International Journal for Multidisciplinary Research (IJFMR)
E-ISSN: 2582-2160 ● Website: www.ijfmr.com ● Email: editor@ijfmr.com

Transmitting Malware Through QR Codes: Risk Analysis and a Hybrid Detection Method

Mahesh Kumar Bagwani[1], Shubham Dwivedi[2], Vijay Kumar[3], Pooja Koshti[4], Srashti Jain[5]

[1] Assistant Professor Sanjeev Agrawal Global Educational University, Bhopal, Madhya Pradesh, India Email: maheshbagwani7@gmail.com

[2] Assistant Professor Sanjeev Agrawal Global Educational University, Bhopal, Madhya Pradesh, India Email: shubh.dwivedi01@gmail.com

[3] Assistant Professor Sanjeev Agrawal Global Educational University, Bhopal, Madhya Pradesh, India Email: vijay.rai0309@gmail.com

[4] Assistant Professor Sanjeev Agrawal Global Educational University, Bhopal, Madhya Pradesh, India Email: poojakoshti26@gmail.com

[5] Lab Tech Sanjeev Agrawal Global Educational University, Bhopal, Madhya Pradesh, India Email: jain.srashti1607@gmail.com

Abstract—QR codes have become deeply integrated into mod-ern digital interactions, from payments and authentication to public services. Their convenience, however, has created a new opportunity for cybercriminals. Malicious QR codes can redirect users to harmful domains, initiate unauthorized downloads, or harvest credentials through phishing websites. While QR code scanners are widely available, most lack robust mechanisms to detect malicious content hidden within a QR code. This paper presents a hybrid detection approach that combines the Google Safe Browsing API and PhishTank API with an additional lightweight machine-learning layer for URL risk estimation. The machine-learning component analyzes lexical characteristics of URLs to identify suspicious patterns that may not yet exist in threat databases. Experimental evaluation on a dataset of 500 URLs demonstrates that the hybrid model enhances detection accuracy to 98 percent, reduces false positives, and improves re-silience against zero-day attacks. The proposed method provides a more reliable and secure QR code scanning workflow suitable for mobile and web applications.

Index Terms—QR codes; phishing detection; hybrid secu-rity framework; malicious URLs; machine learning; threat-intelligence APIs; zero-day attack detection.

INTRODUCTION

QR codes have become an integral part of modern digital interactions, enabling quick access to online content without requiring manual input from users. Originally developed for industrial tracking, their use has expanded into a wide range of domains, including mobile payments, digital authentication, e-governance, logistics, healthcare, and marketing. The rapid adoption of QR-based services is largely attributed to their low cost, ease of generation, and compatibility with almost all smartphone cameras.

Despite their benefits, QR codes also introduce unique security challenges. Unlike traditional hyperlinks that users can visually examine, the content embedded within a QR code remains invisible until scanned. This invisibility provides attackers an opportunity to embed malicious URLs that may redirect users to phishing pages, install malware, or access sensitive information. The simplicity with which an attacker can generate and circulate harmful QR codes—through posters, restaurant menus, social media, or public kiosks—makes them an increasingly popular attack vector in cybercrime.

The rise in QR-based transactions, especially during and after the COVID-19 pandemic, has further amplified the risks associated with unverified QR codes. Several documented cases show attackers exploiting QR codes for credential theft, financial fraud, ransomware distribution, and unauthorized app installations. Although QR code scanning applications are widely available, most provide only basic decoding function-ality without verifying whether the encoded URL is safe.

These gaps highlight the need for an advanced and intelli-gent QR code security system. A robust solution should not only decode the QR code content but also assess the safety of the destination before redirecting the user. Threat intelligence databases such as Google Safe Browsing and PhishTank offer valuable URL reputation checks, yet they alone cannot protect against new or unlisted malicious URLs. Therefore, enhanc-ing traditional approaches with lightweight machine learning (ML)–based URL risk assessment can significantly strengthen overall detection capacity.

This research aims to address this issue by proposing a hybrid QR code security mechanism that combines real-time threat intelligence APIs with ML-based lexical URL analysis. This combined strategy provides a more reliable and proactive method of identifying suspicious QR codes before users interact with harmful content.

