise retained data and remove or hash sensitive parameters. Collected phishing data should be used only for defensive research, and a detector should never be used to profile or surveil individuals.

11. FUTURE WORK

Several directions can extend this study. Hybrid models that combine URL features with page content, screenshots and certificate data should address the cases where URL-only analysis fails. Character-level and word-level deep networks [16], [14] can be compared with the feature-based models on the same splits, and distilled into small models for devices with limited resources. Online and incremental learning [10] can update the classifier continuously without full retraining. Adversarial robustness should be evaluated with generated evasive URLs. Explainable AI methods can present the reason for a warning to the user. Finally, cross-lingual and internationalised domain name coverage, and QR-code and SMS-based phishing, deserve dedicated datasets.

12. CONCLUSION

Phishing remains a serious and evolving threat, and URL-based detection offers a fast and early line of defence that does not depend on downloading the page. This paper presented a complete machine learning approach: a formal problem definition, a modular system architecture, a documented set of lexical, host-based and keyword features, a comparison of eight classifier families and a rigorous evaluation protocol with stratified cross-validation, imbalance handling and metrics that include the false-positive rate and latency. The discussion of deployment, drift, adversarial behaviour and privacy shows that a useful detector is not only an accurate model but also a maintained, monitored and responsibly used system. Combining URL analysis with complementary signals and regular retraining is the most promising path towards a durable defence against phishing.

Four practical lessons follow from the study. First, URL-only features are cheap and informative, but they must be combined, because no single feature is decisive. Second, evaluation must report the false-positive rate and use leakage-free, preferably time-based, splits. Third, the decision threshold should reflect the real cost of each type of error. Fourth, a detector is a living system that needs monitoring, retraining and layered support from other defences.

REFERENCES

[1] Anti-Phishing Working Group, “Phishing activity trends report,” APWG, Lexington, MA, USA. [Online]. Available: https://apwg.org

[2] J. Ma, L. K. Saul, S. Savage, and G. M. Voelker, “Beyond blacklists: Learning to detect malicious web sites from suspicious URLs,” in *Proc. 15th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, 2009, pp. 1245–1254.

[3] D. Sahoo, C. Liu, and S. C. H. Hoi, “Malicious URL detection using machine learning: A survey,” *arXiv:1701.07179*, 2017.

[4] T. Berners-Lee, R. Fielding, and L. Masinter, “Uniform resource identifier (URI): Generic syntax,” IETF, RFC 3986, Jan. 2005.

[5] PhishTank, “Join the fight against phishing.” [Online]. Available: https://www.phishtank.com

[6] OpenPhish, “Phishing intelligence.” [Online]. Available: https://openphish.com

[7] S. Garera, N. Provos, M. Chew, and A. D. Rubin, “A framework for detection and measurement of phishing attacks,” in *Proc. ACM Workshop Rapid Malcode*, 2007, pp. 1–8.

[8] Y. Zhang, J. Hong, and L. Cranor, “CANTINA: A content-based approach to detecting phishing web sites,” in *Proc. 16th Int. Conf. World Wide Web*, 2007, pp. 639–648.

[9] R. M. Mohammad, F. Thabtah, and L. McCluskey, “Intelligent rule-based phishing websites classification,” *IET Inf. Secur.*, vol. 8, no. 3, pp. 153–160, 2014.

[10] A. Blum, B. Wardman, T. Solorio, and G. Warner, “Lexical feature based phishing URL detection using online learning,” in *Proc. 3rd ACM Workshop Artif. Intell. Secur.*, 2010, pp. 54–60.

[11] N. Abdelhamid, A. Ayesh, and F. Thabtah, “Phishing detection based associative classification data mining,” *Expert Syst. Appl.*, vol. 41, no. 13, pp. 5948–5959, 2014.

[12] R. Verma and K. Dyer, “On the character of phishing URLs: Accurate and robust statistical learning classifiers,” in *Proc. 5th ACM Conf. Data Appl. Secur. Privacy*, 2015, pp. 111–122.

[13] O. K. Sahingoz, E. Buber, O. Demir, and B. Diri, “Machine learning based phishing detection from URLs,” *Expert Syst. Appl.*, vol. 117, pp. 345–357, 2019.

[14] A. C. Bahnsen, E. C. Bohorquez, S. Villegas, J. Vargas, and F. A. González, “Classifying phishing URLs using recurrent neural networks,” in *Proc. APWG Symp. Electron. Crime Res. (eCrime)*, 2017, pp. 1–8.

[15] S. Hochreiter and J. Schmidhuber, “Long short-term memory,” *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, 1997.

[16] H. Le, Q. Pham, D. Sahoo, and S. C. H. Hoi, “URLNet: Learning a URL representation with deep learning for malicious URL detection,” *arXiv:1802.03162*, 2018.

[17] V. Le Pochat, T. Van Goethem, S. Tajalizadehkhoob, M. Korczyński, and W. Joosen, “Tranco: A research-oriented top sites ranking hardened against manipulation,” in *Proc. Netw. Distrib. Syst. Secur. Symp. (NDSS)*, 2019.

[18] C. E. Shannon, “A mathematical theory of communication,” *Bell Syst. Tech. J.*, vol. 27, no. 3, pp. 379–423, 1948.

[19] V. I. Levenshtein, “Binary codes capable of correcting deletions, insertions and reversals,” *Sov. Phys. Dokl.*, vol. 10, no. 8, pp. 707–710, 1966.

[20] F. Pedregosa *et al.*, “Scikit-learn: Machine learning in Python,” *J. Mach. Learn. Res.*, vol. 12, pp. 2825–2830, 2011.

[21] T. Chen and C. Guestrin, “XGBoost: A scalable tree boosting system,” in *Proc. 22nd ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, 2016, pp. 785–794.

[22] L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone, *Classification and Regression Trees*. Belmont, CA, USA: Wadsworth, 1984.

[23] L. Breiman, “Random forests,” *Mach. Learn.*, vol. 45, no. 1, pp. 5–32, 2001.

[24] C. Cortes and V. Vapnik, “Support-vector networks,” *Mach. Learn.*, vol. 20, no. 3, pp. 273–297, 1995.