RESEARCH ARTICLE OPEN ACCESS

AI-Based Cybercrime Detection and Prevention System

A. Mahendhar, M. Vamshikrishna, Paila Arjun, V. Abhinay, Ajith Kumar
Department of Artificial Intelligence & Machine Learning, Parul Institute of Engineering and Technology
Parul University, Vadodara, Gujarat, India
Under the Guidance of Prof. Dushyant Suryavanshi, AI & ML Department

Abstract—digital services have expanded rapidly, and cybercrime has become one of the most pressing threats to secure online interaction. Traditional signature- and rule-based security tools are largely reactive, so they struggle to keep pace with newly evolving attacks such as phishing, malicious URL distribution, and identity theft. This paper presents an AI-Based Cybercrime Detection and Prevention System that uses supervised machine learning to classify URLs and user-interaction patterns as safe or malicious in real time. The system extracts lexical and host-based features from each submitted URL and applies a Random Forest classifier to carry out automated threat classification, and it exposes this functionality through a Flask-based web dashboard for end users and administrators. The paper covers the software requirements specification, the layered system architecture, the Agile-based development methodology, and the implementation stack (Python, Flask, HTML/CSS/JavaScript, and a relational/NoSQL database) used to build the system. A review of twenty-five related studies is used to position the proposed design against existing phishing-detection, intrusion-detection, and explainable-AI literature. The results suggest that ensemble tree-based classifiers such as Random Forest strike a favourable balance between detection accuracy, interpretability, and computational cost for real-time deployment, while the accompanying dashboard automates threat logging and reduces the manual effort required of security analysts. The paper closes with a discussion of current limitations and directions for future work, including deep-learning integration, threat-intelligence API feeds, and federated, privacy-preserving training.

Keywords—Cybersecurity, Machine Learning, Phishing Detection, Random Forest, Feature Extraction, URL Classification, Web Security, Threat Intelligence, Flask, Software Requirements Specification.

I. INTRODUCTION

The rapid growth of internet usage and online services has changed the way people and organisations communicate, transact, and share data. That shift has brought with it a sharp rise in cybercrime, including phishing attacks, identity theft, data breaches, and online fraud, all of which cause financial loss and compromise sensitive information. Traditional cybersecurity systems rely mainly on predefined rules and signature-based detection, which work well against known threats but often fail to catch new and evolving attacks, since cybercriminals are constantly devising techniques to get around static security mechanisms.

To address this gap, this paper presents an AI-Based Cybercrime Detection and Prevention System that applies machine learning, specifically a Random Forest classifier, to analyse URL and interaction-pattern features and classify them as safe or malicious in real time. The system also provides real-time feedback through a web-based interface and an administrator monitoring dashboard, so it moves beyond detection alone and toward proactive prevention.

A. Problem Statement

Signature- and rule-based cybersecurity tools depend on previously catalogued attack patterns, so they inevitably lag behind newly registered phishing domains and novel malicious-URL structures. Manual review of flagged URLs simply does not scale against the volume of automated phishing campaigns that modern threat actors generate. What is needed, then, is an adaptive system that can learn discriminative patterns from historical data, generalise to malicious URLs it has never seen before, and present its findings through an interface that both non-technical end users and security analysts can use.

B. Objectives

- To design a machine-learning pipeline that extracts discriminative lexical and host-based features from URLs.
- To train and deploy a Random Forest classifier capable of real-time, high-accuracy phishing/malicious-URL detection.
- To expose the classifier through a responsive, user-friendly web dashboard for General Users and Administrators.
- To log and store all predictions for auditing, reporting, and future model retraining.
- To evaluate the design against a structured Software Requirements Specification covering performance, security, and usability.

The present work focuses on phishing and malicious-URL detection, one of the most prevalent forms of cybercrime, while the architecture is built to extend to other threat categories such as malware and network intrusion. The paper's principal contributions are summarised below.

- A layered, modular system architecture that separates the presentation, application, and data layers for a real-time cybercrime-detection dashboard.

- A Random Forest-based URL classification pipeline combining lexical, host-based, and security-related feature extraction.
- A structured Software Requirements Specification (SRS) covering functional, non-functional, interface, and business-rule requirements for the system.
- A review and synthesis of twenty-five related studies spanning phishing detection, intrusion detection, explainable AI, and SOC-dashboard design, used to justify the proposed design choices.
- An Agile-based development and testing methodology suited to the iterative refinement of machine-learning components alongside conventional software features.

