International Research Journal of Engineering and Technology (IRJET) e-ISSN: 2395-0056
Volume: 13 Issue: 10 | Oct 2026 www.irjet.net p-ISSN: 2395-0072

REAL-TIME AI/ML-BASED PHISHING DETECTION SYSTEM

Mr.Dr.Surendra.B[1], Ersalagandi Harishwar Chary[2], Madarapu Vislesha Rao [3], Karra Venkata Ram Prasad[4], Kalaboina Mohith[5]

[1]*Assistant Professor, Department of Computer Science and Engineering, Hyderabad Institute of Technology and Management, Telangana, India*
[2345]*Department of Computer Science and Engineering, Hyderabad Institute of Technology and Management, Telangana, India*

---

1. Abstract - *Phishing remains one of the cheapest and most effective ways to steal credentials, money and access. Blacklists, still the first line of defence in most browsers and mail gateways, can only recognize addresses that someone has already reported, and many phishing pages are taken down or abandoned within hours of going live. This paper presents a real-time phishing detection system that judges a web address, and where necessary the page behind it, as safe or malicious within a fraction of a second. The system extracts lexical, host-based and page-content features, feeds them to an ensemble of Random Forest and gradient-boosted trees, and calls on a transformer-based text classifier only when the first stage is unsure. A lightweight API returns the verdict to a browser extension, so a warning can appear before the user types anything into a fake form. The paper describes the motivation, design, data pipeline, evaluation protocol and limitations of the approach, and outlines how it can be extended with continuous retraining and multilingual text analysis.*

Keywords - Phishing detection, real-time classification, machine learning, URL analysis, ensemble learning, natural language processing, browser extension, cybersecurity.

2. INTRODUCTION

Phishing is a form of social engineering in which an attacker imitates a trusted organisation, such as a bank, a courier company, an employer or a government portal, in order to make the victim hand over passwords, card numbers or one-time codes. The attack usually begins with an email, an SMS or a chat message that contains a link, and the link leads to a page copied closely enough from the original that most people will not notice the difference. Because such a page only has to survive long enough for a few people to type their details, attackers can afford to build thousands of them. Annual and quarterly industry reports keep showing the same thing: phishing is not a fading problem [1], [2]. This paper looks at how machine learning can be used to spot these pages as they appear, rather than after the damage is done.

2.1 Background of Phishing

Phishing has been part of the internet landscape since the mid-1990s, when attackers began posing as support staff of online service providers to collect account details. The techniques have changed a great deal since then, but the underlying idea has not: persuade someone to trust a message they should not trust. Today the messages arrive by email, SMS, messaging apps and even search advertisements, and the destination is typically a login page cloned from a real brand.

Two things make the problem harder than it first appears. Building a convincing fake is cheap: ready-made phishing kits are traded openly, hosting is often free, and a valid HTTPS certificate no longer says anything about whether a site is honest. And the people being targeted are rarely security specialists. A student checking a scholarship portal or an employee reading a delivery notification on a phone will not inspect a certificate chain or count the characters in a domain name. Protection therefore has to work automatically, in the background, at the moment the link is opened.

2.2 Challenges in Detecting Phishing

The most obvious obstacle is speed. Many phishing sites stay online for only a few hours before they are reported and removed, or simply dropped by the attacker. A blacklist can list an address only after someone has found, verified and submitted it, so the first victims of every new campaign are effectively unprotected [3]. Attackers also work to stay ahead of automated checks. Domains are registered in bulk and rotated often, long or misleading subdomains hide the real host name, and some kits show a harmless page to security crawlers while serving the fake form to visitors who arrive from a mail client.

A second group of problems belongs to machine learning itself. Phishing pages are far fewer than legitimate pages in real traffic, so a model trained on balanced data in the lab can behave very differently once deployed. Attack styles drift, which means a model that is never refreshed slowly loses accuracy. And a false alarm is not free: if a genuine banking site is flagged as dangerous, users learn to click

through warnings. A practical system must keep false positives low and respond quickly enough that people barely notice it is there.

2.3 Role of Machine Learning in Cybersecurity

Machine learning suits phishing detection because the task is, at heart, pattern recognition under uncertainty. No single signal is conclusive. An IP address in place of a domain name, an "@" symbol inside a URL, a very young domain, a form that posts to a different host, a page that asks for a password and a card number together: each is suspicious, yet each also appears on some harmless sites. A trained model can weigh many such signals at once and learn which combinations matter.