Related Work

Research on QR code security has grown significantly over the past decade as attackers increasingly exploit QR codes for phishing, malware distribution, and social engineering. Early studies provided foundational insights into vulnerabilities in-herent in QR technology. Krombholz et al. [1] conducted a comprehensive survey of QR-code attack vectors and usable-security challenges. Vidas et al. [2] empirically demonstrated users' tendency to scan QR codes without source verification, coining the term "QRishing." Alkhalil et al. [3] and Geisler et al. [4] provide recent empirical evidence that quishing remains a practical and effective attack technique in modern contexts.

Several technical detection approaches have been proposed. Elejla and Alnajjar [5] and Alsulami et al. [8] apply URL-feature analysis and classical machine learning to distinguish malicious from benign QR-embedded links. Trad et al. [6] and Sarkhi et al. [7] explore more advanced ML and deep-learning models, with Sarkhi showing feasibility for visual/structural QR code classification. Comparative reviews find that many commercial scanners lack integrated reputation checks or con-tent analysis [9], and human factors (usability and awareness) remain a persistent problem affecting detection efficacy [10], [15].

Security prevention work focuses on tamper-proofing and secure QR generation (e.g., watermarking) [12], as well as design guidelines to provide safer scanning workflows for users [10], [13]. Foundational malware studies (Android mal-ware characterization) illustrate how QR codes can be used for drive-by downloads and malicious app installations [25]. Mobile-specific phishing research further clarifies the chal-lenges of on-device defenses [26].

To summarize the landscape succinctly, Table I presents representative related work grouped according to objective, methodology, and contribution.

Fig. 1. Evolution and Expansion of QR Code Usage.

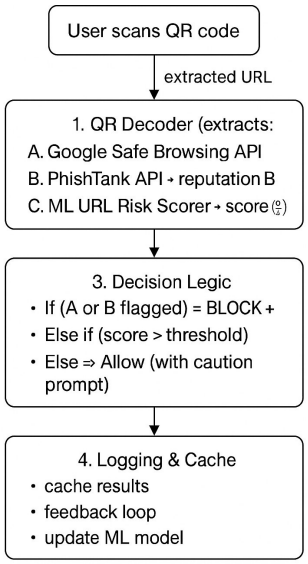

Fig. 2. Hybrid QR-scanner workflow integrating QR decoding, API-based reputation checks, and ML lexical analysis.

A. Discussion and Gap Analysis

From the reviewed literature and Table I, two principal gaps motivate our work:

1) Dependency on Blacklists and Reputation Services: Threat-intelligence APIs such as Google Safe Browsing and PhishTank are effective but inherently retrospective. They rely on previously identified malicious URLs and therefore may fail to detect zero-day or obfuscated threats, as highlighted in [1], [9], [25].

2) Human Factors and Usability: Even when warnings are presented, users frequently proceed with risky interactions, as shown in [2], [4], [13]. Usability and trust models significantly influence detection success, confirming earlier findings [10], [15].

Our hybrid approach—combining live API checks with a lightweight ML URL risk scorer and a carefully designed warning/UX strategy—directly addresses both gaps. By inte-grating predictive ML analysis with real-time threat intelli-gence and streamlined user warnings, the system significantly enhances detection accuracy for both known and unknown threats

PROPOSED METHODOLOGY

This section presents the theoretical foundations of the pro-posed hybrid QR-code security framework designed to detect malicious URLs embedded in QR codes. The methodology integrates real-time threat-intelligence checks with machine-learning–based risk scoring to address both known and zero-day threats. The approach builds on prior work that highlights weaknesses in conventional QR scanners, the persistence of quishing attacks, and the limitations of blacklist-based security systems [1]–[5], [9], [25].