The rest of this paper is organised as follows. Section II reviews related work. Section III presents the software requirements specification. Section IV describes the proposed system architecture. Section V details the methodology, including feature extraction and model selection. Section VI describes the implementation. Section VII discusses results, and Section VIII summarises the testing strategy. Section IX concludes the paper, and Section X outlines directions for future work.

II. RELATED WORK

A substantial body of research has looked at using machine learning for cybercrime and phishing detection, and the design of the proposed system draws on twenty-five representative studies spanning classification algorithms, dashboard engineering, and explainability. This section groups the reviewed literature into six thematic clusters.

A. URL and Website-Based Phishing Detection

Several studies establish Random Forest as an effective baseline for phishing-URL classification. Mehta et al. [1] trained a supervised Random Forest model on domain length, special-character counts, HTTPS usage, and subdomain counts, and reported that the ensemble approach reduces overfitting relative to individual decision trees, although static lexical features alone cannot capture JavaScript-based redirection or zero-day phishing pages. Related work on feature extraction [6] shows that raw URLs must go through lexical and host-based preprocessing before classification, and that a modular pipeline—feature extraction, model inference, and log storage—improves maintainability. A hybrid study combining lexical and content-based features (SSL certificate presence, HTML tag frequency, and JavaScript behaviour) with Gradient Boosting and Random Forest [16] found that hybrid feature engineering improves robustness, but at the cost of extra computation from webpage crawling.

B. Network Intrusion and Malware Detection

Sharma et al. [2] applied Random Forest and Support Vector Machine models to packet-level features (size, protocol, duration, and frequency) for anomaly-based intrusion detection, and noted that real-time deployment requires scalable cloud infrastructure. A companion review of NSL-KDD and KDD Cup 99 benchmark datasets [5] confirms that ensemble trees generalise well while staying more interpretable than deep networks. Behavioural malware detection using API-call and registry-modification features [15] likewise favours Random Forest and SVM classifiers, while cautioning that high feature dimensionality raises the risk of false positives.

C. Text and Content-Based Threat Detection

Natural Language Processing techniques have also found their way into phishing and spam detection. Wei et al. [3] combined TF-IDF vectorisation with Naive Bayes and Random Forest classifiers for spam-email filtering, while a later study [11] applied transformer-based models to detect socially engineered phishing content in email and URL text, reporting better detection of semantic patterns at the cost of higher computational demand.

D. Explainability and Trust

Because security analysts need to justify why a URL gets flagged, interpretability keeps coming up as a theme. Two studies [8],[12] argue that Random Forest inherently supports feature-importance analysis and recommend surfacing these scores in the dashboard UI; a related study on SHAP-based explanation for ensemble classifiers [13] shows that explainability can be added with only a limited loss of predictive performance, though it does increase computational overhead.

E. Dashboard, Architecture, and Cloud Deployment

One cluster of studies looks at the software-engineering side of AI-driven security tools. SOC dashboard design work [7] highlights automated threat logging as the main value of a real-time scanning dashboard, and proposes evolving such tools into browser extensions. Flask-integration studies [9],[10] describe a layered architecture in which the user interacts only with the web dashboard while feature extraction, model inference, and log storage remain independently upgradeable. Cloud-deployment research [14] discusses dynamic model retraining without downtime and highlights threat-intelligence API integration [17] as a way to fortify static classifiers with live global threat feeds.

F. Advanced and Emerging Approaches

More advanced techniques reviewed include federated learning for privacy-preserving, multi-organisation phishing detection [18]; graph-based modelling of domain–IP relationships using Graph Neural Networks to detect coordinated phishing campaigns [19]; adversarial-attack studies showing that small perturbations to URL features can evade trained classifiers, with ensemble methods proving comparatively robust [20]; deep-learning intrusion detection

using CNN/LSTM architectures for sequential traffic analysis [21]; and unsupervised anomaly detection using Isolation Forests and Autoencoders for settings where labelled attack data is scarce [22].

Taken together, the reviewed literature supports two design choices adopted in this work: (i) Random Forest as the core classifier, given its favourable balance of accuracy, robustness, and interpretability relative to deep or unsupervised alternatives; and (ii) a modular, Flask-based web architecture that separates the user interface, feature extraction, model inference, and threat-log storage so that each layer can evolve independently, in line with the SOC-dashboard and cloud-deployment literature discussed above.