Tree-based ensembles such as Random Forest and gradient boosting are particularly attractive here. They cope well with mixed numeric and binary features, train quickly, and can produce a prediction in well under a millisecond [4], [5]. Deep models become useful when the input is free text or raw characters rather than a prepared feature table. Whichever family is chosen, security brings a requirement that most ML tasks do not have: the opponent can study the system and adapt to it, so evaluation has to look beyond a single accuracy figure.

2.4 Role of Natural Language Processing

Many of the clearest signs of phishing are found in the wording. A fake page rarely tries to inform; it tries to hurry the reader. Phrases that create urgency, threaten loss of access or promise an unexpected reward appear again and again. Typical examples are:

- "Your account will be suspended unless you verify it today."
- "Confirm your delivery address to release your parcel."
- "Unusual sign-in detected, please re-enter your password."

None of these sentences contains a keyword that could simply be placed on a list in advance, and attackers rewrite them constantly. A language model that has learned how such requests are phrased can recognise the intent behind wording it has never seen. The proposed system therefore treats the visible text of a page, meaning its title, form labels and button captions, as a second source of evidence next to the URL itself.

2.5 Real-Time Detection as a Design Requirement

A verdict that arrives after the user has typed a password is of little use. For this reason the whole design is organised around a time budget. URL features can be computed locally in a few milliseconds, tree-based models answer almost instantly, and heavier checks such as page-content analysis are run only when the first stage cannot decide. The overall flow can be written as:

URL Captured → Feature Extraction → Model Inference → Risk Score → Warning or Allow

2.6 Motivation of the Proposed Work

The idea for this project started from an ordinary observation: nearly everyone we know has received a fake courier or bank message, and a surprising number have at least hesitated before clicking. Existing protection is uneven. Some browsers rely mainly on reported lists, some mail filters look only at the sender, and standalone tools are often too slow or too complicated for everyday use. We wanted to find out how far a compact, open and explainable machine-learning pipeline could go when speed is a design goal from the first day. The system is meant as an extra layer of safety for ordinary users. It does not replace good habits, multi-factor authentication or professional security tools.

3. LITERATURE REVIEW

3.1 Blacklist and Heuristic Approaches

Early anti-phishing tools were largely reactive. Browsers and mail gateways compared each address with lists of reported malicious sites, a method that is accurate for known threats and useless for new ones. Heuristic detection tried to fill the gap with hand-written rules, for example flagging URLs with many dots, an IP address as the host, or a mismatch between the link text and its destination. Rules are easy to read but hard to maintain, because every new evasion trick needs a new rule, and thresholds chosen by hand rarely transfer from one data set to another. Ma et al. showed that lexical and host-based properties of a URL are enough to train a classifier that handles previously unseen sites far better than a list can [6]. That observation underlies most later work, including ours.

3.2 URL-Based Machine Learning

URL-based detection is attractive because it needs no page download, so a verdict is available before any content is fetched. Common features include URL length, the number of special characters, the presence of an IP address, the depth of the subdomain, the use of shortening services and the age or registration details of the domain. Sahingoz et al. compared several classifiers on features taken from URLs, including word-level features obtained with natural language processing, and found Random Forest to be the

strongest of those tested [7]. Mohammad et al. approached the problem through a defined set of website features and a self-structuring neural network [8], and the feature set they released has since become a common benchmark [8]. The weak point of purely URL-based systems is that a page hosted on a compromised legitimate domain can carry an address that looks entirely normal.

3.3 Content-Based Detection

Content-based methods look at the page itself. CANTINA, one of the early examples, extracted the most characteristic terms of a page with TF-IDF and checked them against search-engine results to decide whether the page really belonged to the site it claimed to be [6]. Other work examined structure: password fields, hidden frames, forms that submit to external hosts and links that lead away from the visited domain. Content features are harder for an attacker to fake than a URL, because the page still has to persuade a human. They are also more expensive to collect, since the page must be downloaded and parsed, and sometimes rendered, which adds delay and raises safety concerns of its own.