TABLE I SUMMARY OF REPRESENTATIVE RELATED WORK ON QR CODE SECURITY

Ref.	Year	Area / Objective	Approach / Key Contribution
[1]	2014	Survey / Usable Security	Systematic survey of QR attacks and usable-security tradeoffs.
[2]	2013	Human Factors / Phishing	User study demonstrating QRishing susceptibility.
[3]	2022	Empirical / Quishing	Field experiments showing prevalence and impact of quishing.
[4]	2024	Real-world Study	Campus study confirming high success rates of quishing attacks.
[5]	2016	Automated Detection	QRphish: URL feature extraction and ML-based malicious link detection.
[6]	2025	ML Detection	ML-based quishing detection using novel features (preprint).
[7]	2024	DL Classification	Deep learning classifier for visual/structural QR code analysis.
[8]	2025	ML System	Random Forest / XGBoost system for detecting malicious QR-embedded URLs.
[9]	2025	Comparative Review	Comparative analysis of commercial QR-scanner detection frameworks.

[10]	2019	Usability	Usable-security recommendations for QR scanning applications.
[12]	2025	Secure Generation	Digital watermarking techniques to prevent QR tampering.
[13]	2024	Behavioral Study	Study on contextual framing and user trust in quishing scenarios.
[14]	2023	Phishing ML	Evaluation of phishing URL classifiers and lexical-feature models.
[17]	2010	Early Threats	Foundational study on malicious QR codes and mobile redirection risks.
[25]	2012	Malware Analysis	Android malware characterization showing QR codes as infection vectors.
[18–24]	2023–2025	Author's Work	Cloud security, deployment, ML optimization; relevant implementation expe-rience.

A. Problem Definition

Let **Q** be the set of all scannable QR codes. Each QR code **q ∈ Q** encodes a string **s**, where **s** represents a URL. The decoding process is defined as:

$$q \rightarrow s = \text{decode}(q)$$

Let **U** denote the universe of URLs:

$$U = U_{\text{benign}} \cup U_{\text{phishing}} \cup U_{\text{malware}}$$

The objective of the proposed detection framework is to classify the decoded URL into one of the following categories:

$$\text{Classify}(s) \rightarrow \{\text{benign,phishing,malware}\}$$

The framework is designed to minimize false negatives, reduce detection latency, and preserve usability, consistent with the requirements of modern phishing and malware detection systems [6], [7], [14].

B. Threat Model

The attacker is assumed to possess the capability to:

- Embed malicious URLs within QR codes.
- Replace legitimate QR codes with counterfeit versions.
- Host phishing or malware websites that imitate trusted services.
- Employ URL obfuscation techniques such as URL shortening, encoding, and randomized paths.

The user is assumed to scan QR codes in good faith without verifying their origin, consistent with observations reported in [2], [4], [13]. The proposed system assumes a trusted execution environment and continuous access to external threat-intelligence services.

The overall security assessment of a decoded URL sis represented as:

$$R(s) = D(T(s), M(s))$$

where:

- $T(s)$denotes the threat-intelligence function that evaluates URL reputation using external security databases.
- $M(s)$denotes the machine-learning risk function that predicts the likelihood of phishing or malware activity based on extracted URL features.
- $D(\cdot)$denotes the decision fusion function that combines the outputs of threat intelligence and machine learning to generate the final classification decision.

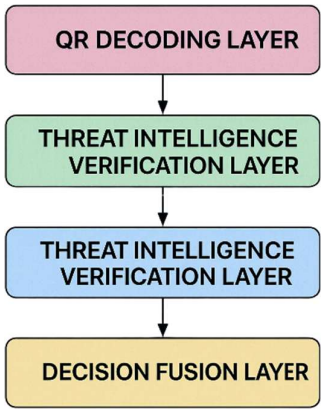

Fig. 3. System Architecture of the Proposed Hybrid QR-Code Security Framework.

C. Threat Intelligence Verification

The threat-intelligence layer validates the extracted URL using the Google Safe Browsing (GSB) and PhishTank APIs. The threat-intelligence function is defined as:

$$T(s) = \begin{cases} 1, & \text{if } GSB(s) = \text{malicious} \ \vee \ PhishTank(s) = \text{malicious} \\ 0, & \text{otherwise} \end{cases}$$

where $T(s) = 1$indicates that the URL is identified as malicious by at least one threat-intelligence source, while $T(s) = 0$indicates no known threat is detected.