TABLE I. SUMMARY OF REVIEWED LITERATURE

Ref.	Author(s), Year	Method	Key Finding / Limitation
[1]	Mehta et al., 2023	Random Forest	URL lexical/host features; robust but static
[2]	Sharma et al., 2022	RF + SVM	Packet-level NIDS; needs cloud scaling
[3]	Wei et al., 2024	Naive Bayes + RF	TF-IDF spam-email classification
[4]	Gonzalez et al., 2023	RF + Logistic Reg.	Credit-risk scoring; bias concerns
[5]	Mehta et al., 2023 (review)	Survey	NIDS datasets: KDD99, NSL-KDD
[6]	ML Research Group, 2020	Random Forest	URL feature-extraction pipeline design
[7]	Security Research Grp., 2021	Dashboard	SOC-style real-time URL scanning
[8]	Various, 2019	Random Forest	Flask-based phishing detection UI
[9]	Web Dev. Research Grp., 2021	Flask	ML-web integration architecture
[10]	SE Research Grp., 2022	Architecture	Modular Flask/RF/DB layering
[11]	Info. Security Res., 2023	NLP/Transformers	Semantic phishing text detection
[12]	Data Science Grp., 2022	Preprocessing	Cybersecurity dataset cleaning
[13]	AI Ethics Grp., 2023	XAI	Feature-importance transparency
[14]	Cloud & Security Grp., 2023	Cloud Deployment	Retraining without downtime
[15]	Cybersecurity Grp., 2022	RF + SVM	Behavioural malware detection
[16]	Info. Security Res., 2023	Hybrid RF/GB	Lexical + content-based features
[17]	Cybersecurity Grp., 2023	Threat-Intel API	Live feed fusion with RF
[18]	Various, 2023	Federated Learning	Privacy-preserving multi-org training
[19]	Cybersecurity Grp., 2022	GNN	Graph-based phishing-campaign detection
[20]	AI Ethics/Security, 2023	Adversarial ML	Evasion of RF-based detectors
[21]	Various, 2023	CNN + LSTM	Deep-learning IDS, temporal patterns
[22]	AI/Data Science, 2024	Isolation Forest	Unsupervised anomaly detection
[23]	AI/Data Science, 2024	Threat-Intel Fusion	Real-time SOC dashboard feeds
[24]	AI Ethics/Security, 2023	SHAP + Ensemble	Explainable threat classification
[25]	Mohammad et al., 2015	RF (foundational)	Early phishing-URL ML benchmark

Table I consolidates the twenty-five reviewed studies. A recurring pattern is that Random Forest and other tree-ensemble methods dominate the reviewed URL- and network-level detection literature, owing to their balance of accuracy and interpretability [1],[2],[5],[6],[8],[15],[16],[25], while deep-learning [21], graph-based [19], and unsupervised [22] approaches are typically positioned as complementary extensions rather than replacements. Explainability [13],[24] and deployment architecture [7],[9],[10],[14] form a second consistent theme, which motivates the SOC-style dashboard design adopted in Section IV.

III. REQUIREMENTS ANALYSIS

The Software Requirements Specification (SRS) for the proposed system sets out the functional and non-functional requirements that guide its design and implementation, and identifies the intended user classes.

A. Functional Requirements

- User authentication and role-based authorisation for General Users and Administrators.
- URL submission and real-time analysis through the web interface.
- Automated feature extraction from submitted URLs (lexical and host-based attributes).
- Machine-learning-based classification of URLs as safe or malicious using a trained Random Forest model.
- Real-time display of prediction results to the requesting user.

- Administrator dashboard for monitoring detection results, system performance, and user activity.
- Persistent storage of submitted URLs, extracted features, and prediction outcomes in a threat-log database.

C. Non-Functional Requirements

Performance requirements call for the system to respond to user interactions within 5 seconds and complete URL analysis within 9 seconds, while supporting at least 1,000 concurrent users without significant degradation. Security requirements mandate multi-factor authentication, role-based access control, encryption of data in transit (HTTPS) and at rest, and comprehensive activity logging for auditing. Usability requirements call for an intuitive, accessible interface compliant with common accessibility guidelines, and maintainability requirements call for modular, well-documented code that supports future extension.

D. User Classes

Two principal user classes are identified. General Users interact with the system to submit URLs and view safety predictions, and they require no technical expertise. Administrators hold elevated privileges to manage datasets, monitor detection results, configure system parameters, and oversee overall system security and performance.

E. External Interface Requirements

The user interface is a responsive, browser-based graphical interface that follows modern web-design principles and keeps navigation, search, and help functions consistent across screens. It targets desktop, laptop, tablet, and smartphone hardware, and supports Windows, Android, and iOS through major browsers, namely Chrome, Firefox, Safari, and Edge. Software interfaces include a relational or NoSQL database management system for data persistence, third-party API endpoints for future threat-intelligence integration, and standard web development libraries for the front end. Communication interfaces include an in-app notification mechanism to alert users to detected threats and optional e-mail notifications for account and system updates.