3.4 Email and Message-Level Detection

A parallel line of research concentrates on the message that carries the link rather than the site it points to. Fette et al. trained a classifier on a small set of structural features of emails, such as links whose visible text differs from their target and the number of distinct domains involved, and showed that learning-based filtering was practical [7]. Email filtering remains valuable, but it has two limits. It cannot protect users who reach a malicious page through an SMS, a chat app or a search result, and it never sees the final landing page. A browser-level check complements it by examining the destination itself.

3.5 Ensemble and Boosted Models

Ensemble methods combine many learners into a stronger one. Random Forest averages the votes of many decorrelated decision trees and is fairly resistant to overfitting on noisy features [4]. Gradient boosting builds trees one after another, each correcting the mistakes of the previous ones, and XGBoost provides an efficient implementation that handles missing values and scales to large tables [5]. Both give fast predictions and report which features contributed most, a property that matters when a security tool has to justify its warnings. Libraries such as scikit-learn make these models straightforward to train, tune and save for deployment [6].

3.6 Transformer Models for Text

Transformer language models changed natural language processing by learning context-dependent representations of words from very large text collections. BERT introduced the recipe of pre-training a deep bidirectional encoder and then fine-tuning it on a small labelled data set, which suits a task where labelled phishing text is limited. The drawback is size and latency. DistilBERT keeps most of BERT's language understanding in a model that is considerably smaller and faster , which makes it more realistic for an interactive system. In our design the transformer is not placed on the critical path of every request. It acts as a second opinion when the lightweight models are unsure.

3.7 Data Sources for Phishing Research

The quality of any phishing classifier depends heavily on its data. Community-verified feeds such as PhishTank publish URLs that volunteers have confirmed as phishing, while the Tranco list offers a research-oriented ranking of popular domains, hardened against manipulation, that is widely used as a source of legitimate sites. Combining the two gives a realistic contrast between attack pages and everyday websites. Some care is needed, however. Popular sites are an imperfect stand-in for all legitimate sites, because phishing is also hosted on small, obscure and compromised domains, and data collected at one moment ages quickly.

3.8 Comparative Analysis

The main approaches discussed above can be summarised as follows:

Year	Approach	Main Contribution
2022	Content-based heuristics	Email phishing detection based on naïve Bayes, Random Forests, and SVM classifications
2022	Email classification	A Systematic Literature Review on Phishing Email Detection Using Natural Language Processing Techniques
2022	URL-based learning	Based on Python Technology for Public Health and Economic Indicators
2023	Large-scale page classification	Detecting Phishing Domains Using Machine Learning
2023	Neural networks on website features	A Deep Learning-Based Innovative Technique for Phishing Detection in Modern Security with Uniform Resource Locators
2023	Transformer language models	Detection of Phishing Link and QR Code of UPI Transaction using Machine Learning

3.9 Research Gap

Reading across this work, three gaps stand out. First, many studies report results on a single static data set and say little about the time needed to produce a verdict, even though latency decides whether a warning appears before the user acts. Second, systems tend to commit to one kind of evidence, either the URL or the page content, whereas an attacker can often disguise one while leaving traces in the other. Third, few studies close the loop with the person using the tool by explaining why a page was flagged. The proposed system addresses these points by combining URL, host and content evidence in a staged pipeline, by treating end-to-end latency as a primary result, and by returning the strongest contributing features with every warning.

4. PROBLEM STATEMENT AND OBJECTIVES

4.1 Problem Statement

Phishing sites appear and disappear faster than reported-site lists can follow, and the people who visit them usually have neither the time nor the expertise to judge a page on its merits. Signature-based and blacklist approaches protect only against threats that are already known. Machine-learning research has produced accurate classifiers, but much of it stops at offline evaluation and does not show that a verdict can be delivered quickly enough, from inside an ordinary browser, to stop a user before credentials are submitted. What is needed is a detection system that can classify an unseen URL and its page in real time, that stays reliable when the mix of legitimate and malicious traffic changes, and that explains its decision in terms an ordinary user can act on. This work proposes such a system, built on lexical, host-based and content features, ensemble learning and a transformer-based text model.