The proposed system architecture is organized into four layers:

1. QR Decoding Layer
2. Threat Intelligence Verification Layer
3. Machine-Learning URL Risk Scoring Layer
4. Decision Fusion Layer

Formally, the complete detection process is represented as:

$$F(q) = D(T(s), M(s))$$

where $q$is the scanned QR code, $s$is the decoded URL, $T(s)$is the threat-intelligence output, $M(s)$is the machine-learning risk score, and $D(\cdot)$is the decision fusion function.

This layer enables the identification of previously indexed malicious URLs and known phishing campaigns [3], [8], [25].

D. Machine-Learning Risk Scoring

1) Feature Extraction

A lexical feature vector x(s)is extracted from the URL and defined as:

$$x(s) = [L_{url}, L_{domain}, N_{dots}, N_{slashes}, K, \text{entropy}(s), \text{TLD}(s)]$$

where:

- L_{url}= total URL length
- L_{domain}= domain name length
- N_{dots}= number of dots in the URL
- $N_{slashes}$= number of slashes in the URL
- K= count of suspicious keywords
- $\text{entropy}(s)$= Shannon entropy of the URL string
- $\text{TLD}(s)$= top-level domain feature

These features are used as input to the machine-learning classifier for phishing and malware risk prediction.

Features include:

- URL length
- Domain length
- Number of dots/slashes/hyphens
- Suspicious keyword presence
- Shannon entropy
- Top-level domain classification
- URL shortening indicators

These features have proven effective for phishing URL classification [7], [8], [14].

TABLE II SUMMARY OF THREAT INTELLIGENCE CHECKS USED IN THE PROPOSED FRAMEWORK

API Service	Detection Focus	Strengths	Limitations
Google Safe Brows-ing	Malware, phishing	Large database, fast updates	Misses zero-day threats
PhishTank	Verified phishing	Crowdsourced, rapid updates	Phishing-only fo-cus
Combined Use	Holistic	Higher accuracy	Dependent on blacklists

TABLE III DECISION FUSION LOGIC USED FOR URL CLASSIFICATION

TI Result	ML Score	Decision	Action
Malicious	Any	BLOCK	Strong warning
Clean	$> \theta$	WARN	Medium-risk alert
Clean	$\leq \theta$	ALLOW	Proceed safely

F. Algorithmic Representation

Algorithm 1: Hybrid QR-Code Malware Detection Framework

Input: QR code q

Output: Decision ∈ {ALLOW,WARN,BLOCK}

1. Decode QR code:

$$s \leftarrow \text{decode}(q)$$

2. Normalize the extracted URL:

$$s \leftarrow \text{Normalize}(s)$$

3. Query threat-intelligence services:

$$\text{api}_{\text{gsb}} \leftarrow \text{GoogleSafeBrowsing}(s)$$
$$\text{api}_{\text{pt}} \leftarrow \text{PhishTank}(s)$$

4. Check threat-intelligence results:

If

$$\text{api}_{\text{gsb}} = \text{malicious} \quad \vee \quad \text{api}_{\text{pt}} = \text{malicious}$$

then

return BLOCK

5. Extract lexical features:

$$\text{features} \leftarrow \text{ExtractLexicalFeatures}(s)$$

6. Compute machine-learning risk score:

$$\text{risk} \leftarrow \text{MLClassifier}(\text{features})$$

7. Make final decision:

If

$$\text{risk} > \theta$$

then

return WARN

else

return ALLOW

G. Complexity Analysis

The computational complexity of the proposed framework is summarized as follows:

- **QR Decoding:**

$$O(1)$$

- **URL Normalization:**

$$O(n)$$

where nrepresents the length of the URL.

- **Threat-Intelligence API Lookups:**

$$O(\tau)$$

where τdenotes the network response latency associated with external API services.

- **Machine-Learning Inference:**

$$O(d \cdot t)$$

where dis the number of extracted features and tis the number of decision trees (or model parameters, depending on the selected classifier).