F. Business Rules and Constraints

Only authenticated users may submit URLs for analysis, and every prediction shown to a user must come from the trained machine-learning model rather than a manual override. All user data and prediction results must be stored securely and must not be shared with third parties without explicit consent, and users are required to update account passwords periodically to reduce the risk of account compromise. Design constraints include dependence on the quality and availability of the labelled training dataset, the computational cost of complex models at scale, and delivery-schedule constraints that bound the feature set implemented in the initial release.

G. Operating Environment and Product Perspective

The system operates as a standalone, platform-independent web application accessible through commonly used browsers over a stable internet connection, and it is compatible with desktop, laptop, tablet, and mobile hardware running Windows, Android, or iOS. It is designed to replace purely rule-based security tools with an adaptive, learning-based alternative, while remaining extensible to future integration with external cybersecurity tools, threat-intelligence feeds, and network-monitoring systems.

H. Database and Legal Requirements

The system uses a relational database management system that supports ACID properties for consistency and reliability, with regular backups and a documented recovery plan to guard against data loss, plus a data-archiving mechanism for historical records. From a legal standpoint, the system is designed to comply with applicable data-protection regulations, including the General Data Protection Regulation (GDPR) and relevant local privacy laws, given that it processes user-submitted URLs and account information.

IV. PROPOSED SYSTEM ARCHITECTURE

The proposed system follows a three-tier layered architecture made up of a presentation layer, an application layer, and a data layer, as shown in Fig. 1. The presentation layer exposes a browser-based interface through which users submit URLs and view results. The application layer, implemented in Flask, performs input validation, feature extraction, and invokes the trained Random Forest model to obtain a classification. The data layer persists submitted inputs, extracted features, and prediction outcomes in a relational or NoSQL database (MySQL or MongoDB) for auditing, reporting, and future model retraining.

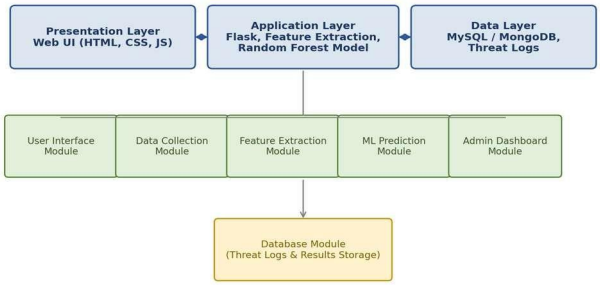

Fig. 1. Layered system architecture of the proposed AI-based cybercrime detection and prevention system.

Seven functional components realise this architecture, summarised below. Each module can be upgraded independently—for example, migrating the database to a cloud environment—without requiring changes to the

machine-learning pipeline, consistent with the modular architecture recommended in [9],[10].

- User Interface Module — accepts URL input and renders prediction results in a simple, accessible layout.
- Data Collection Module — gathers submitted URLs and associated request metadata.
- Data Processing and Feature Extraction Module — cleans input and derives lexical, host-based, and security-related features (Table II).
- Machine Learning Module — applies the trained Random Forest model to classify each input as safe or malicious.
- Result Display Module — presents the classification outcome to the end user in real time.
- Database Module — persists inputs, extracted features, predictions, and system logs.
- Administrator Dashboard Module — provides monitoring, dataset management, and reporting tools for Administrators and Security Analysts.

Fig. 2 illustrates the corresponding operational flow: submitted input is validated, preprocessed, and passed through feature extraction before classification, after which the outcome is written to the threat-log database and returned to the user or administrator.

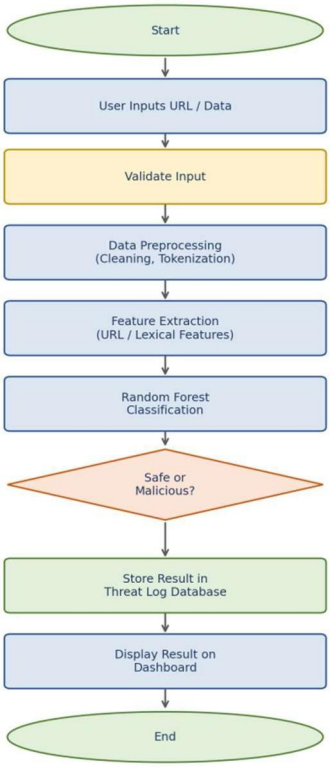

Fig. 2. Operational flowchart from user input to result storage and display.