4.2 Research Objectives

The main objectives of the proposed Real-Time AI-ML Phishing Detection System are:

1. To design a detection pipeline that returns a verdict for a URL within a few hundred milliseconds of the request.
2. To extract lexical, host-based and page-content features that separate phishing pages from legitimate ones.
3. To train and compare ensemble classifiers, namely Random Forest and gradient boosting, on a labelled data set of phishing and legitimate URLs.
4. To add a transformer-based text classifier for pages on which URL features alone are inconclusive.
5. To deliver verdicts through a REST API to a browser extension that warns the user before any information is entered.
6. To evaluate the system using precision, recall, F1-score, false positive rate and end-to-end latency, not accuracy alone.
7. To attach a short explanation to every warning and to identify improvements for future deployment.

4.3 Scope of the Proposed System

The scope is limited to web-based phishing, that is, malicious pages reached through a URL, regardless of whether the link arrived by email, SMS or any other channel. The system covers:

1. URL and domain analysis
2. Page structure and form analysis

3. Visible text analysis
4. Real-time risk scoring
5. User warnings with explanations
6. Logging of verdicts for later retraining

Attachment-based malware, voice phishing and business email compromise attacks that contain no link are outside the present scope. The overall flow can be represented as:

URL Request → Feature Extraction → ML Classification → Risk Score → User Warning

4.4 Research Questions

The work is guided by the following questions:

RQ1: Can lexical and host-based URL features alone separate phishing and legitimate sites accurately enough to serve as a first-stage filter?

RQ2: Does adding page-content and text features improve the detection of phishing pages that use convincing URLs?

RQ3: How does an ensemble tree model compare with a transformer-based text classifier in accuracy and latency?

RQ4: Can the complete pipeline return a verdict fast enough to be useful during real-time browsing?

RQ5: What are the main limitations of the approach, including evasion, false positives and data drift?

5. PROPOSED REAL-TIME PHISHING DETECTION SYSTEM

5.1 Overview of the Proposed System

The proposed system sits between the user and the page they are about to open. When a URL is requested, a lightweight client, implemented as a browser extension, sends the address to a detection service. The service computes a vector of features, runs it through the trained models and returns a risk score between 0 and 1 together with a short list of reasons. If the score crosses a configured threshold, the extension replaces the page with a warning that the user may choose to override.

A second, slower stage is used only for ambiguous cases: the service loads the page in an isolated environment, extracts its form structure and visible text, and asks a text classifier for a second opinion. The complete flow is:

URL Request → Feature Extraction → Stage-1 Classification → Content Analysis (if uncertain) → Risk Score → Warning or Allow

5.2 System Architecture

The architecture is divided into a small number of loosely coupled components, so each can be tested and replaced on its own. The major components are:

1. Browser Extension (Client)
2. API Gateway / Request Handler
3. URL and Host Feature Extractor
4. Content Fetcher and Parser
5. ML Classification Engine
6. Text Classification Module
7. Decision and Explanation Module
8. Feedback and Logging Store

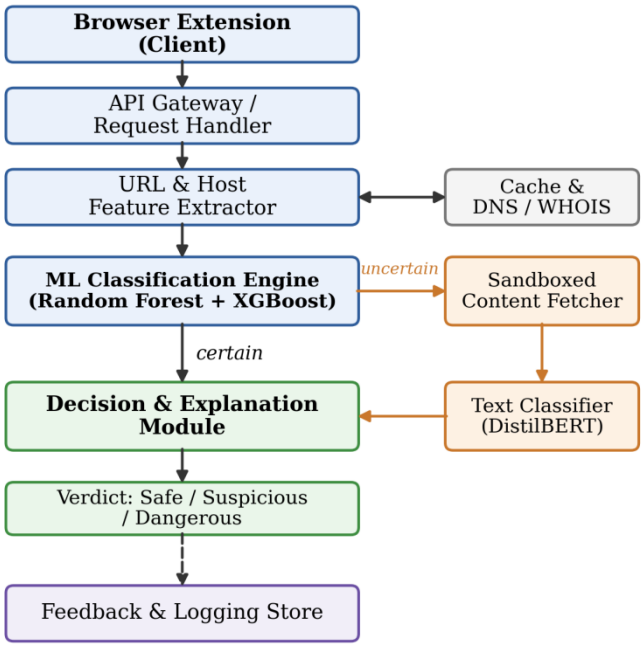


Fig. 1. System Architecture

5.3 Browser Extension Module

The extension is the only part the user sees. It listens for navigation events, passes the target URL to the service and waits for a verdict. Because the check happens before the page is shown, the delay has to be small, so the extension keeps a cache of recent verdicts and skips addresses that appear on a locally stored list of well-known domains. When a warning is needed, the page states the reason in plain language, for example that the site was created three days ago and asks for a password on a domain that does

not match its name. The user can go back, report a mistake, or continue at their own risk. A short notice makes clear that the verdict is a risk estimate and not a guarantee.