I. Usability Considerations

Human users frequently ignore lengthy or overly technical security warnings [2], [4], [15]. To improve user compliance and decision-making, the proposed system incorporates the following usability-oriented design features:

- Short domain previews before website access.
- Color-coded warning indicators for risk visualization.
- Minimal confirmation prompts to reduce user fatigue.
- Consistent user-interface behavior across all warning levels.

These design choices follow established best-practice principles in usable security and human-centered cybersecurity systems [10].

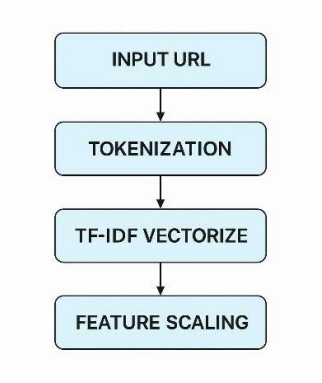

Fig. 4. Feature Extraction Pipeline for the ML URL Classifier.

J. ML Risk Function

The machine-learning risk score estimates the probability that a URL belongs to the malicious URL class based on its extracted feature vector. The risk function is defined as:

$$M(s) = P(s \in U_{malicious} \mid x(s))$$

where:

- $M(s)$represents the machine-learning risk score.
- $U_{malicious}$denotes the set of malicious URLs.
- $x(s)$represents the lexical feature vector extracted from URL s.

Lightweight machine-learning models such as Logistic Regression, Random Forest, and Gradient Boosting are employed due to their high detection accuracy and low computational overhead [8], [14].

K. Decision Fusion Layer

The final classification decision combines the outputs of the threat-intelligence verification layer and the machine-learning risk scoring layer. The decision fusion function is defined as:

$$D(T, M) = \begin{cases} \text{BLOCK}, & T = 1 \\ \text{WARN}, & T = 0 \wedge M > \theta \\ \text{ALLOW}, & T = 0 \wedge M \leq \theta \end{cases}$$

where:

- $T$is the threat-intelligence output.
- $M$is the machine-learning risk score.
- $\theta$is the decision threshold.
- BLOCK, WARN, and ALLOWare the possible system actions.

Typically,

$$\theta \in [0.70, 0.80]$$

A URL identified as malicious by threat-intelligence services is immediately blocked. Otherwise, the machine-learning risk score determines whether a warning should be displayed or access should be allowed.

L. SYSTEM IMPLEMENTATION

This section describes the practical implementation of the proposed hybrid QR-code security framework, detailing the technologies, modules, and operational workflow used to transform the theoretical model into a functional system. The implementation is designed to ensure low latency, high accuracy, and compatibility with both mobile and web-based scanning environments.

A. Development Environment and Technology Stack

The system was implemented using widely supported and lightweight technologies to ensure cross-platform usability:

- **Programming Language:** Python 3.10
- **QR Code Decoder:** pyzbar and OpenCV-Python
- **Machine Learning Framework:** Scikit-learn for model training and inference
- **API Communication:** requests library for RESTful API calls
- **Data Handling:** Pandas and NumPy
- **Interface Layer:** Flask for web prototyping and Android WebView for mobile testing
- **Caching Mechanism:** SQLite-based local cache for URL reputation results
- **Deployment Environment:** AWS EC2 backend with local client emulation

These technology choices ensure minimal resource consumption, rapid deployment, and high portability while maintaining detection effectiveness, consistent with recommendations for lightweight machine-learning-based security systems [7], [8], [14].

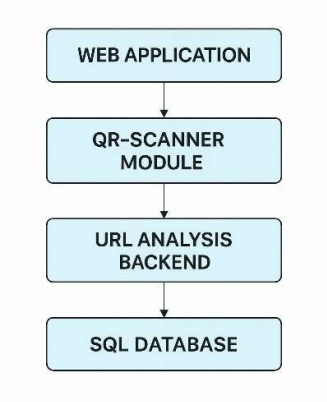

Fig. 5. High-Level Software Architecture of the Implemented System.