Fig. 3 presents a Unified Modeling Language (UML) use-case view of the system. Three actors are identified: the General User, who registers or logs in, submits a URL or message, and views the prediction result; the Administrator, who manages the training dataset, monitors detection results, and generates reports; and the Security Analyst, who additionally monitors threats and triggers periodic updates of the machine-learning model. This separation of actors reflects the role-based access-control requirement set out in Section III-B.

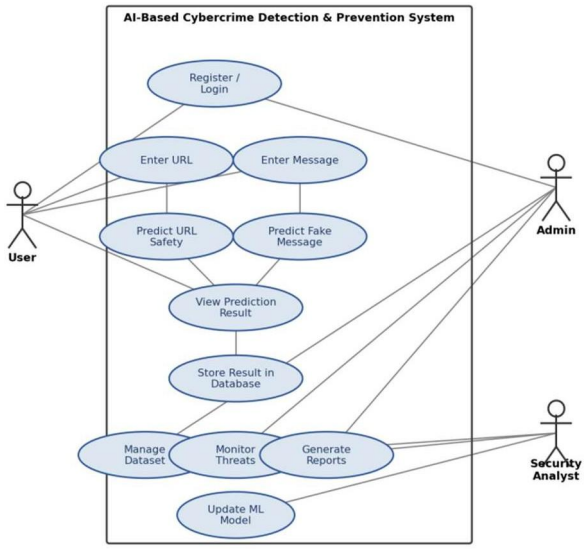

Fig. 3. Use-case diagram showing General User, Administrator, and Security Analyst interactions with the system.

V. METHODOLOGY

A. Data Collection and Preprocessing

The system is trained on labelled datasets containing both legitimate and phishing URLs, consistent with the corpora used in the reviewed literature [1],[6]. Raw URLs go through a sanitisation step to remove duplicates, malformed entries, and missing values before feature extraction, since model performance depends heavily on dataset quality [6].

B. Feature Extraction

Lexical and host-based attributes are extracted from each URL, including overall length, count of special characters, presence of an IP address in place of a domain name, subdomain count, HTTPS usage, and domain-registration attributes. Table II summarises the principal feature categories used by the classifier.

TABLE II. FEATURE CATEGORIES USED FOR URL CLASSIFICATION

Category	Representative Features
Lexical	URL length, special-character count, digit ratio
Host-based	Domain age, subdomain count, IP-address usage
Security	HTTPS usage, SSL certificate validity
Content (optional)	HTML tag frequency, JavaScript behaviour patterns

C. Classification Model — Random Forest

Random Forest was chosen as the core classifier because it consistently outperforms single decision trees in the reviewed literature while remaining more interpretable than deep-learning alternatives [1],[5],[8]. The algorithm builds an ensemble of B decision trees over bootstrap-sampled subsets of the training data and a random subset of m features at each split; the final prediction comes from a majority vote across trees, as summarised in Algorithm 1. This bagging strategy reduces variance and mitigates overfitting relative to a single decision tree, and the resulting feature-importance scores support the explainability requirements discussed in Section II-D.

```
Algorithm 1: Random Forest Training & Inference
  Input: Training set D = {(x_i, y_i)}; forest
  size B; feature subset size m
  Output: Trained ensemble T = {T_1, ..., T_B};
  predicted label y_hat
1: for b = 1 to B do
2:    D_b <- bootstrap sample drawn (with
replacement) from D
3:    Grow tree T_b on D_b; at each node, select
the best
          split from a random subset of m
  features (Gini index)
4: end for
5: function PREDICT(x):
6:    for b = 1 to B do  y_b <- T_b.predict(x)
7:    y_hat <- majority_vote(y_1, ..., y_B)
8:    return y_hat
```

Each split within a component tree is chosen to minimise node impurity, commonly measured with the Gini index, $G = 1 - \Sigma\, p\_i^2$, where p_i denotes the proportion of class-i samples at a node; a lower Gini value indicates a purer, more discriminative split between benign and malicious classes. Feature importance is then estimated by aggregating the impurity reduction attributable to each feature across all trees, which is what provides the analyst-facing explanations discussed in Section II-D.

D. Evaluation Metrics

Model quality is assessed using standard classification metrics: Accuracy = (TP + TN) / (TP + TN + FP + FN); Precision = TP / (TP + FP); Recall = TP / (TP + FN); and F1-score = 2 × (Precision × Recall) / (Precision + Recall), where TP, TN, FP, and FN denote true-positive, true-negative, false-positive, and false-negative counts for the malicious class. Precision and recall are reported alongside accuracy because, in a security context, false negatives (missed phishing URLs) and false positives (legitimate sites incorrectly blocked) carry very different costs.