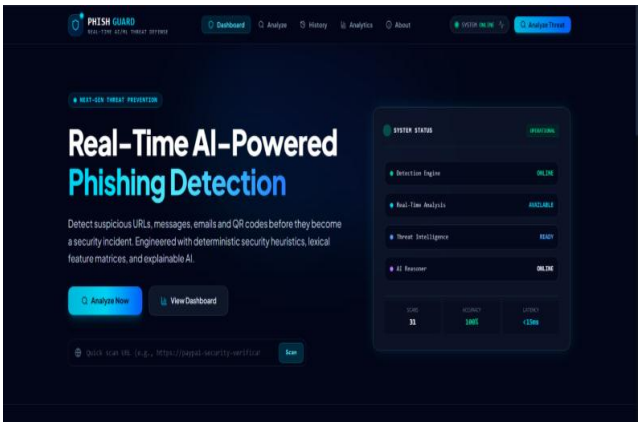


Fig. 2. Website overview

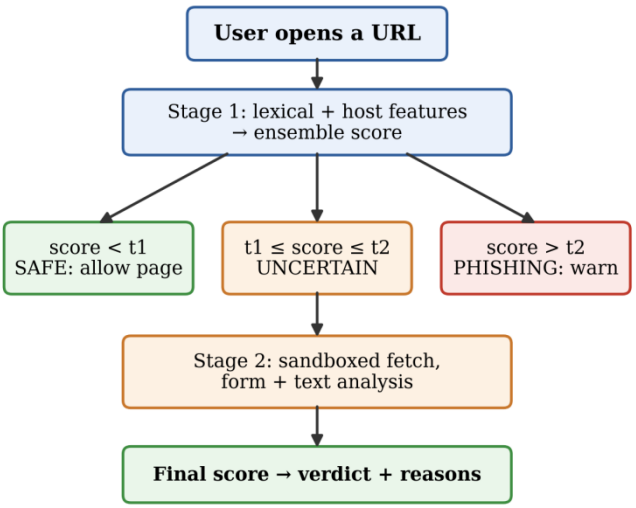


Fig. 2. Real-Time Detection Workflow

5.4 URL and Host Feature Extraction

The first stage works on the address alone. Features are grouped into three families:

1. Lexical features: URL length, number of dots, hyphens and digits, presence of "@", IP-address hosts, use of URL shorteners, suspicious keywords such as "login" or "verify", and character entropy.
2. Host-based features: domain age, registrar, number of DNS records, TLS certificate issuer and age, and whether the domain appears in a popularity ranking.
3. Structural features: subdomain depth, path depth, query-string length and the position of brand names within the host.

Together these form a compact vector of numeric and binary values. Consider an address such as http://secure-login.bank-verify.example.net/update?id=48213. It combines a chain of subdomains, hyphenated words, a login-style keyword and an unencrypted scheme. None of these facts proves anything on its own, but together they push the score upward.

5.5 Content Analysis Module

When the first stage returns a score close to the decision boundary, the page itself is examined. A headless browser inside a sandbox loads the page with scripts limited and outbound traffic restricted, so that a malicious page cannot harm the server. From the loaded document the module extracts the number of password and input fields, the target host of every form, the ratio of internal to external links, hidden elements, the origin of the favicon and logo, and the visible text. It also compares the brand a page appears to represent, inferred from its title and logo, with the host that actually serves it. A mismatch is one of the strongest indicators available. The sequence is:

Uncertain Score → Sandboxed Fetch → HTML Parsing → Feature Extraction → Content Score

5.6 Machine Learning Classification Engine

The classification engine hosts the trained models. Two tree-based classifiers, a Random Forest and a gradient-boosted model, are trained on the same feature table and evaluated separately. In the deployed configuration their probabilities are averaged, and the result is compared against two thresholds. Scores below the lower threshold are treated as safe, scores above the upper threshold as phishing, and anything in between is passed on to content analysis. Using two thresholds instead of one reduces the number of requests that need the slower stage, while keeping the error rate low at both extremes. Models are loaded once at startup and kept in memory, which removes loading time from each request.