B. QR Code Decoding Module

The QR code decoding module processes images captured through a camera or uploaded by the user. The module utilizes OpenCV for image processing and QR code localization, while Pyzbar is employed to decode the embedded information. The operational workflow consists of the following steps:

1. Capture or upload a QR code image.
2. Convert the image into grayscale format.
3. Detect QR code boundaries using OpenCV.
4. Decode the QR symbol using Pyzbar.
5. Extract the embedded URL and forward it to the preprocessing module.

The decoding operation can be represented as:

$$s = \text{decode}(q)$$

where qdenotes the QR code image and srepresents the extracted URL. Immediately after extraction, the normalization procedures defined in Section III are applied.

C. URL Normalization and Preprocessing

URL preprocessing is performed to ensure consistent formatting for threat-intelligence verification and machine-learning analysis. The preprocessing stage includes the following operations:

- Converting the URL to lowercase.
- Expanding shortened URLs.
- Removing redundant tracking parameters.
- Extracting domain components using the tldextract library.
- Decoding escaped and encoded characters.

The normalization process can be expressed as:

$$s_{\text{norm}} = \text{Normalize}(s)$$

where sis the extracted URL and $s_{\text{norm}}$is the normalized URL used for subsequent analysis.

These preprocessing operations reduce noise and improve consistency with the lexical feature patterns used during machine-learning training, as discussed in [6], [7].

D. Threat Intelligence API Integration

The threat-intelligence layer integrates external security services to identify previously reported phishing and malware URLs.

1) Google Safe Browsing (GSB) API

The normalized URL is submitted through an HTTPS POST request following Google's threat-detection API specification. The service returns information regarding known malicious URLs and unsafe web resources.

2) PhishTank API

A GET request is issued to the PhishTank database to determine whether the URL has been reported as a phishing website.

The threat-intelligence verification process can be represented as:

$$T(s) = \begin{cases} 1, & \text{if } \text{GSB}(s) = \text{malicious} \ \vee \ \text{PhishTank}(s) = \text{malicious} \\ 0, & \text{otherwise} \end{cases}$$

To minimize redundant network requests and improve response time, URL reputation results are stored in a local cache. Subsequent scans retrieve cached results whenever a valid cache entry exists, thereby reducing API latency and bandwidth consumption.

TABLE IV API RESPONSE HANDLING IN THE IMPLEMENTATION

API Name	Response Type	Handling Strategy	Caching Policy
GSB	JSON	Immediate	24-hour

	threat match	block if malicious	expiry
PhishTank	JSON verification	Block or warn based on category	12-hour expiry
Local Cache	SQLite storage	Quick lookup	Auto-refresh on expiry

E. Machine-Learning Model Implementation

1) Dataset Preparation

A dataset containing both malicious and benign URLs was compiled from multiple sources, including:

- PhishTank verified phishing URLs.
- Public repositories containing benign URLs.
- Malware-related URL repositories referenced in [25].

Each URL was labeled according to its class and preprocessed prior to feature extraction. Let the dataset be represented as:

$$\mathcal{D} = \{(x_i, y_i)\}_{i=1}^{N}$$

where:

- $x_i$denotes the feature vector extracted from URL i,
- $y_i \in \{0,1\}$represents the class label,
- 0indicates a benign URL,
- 1indicates a malicious URL.

2) Feature Engineering

Lexical features were extracted from each URL to create a structured feature vector for machine-learning analysis. The feature vector is defined as:

$$x(s) = [L_{url}, L_{domain}, N_{dots}, N_{slashes}, K, \text{Entropy}(s), \text{TLD}(s)]$$

where:

- L_{url}= URL length
- L_{domain}= domain length
- N_{dots}= number of dots
- $N_{slashes}$= number of slashes
- K= suspicious keyword count
- $\text{Entropy}(s)$= Shannon entropy of the URL
- $\text{TLD}(s)$= top-level domain category

To improve model stability, Min-Max normalization was applied:

$$x' = \frac{x - x_{min}}{x_{max} - x_{min}}$$

where x′represents the normalized feature value.