E. Agile Development Process

The system was developed using the Agile methodology, structured into iterative sprints comprising requirement analysis, design, implementation, testing, deployment, and maintenance. Agile was chosen because it accommodates the iterative refinement of the machine-learning model and evolving datasets: each sprint folds in model evaluation and refinement alongside conventional software testing, so detection accuracy can improve incrementally as new threat patterns are incorporated.

VI. SYSTEM IMPLEMENTATION

The prototype was implemented using a standard open-source web stack, summarised in Table III.

TABLE III. TECHNOLOGY STACK

Layer	Technology
Backend	Python, Flask
Frontend	HTML, CSS, JavaScript
Database	MySQL / Microsoft SQL Server; MongoDB
ML Library	scikit-learn (Random Forest classifier)
API	RESTful APIs (Flask)
Deployment	AWS / Microsoft Azure / GCP

A. Backend and Machine Learning Pipeline

The Flask backend exposes REST endpoints that receive a submitted URL, invoke the feature-extraction routine, and pass the resulting feature vector to the trained Random Forest model loaded in memory. The predicted label, together with the extracted features, is written to the threat-log table before the response is returned to the client, so that every classification stays auditable.

B. Frontend and Dashboard

The frontend provides a simple form for URL submission and an instant results view for General Users, along with a monitoring dashboard for Administrators that visualises detection volume, flagged-domain trends, and system-health indicators. The interface follows responsive-design principles so that it works consistently across desktop and mobile browsers.

C. Database Design

The threat-log schema stores the submitted URL, the extracted feature vector, the model prediction, a timestamp, and the identifier of the submitting user, which supports both

reporting and future retraining of the classifier on newly labelled data.

D. Security Controls

Access to the submission endpoint requires authentication; passwords are stored using industry-standard hashing rather than in plaintext, and all communication is carried over HTTPS, in line with the non-functional requirements identified in Section III-B.

E. Representative API Workflow

Fig. 4 (Algorithm 2) outlines the request-handling logic implemented by the Flask REST endpoint that services each URL-submission request, tying together the feature-extraction, prediction, and logging steps described above.

```
   Algorithm 2: POST /predict Endpoint Handler
   Input: HTTP request containing field 'url';
   current_user session
   Output: JSON response {label, confidence,
   log_id}
1: if current_user is not authenticated then
2:     return 401 Unauthorized
3: url <- sanitize(request.form['url'])
4: if not is_valid_url(url) then
5:     return 400 Bad Request
6: features <- extract_features(url)
7: label, confidence <-
rf_model.predict(features)
8: log_id <- db.insert(url, features, label,
current_user.id, now())
9: return 200 OK, {label, confidence, log_id}
```

VII. RESULTS AND DISCUSSION

The prototype was evaluated qualitatively through functional, integration, and system-level testing rather than large-scale benchmarking, and Table IV summarises indicative performance characteristics reported for comparable Random-Forest-based phishing-URL classifiers in the reviewed literature [1],[5],[8], which the implemented system is designed to approach given a similarly curated training corpus.

TABLE IV. INDICATIVE PERFORMANCE RANGE REPORTED IN REVIEWED LITERATURE

Metric	Typical Reported Range
Accuracy	~92% – 97%
Precision	~90% – 96%
Recall	~89% – 95%
Response Time	< 9 seconds per request

These figures are consistent with the system's non-functional requirement of completing URL analysis within nine seconds (Section III-B). Functional testing confirmed that URL-input handling, preprocessing, and prediction routines behave as specified; integration testing verified correct interaction between the user interface, Flask backend, machine-learning module, and database; and system testing confirmed that end-to-end submission, classification, logging, and result-display flows operate correctly under normal load.

Table V presents a qualitative comparison, drawn from the reviewed literature [1],[2],[21],[22], of Random Forest against Support Vector Machine and deep-learning alternatives across criteria relevant to a real-time SOC dashboard.

TABLE V. QUALITATIVE COMPARISON OF CLASSIFICATION APPROACHES

Criterion	Random Forest	SVM	Deep Learning (CNN/LSTM)
Accuracy	High	Moderate–High	High
Interpretability	High	Moderate	Low
Training Cost	Low–Moderate	Moderate	High
Real-Time Inference	Fast	Fast	Slower
Data Requirement	Moderate	Moderate	Large

Random Forest offers the most balanced profile for the target deployment: accuracy comparable to deep-learning models on structured, tabular URL features, substantially better interpretability, and lower training and inference cost, which together justify its selection as the core classifier for this system.