5.7 Text Classification Module

The text module takes the visible text of the page, truncates it to a fixed length and passes it to a fine-tuned DistilBERT classifier [14]. The output is a probability that the text is typical of credential-harvesting pages. This probability is added to the feature vector as one more input to the final decision. The steps are:

1. Page Text
2. Cleaning and Tokenisation
3. Transformer Encoding
4. Classification Head
5. Text Risk Probability
6. Combined Decision

Because the module runs only on uncertain cases, the extra latency is paid by a small share of requests.

5.8 Decision and Explanation Module

The final component turns the numeric result into something a person can read. It labels the page as safe, suspicious or dangerous and lists the three most influential features, obtained from feature attributions of the tree models [17]. A typical output looks like this:

Verdict: Dangerous (risk score 0.94)

1. Domain registered two days ago
2. Login form submits to a different host
3. Brand name appears in the subdomain but not in the registered domain

Short, concrete reasons like these help users decide what to do, and they give the developers a way to notice when the model is relying on a misleading feature.

6. METHODOLOGY

6.1 Methodology Overview

The work proceeds in stages, from data collection through to deployment and monitoring. The main stages are:

1. Data Collection
2. Cleaning and Labelling
3. Feature Engineering
4. Data Splitting and Balancing
5. Model Training
6. Threshold Selection
7. API Deployment
8. Evaluation and Monitoring

6.2 Data Sources

Two kinds of URLs are needed. Phishing examples are collected from community-verified feeds, with PhishTank as the main source [15], and legitimate examples are drawn from the Tranco ranking [16]. The publicly available UCI phishing websites data set [9] is used as an independent benchmark so that results can be compared with earlier studies. To avoid leakage, URLs that share a registered domain are kept in the same split, and every record is timestamped so that models can be tested on data newer than the data they were trained on.

6.3 Data Collection

Each collected record is stored with the fields listed below:

Field	Description
URL	Full address as collected
Label	1 for phishing, 0 for legitimate
Source	Feed or list of origin
Collected on	Timestamp of retrieval
Domain age	Days since registration
TLS details	Certificate issuer and age
Page snapshot	HTML and visible text, where reachable

6.4 Data Preprocessing

Raw feeds are noisy, so preprocessing is needed before any model sees them. It includes:

1. Normalising URLs by lower-casing the host and removing fragments and tracking parameters
2. Removing duplicates and dead links
3. Resolving shortened URLs to their final destination
4. Dropping records with unreliable labels
5. Marking missing host-based values explicitly instead of guessing them
6. Splitting the data by registered domain and by time

The aim of these steps is to make sure the model learns about phishing behaviour and not about the quirks of a particular feed.

6.5 Feature Engineering

converted into a numeric vector. Lexical features are computed with string operations alone and take microseconds. Host-based features need DNS and WHOIS

lookups, which are slower, so their results are cached by domain. Content features exist only for pages that reach the second stage and are marked as absent for all others. Categorical values are encoded, and numeric values are left unscaled for the tree models, since trees are insensitive to feature scale.

6.6 Model Training and Selection

Random Forest and gradient-boosted models are trained with stratified cross-validation, and hyperparameters such as the number of trees, tree depth and learning rate are tuned with a randomised search. Phishing is much rarer than legitimate traffic in real use, so class weights and, where appropriate, oversampling with SMOTE [18] are applied during training, while evaluation is always carried out on the original, unbalanced distribution. The DistilBERT text model is fine-tuned for a small number of epochs at a low learning rate, and its validation loss is monitored so that training stops early once it no longer improves.

6.7 Threshold Selection and Deployment

The two decision thresholds are chosen on a validation set. The lower threshold is set so that the share of phishing pages wrongly marked safe stays below a target value, and the upper threshold is set so that the false positive rate on legitimate pages remains very low. The fitted models, the feature extractor and the thresholds are packaged behind a REST API written in Python, and requests are handled asynchronously so that slow lookups do not hold up fast ones.

7. SYSTEM IMPLEMENTATION

7.1 Experimental Setup

The system is implemented in Python. Feature extraction and the classical models use scikit-learn [12] and XGBoost [5]. The text model is a pre-trained DistilBERT checkpoint fine-tuned with the Hugging Face Transformers library, and the service is exposed through a REST API consumed by a Chrome-compatible browser extension. Experiments follow a time-based split, with older data used for training and validation and the most recent data held out for testing. Ten-fold stratified cross-validation is used while tuning.