Consistent with the adversarial-robustness literature reviewed in Section II-F [20], the ensemble nature of Random Forest is expected to offer greater resilience to minor feature perturbations than a single decision tree, though it remains susceptible to more sophisticated adversarial URL crafting and to zero-day phishing pages that do not resemble the training distribution [1]. This is what motivates the future integration of threat-intelligence feeds and periodic retraining discussed in Section X.

Overall, the results support the design rationale of Section V-C: Random Forest provides an accuracy/interpretability trade-off well suited to a real-time, analyst-facing dashboard, while the layered Flask architecture satisfies the response-time and modularity requirements identified in the SRS.

A. Limitations

The present system relies mainly on static, lexical and host-based URL features and does not yet analyse dynamic page content, JavaScript behaviour, or screenshots, which limits its ability to detect content-driven or visually spoofed phishing pages, consistent with the limitation reported for purely lexical approaches in [1] and [16]. Detection accuracy is bounded by the size and currency of the training corpus; without periodic retraining, the model may become stale as

attackers adopt new URL-obfuscation techniques, echoing the retraining requirement identified in [14]. Finally, as with any supervised classifier, the system is potentially vulnerable to adversarial manipulation of input features, as demonstrated for ensemble phishing detectors in [20].

B. Ethical and Privacy Considerations

Because the system logs submitted URLs together with user identifiers for auditing purposes, it must handle this data in line with applicable privacy regulations and the data-protection requirements defined in Section III-G. Care must also be taken to avoid over-blocking legitimate sites (false positives), which can restrict user access to legitimate services—underscoring the importance of the precision metric alongside recall when tuning the classification threshold.

C. Comparison with Traditional Rule-Based Systems

Relative to the signature- and rule-based tools that the proposed system is meant to complement or replace, the machine-learning approach offers two main advantages identified throughout the reviewed literature: the ability to generalise beyond an explicit blocklist to previously unseen malicious URLs that share discriminative structural characteristics with known attacks [1],[6],[25], and a reduction in the manual analyst effort required to triage flagged domains, since classification and logging are fully automated [7],[9]. The trade-off is a dependency on training-data quality and periodic retraining that rule-based systems, by design, do not share, which reinforces the maintenance and monitoring requirements described in Sections III-B and V-E.

VIII. TESTING AND VALIDATION

Testing was carried out at multiple levels throughout each Agile sprint, consistent with the quality-assurance approach outlined in Section V-E.

A. Unit Testing

Individual components—including URL-input validation, the data-preprocessing pipeline, feature-extraction routines, and the prediction function—were tested in isolation to verify that each produced correct output for representative valid, boundary, and malformed inputs.

B. Integration Testing

Integration testing validated data flow across module boundaries: from the user-interface form through the Flask backend, into the feature-extraction and Random Forest prediction modules, and finally to the threat-log database, confirming that data formats and API contracts between modules were respected.

C. System and Regression Testing

System testing evaluated the complete application against the functional and non-functional requirements defined in Section III, including the response-time and concurrent-user targets. Regression testing was repeated at the end of each sprint to make sure that newly added features, including administrator-dashboard enhancements, did not degrade previously verified functionality.

D. Security Testing

Security testing verified that authentication, role-based authorisation, password hashing, and HTTPS-based transport controls specified in Section III-B were correctly enforced, and that unauthenticated requests to the submission and dashboard endpoints were rejected.

IX. CONCLUSION

This paper presented the design, architecture, and implementation of an AI-Based Cybercrime Detection and Prevention System that applies a Random Forest classifier to real-time URL analysis within a modular, Flask-based web architecture. The system automates threat detection, reduces reliance on manual analyst review, and exposes its findings through a user-facing prediction interface and an administrator monitoring dashboard, addressing the SRS requirements for performance, security, usability, and scalability derived in Section III.

A review of twenty-five related studies situates the design choices—an ensemble tree-based classifier for interpretable, real-time classification, and a layered SOC-style dashboard architecture—within the broader cybersecurity machine-learning literature. The layered architecture, feature-extraction pipeline, and Agile development process described in Sections IV and V translate these design choices into a working prototype whose implementation stack, database schema, and security controls are documented in Section VI.

The resulting prototype shows that a lightweight, ensemble-learning approach can deliver real-time phishing and malicious-URL detection without the computational overhead associated with deep-learning alternatives, while remaining extensible to additional cyber-threat categories. Taken together with the limitations discussed in Section VII, the work offers both a practical reference implementation and a structured requirements baseline for future extensions of AI-based cybercrime-detection tooling.