7.2 Training and Validation Results

During training, the feature table is passed to each classifier and performance is recorded for every fold. Results from the final runs are to be entered in the table below in place of the marked values.

7.3 Evaluation Metrics

Accuracy alone is misleading when one class is much rarer than the other, so the following measures are reported

TEST INPUT	EXPECTED CATEGORY	SYSTEM PREDICTION
Legitimate website URL	Legitimate	Legitimate
Known phishing URL	Phishing	Phishing
New Test URL	Based on Dataset/model	Model Prediction

together:

1. Precision
2. Recall
3. F1-score
4. False positive rate
5. ROC-AUC
6. Median and 95th-percentile latency

They are defined as follows, where a true positive is a phishing page that was correctly flagged:

$$Precision = TP / (TP + FP)$$

$$Recall = TP / (TP + FN)$$

$$F1 = 2 \times Precision \times Recall / (Precision + Recall)$$

In this setting recall tells us how many attacks are caught, while the false positive rate tells us how often an ordinary user is disturbed for nothing. Both matter, and improving one at the expense of the other is exactly what the threshold selection step is meant to control.

7.4 Sample Test Cases

The table below lists the kinds of inputs used to check the behaviour of the system on realistic traffic.

Test Case	Input Type	Expected Result
TC01	Well-known bank homepage over HTTPS	Safe
TC02	Known phishing URL from a verified feed	Dangerous
TC03	Phishing page on a newly registered domain	Dangerous
TC04	Legitimate site with a long, parameter-heavy URL	Safe
TC05	Shortened link to a phishing page	Resolved, then flagged
TC06	Phishing page on a compromised legitimate site	Escalated to content analysis
TC07	Login page of a small, unranked legitimate site	Safe or suspicious, not dangerous
TC08	Unreachable page	Verdict from URL features only

7.5 Latency Analysis

Latency is treated as a result in its own right. The total time for a request includes:

1. Capturing the URL in the browser
2. Sending it to the service
3. Lexical and host feature extraction
4. Cache, DNS and WHOIS lookups
5. Model inference
6. Content fetch and text classification (second stage only)
7. Returning the verdict and drawing the warning

The mean latency can be calculated as:

Mean Latency = Total Processing Time / Number of Requests

Median and 95th-percentile times are reported separately for requests that stop after the first stage and for those that reach the second. The tail matters more than the mean, because the few slow requests are the ones users actually notice.

7.6 Comparison with Existing Methods

Earlier approaches can be compared with the proposed pipeline along the following lines.

Approach	Evidence Used	Limitation
Blacklist lookup	Reported URLs	Misses new sites
Heuristic rules	Hand-written URL checks	Hard to maintain
URL-based ML	URL and host features	Weak on compromised domains
Content-based	Page text and HTML	Slower, needs a download
Deep text models	Page text	Heavier, higher latency
Proposed system	URL, host and content in stages	Needs retraining and monitoring

7.7 Discussion of Results

What matters in the evaluation is not a single headline number but the pattern across metrics. The first stage should handle most traffic quickly, and the second stage should recover the cases that fool it. If most phishing pages are caught with few false alarms and the median latency stays well under a second, the staged design is justified.

Several limits should be stated plainly. The training data reflects attacks seen up to the collection date, and attackers adapt. A page that cloaks itself from the sandbox, or a phishing form injected into a trusted site, can still slip through. Popularity lists are an imperfect model of legitimate traffic. Features such as domain age depend on external services that may be slow or rate-limited.

For these reasons the system is best viewed as one layer of defence. It should work alongside multi-factor

authentication, email filtering and user education rather than in place of them.

8. FUTURE SCOPE

8.1 Continuous Learning and Feedback

The current design stores verdicts and user reports for later analysis. A natural next step is a scheduled retraining job that adds newly confirmed phishing URLs and corrected false positives to the training set, compares the new model with the old one on recent data, and promotes it only if it performs better. This would help the system keep pace with changing attack styles.