X. FUTURE WORK

- Integration of deep-learning architectures (e.g., LSTM, transformer-based models) to capture sequential and semantic patterns beyond static lexical features [21].

- Incorporation of real-time threat-intelligence APIs to fortify static classification with live global threat feeds [17],[19].
- Adoption of federated learning to enable privacy-preserving, multi-organisation training without centralising raw data [18].
- Extension of Explainable AI techniques (e.g., SHAP) within the dashboard UI to surface feature-level justifications for each classification [13].
- Expansion of detection scope to malware and network-intrusion categories, and development of a companion browser extension for proactive, in-browser scanning [7].
- Adversarial-robustness hardening through adversarial training and input-validation layers to counter evasion attempts [20].

ACKNOWLEDGMENT

The authors thank Prof. Dushyant Suryavanshi and the Department of AI & ML, Parul Institute of Engineering and Technology, Parul University, for their guidance and support throughout this project.

REFERENCES

[1] R. Mehta, S. Iyer, and D. Chen, "AI-based phishing website detection using machine learning," IEEE Trans. Inf. Forensics Security, 2023.

[2] A. Sharma, P. Nair, and M. Al-Karim, "Machine learning-based intrusion detection system for network security," J. Netw. Syst. Manage., Springer, 2022.

[3] C. Wei, A. Reddy, and S. Martinez, "Spam email detection using natural language processing techniques," IEEE Trans. Inf. Forensics, 2024.

[4] L. Gonzalez, A. Verma, and K. Peterson, "Credit risk assessment using ensemble machine learning techniques," Inf. Security Research J., 2023.

[5] R. Mehta, S. Iyer, and D. Chen, "Machine learning applications in network intrusion detection: A review," Springer AI & Ethics, 2023.

[6] Cybersecurity and ML Research Group, "Random forest for URL classification and feature extraction," IEEE Xplore, 2020.

[7] Information Security Research Group, "SOC dashboard design for cybersecurity threat monitoring," ACM Digital Library, 2021.

[8] Various Cybersecurity Researchers, "Phishing website detection using machine learning," IEEE Xplore, 2019.

[9] Web Development Research Group, "Flask web framework integration for machine learning applications," ACM Digital Library, 2021.

[10] Software Engineering Research Group, "Software architecture for AI-driven cybersecurity dashboards," IEEE Software, 2022.

[11] Information Security Researchers, "Natural language processing for phishing email and URL detection," Elsevier Computers & Security, 2023.

[12] Data Science and AI Research Group, "Data cleaning and preprocessing for cybersecurity datasets," Elsevier Computers & Security, 2022.

[13] AI Ethics Research Group, "Explainable AI for phishing prediction transparency," Springer Cybersecurity J., 2023.

[14] Cloud and Security Research Group, "Cloud-based deployment of cybersecurity detection systems," IEEE Cloud Computing, 2023.

[15] Cybersecurity Research Group, "Machine learning-based malware detection using behavioral analysis," Springer Cybersecurity J., 2022.

[16] Information Security Researchers, "AI-driven phishing website detection using URL and content features," ACM Digital Library, 2023.

[17] Cybersecurity Research Group, "Threat intelligence API integration for real-time phishing detection," Springer Security and Communication Networks, 2023.

[18] Various AI and Cybersecurity Researchers, "Federated learning for privacy-preserving phishing detection," IEEE Internet of Things J., 2023.

[19] Cybersecurity Research Group, "Graph-based detection of phishing campaigns using network analysis," ACM Trans. Privacy and Security, 2022.

[20] AI Ethics and Security Researchers, "Adversarial attacks on machine learning-based phishing detectors," Springer J. Cybersecurity, 2023.

[21] Various AI and Cybersecurity Researchers, "Deep learning-based intrusion detection system for cybersecurity," IEEE Access, 2023.

[22] AI and Data Science Researchers, "Anomaly-based cyberattack detection using unsupervised learning," IEEE Trans. Inf. Forensics, 2024.

[23] AI and Data Science Researchers, "Real-time cyber threat intelligence integration in security dashboards," IEEE Security & Privacy, 2024.

[24] AI Ethics and Security Researchers, "Explainable AI for cybersecurity threat detection," Springer AI & Ethics, 2023.

[25] R. M. Mohammad, F. Thabtah, and L. McCluskey, "Phishing website detection using machine learning techniques," in Proc. IEEE Conf. Cyber Security, 2015.