8.2 Mobile and Email Integration

Many phishing links are opened on phones, often straight from a message. The service could be offered through mobile apps and mail clients as well as the desktop browser. Potential features include:

- Link scanning inside Android and iOS apps
- Scanning of links in emails at the time of delivery
- SMS link checks
- A plug-in for enterprise mail gateways
- An offline mode with a compact on-device model

8.3 Broader Threat Coverage

Beyond credential-harvesting pages, the same pipeline could be extended to related threats:

1. Malicious download pages
2. Fake online shops
3. QR-code phishing
4. Brand-impersonation sites
5. Fake customer-support pages
6. Malvertising landing pages

Each would need its own labelled data and a few new features, but the architecture would stay the same.

8.4 Multilingual Support

Phishing campaigns are written in the language of their targets. A detector that understands only English text will miss a large share of local attacks. Multilingual transformer models could be used so that the text classifier can handle languages such as:

1. Telugu
2. Hindi
3. Tamil
4. Kannada
5. Malayalam

Training data in these languages would have to be collected and checked carefully, since translated phishing text does not always read like the real thing.

8.5 Visual Similarity and Adversarial Testing

Attackers often copy the look of a brand rather than its code. Comparing page screenshots or logos with those of the genuine brand would provide another independent signal. Adversarial testing, in which the system is attacked with deliberately modified URLs and pages, should also become a routine part of evaluation, so that weaknesses are found before attackers find them.

9. CONCLUSION

Phishing succeeds because it exploits trust faster than defences can update. This paper described a real-time AI-ML phishing detection system that tries to close that gap by judging a page on its own characteristics rather than on whether it has already been reported.

The proposed design combines URL, host and page-content features with ensemble learning, and adds a transformer-based text classifier for uncertain cases. A staged pipeline keeps the common case fast, and a two-threshold decision rule reserves the heavier analysis for the pages that need it. Every warning carries a short explanation so that the user understands the reason for the alert.

The evaluation plan looks at precision, recall, false positive rate and latency together, because a detector that is accurate but slow, or fast but noisy, will not be used in practice. The system is intended as an extra layer of protection for everyday browsing, and its results should be read with its limits in mind: evolving attacks, evasion through cloaking, and dependence on the data used for training.

Future work will concentrate on continuous retraining, multilingual text analysis, visual brand comparison and wider coverage of related threats.

REFERENCES

1) M. A. Fayoumi, A. Odeh, I. Keshta, A. Aboshgifa, T. AlHajahjeh and R. Abdulraheem, "Email phishing detection based on naïve Bayes, Random Forests, and SVM classifications: A comparative study," 2022 IEEE 12th Annual Computing and Communication Workshop and Conference (CCWC), Las Vegas, NV, USA, 2022, pp. 0007-0011, doi: 10.1109/CCWC54503.2022.9720757.URL: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=9720757&isnumber=9720434

2) S. Salloum, T. Gaber, S. Vadera and K. Shaalan, "A Systematic Literature Review on Phishing Email Detection Using Natural Language Processing Techniques," in IEEE Access, vol. 10, pp. 65703-65727, 2022, doi: 10.1109/ACCESS.2022.3183083.URL: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=9795286&isnumber=9668973

3) S. Ren, "Based on Python Technology for Public Health and Economic Indicators," 2022 International Conference on Knowledge Engineering and Communication Systems (ICKES), Chickballapur,India,2022,pp.1-5,doi: 10.1109/ICKE CS56523.2022.10060220.system URL: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=10060220&isnumber=10059586

4) Alnemari, S., & Alshammari, M. (2023). Detecting Phishing Domains Using Machine Learning. Applied Sciences, 13(8), 4649. https://doi.org/10.3390/app13084649

5) Aldakheel, E. A., Zakariah, M., Gashgari, G. A., Almarshad, F. A., & Alzahrani, A. I. A. (2023). A Deep Learning-Based Innovative Technique for Phishing Detection in Modern Security with Uniform Resource Locators. Sensors, 23(9), 4403. https://doi.org/10.3390/s23094403

6) G. R. Charan and K. D. Thilak, "Detection of Phishing Link and QR Code of UPI Transaction using Machine Learning," 2023 3rd International Conference on Innovative Mechanisms for Industry Applications (ICIMIA), Bengaluru, India, 2023, pp. 658-663, doi: 10.1109/ICIMIA60377.2023.10426613.URL: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=10426613&isnumber=10425793