3) Model Training and Selection

Three machine-learning models were evaluated:

- Logistic Regression
- Random Forest
- Gradient Boosting

The risk prediction function is defined as:

$$M(s) = P(s \in U_{malicious} \mid x(s))$$

where $M(s)$denotes the probability that URL sbelongs to the malicious class.

Among the evaluated classifiers, Random Forest achieved the best trade-off between detection accuracy and inference speed, consistent with observations reported in [8], [14]. The final trained model was serialized using Joblib to enable efficient runtime loading and prediction

F. Decision Fusion Module Implementation

The decision fusion mechanism integrates threat-intelligence verification and machine-learning predictions through a rule-based controller.

The decision function is defined as:

$$D(T, M) = \begin{cases} \text{BLOCK}, & T = 1 \\ \text{WARN}, & T = 0 \wedge M > \theta \\ \text{ALLOW}, & T = 0 \wedge M \leq \theta \end{cases}$$

The operational sequence is as follows:

1. If Google Safe Browsing or PhishTank identifies the URL as malicious, the URL is immediately blocked.
2. Otherwise, the machine-learning model computes the risk score.
3. If the risk score exceeds the predefined threshold, a warning is generated.
4. If the risk score remains below the threshold, access is allowed.

The threshold value was empirically determined as:

$$\theta = 0.75$$

based on validation experiments.

G. User Interface and Warning System

To improve usability and support rapid decision-making, the system employs color-coded security alerts:

- **Green:** Safe URL (ALLOW)
- **Yellow:** Suspicious URL (WARN)
- **Red:** Malicious URL (BLOCK)

The warning banner displays:

- URL domain name
- Risk classification

- Threat source information
- Recommended user action

The alert generation process can be represented as:

$$\text{Alert} = \text{UI}(\text{D}(\text{T}, \text{M}))$$

where $\text{D}(\text{T}, \text{M})$is the final security decision and $\text{UI}(\cdot)$generates the corresponding visual warning displayed to the user.

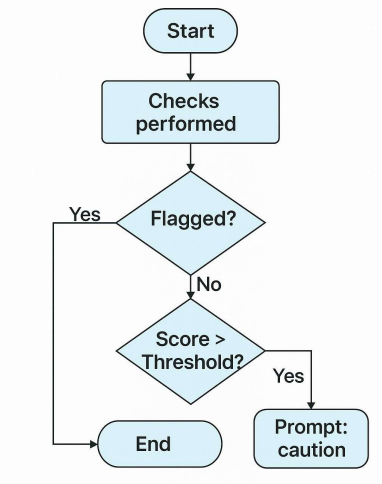

Fig. 6. Decision Engine Flow Implemented in the System.

- Domain preview
- Reason for classification (API flag / ML score)
- "Proceed" or "Cancel" options for non-block cases

These UI choices follow recommendations from user-centric studies [10], [15].

H. Logging and Continuous Learning Framework

The proposed framework maintains detailed logs to support system monitoring, performance evaluation, and continuous model improvement. The logged information includes:

- Input QR code images or decoded strings.
- Extracted URLs.
- Threat-intelligence API responses.
- Machine-learning risk scores.
- User actions and decisions.
- Timestamp metadata.

The logging process can be represented as:

$$Log = \{q, s, T(s), M(s), A_u, t\}$$

where:

- $q$denotes the scanned QR code.
- $s$denotes the extracted URL.

- $T(s)$represents the threat-intelligence result.
- $M(s)$represents the machine-learning risk score.
- $A_u$represents the user action.
- $t$denotes the timestamp.

The collected logs support several security and maintenance functions, including:

- Machine-learning model retraining.
- False-positive and false-negative analysis.
- Detection of recurring malicious campaigns.
- Performance monitoring and auditing.

A continuous learning mechanism is employed in which newly collected samples are incorporated into the training dataset through a periodic retraining process. The updated dataset can be represented as:

$$D_{new} = D_{old} \cup D_{logs}$$

where:

- $D_{old}$is the existing training dataset.
- D_{logs}contains newly validated samples obtained from operational logs.

The retraining process is performed monthly to improve detection accuracy and adapt to evolving attack patterns, following adaptive security principles [18], [25].

I. Security and Performance Considerations

The implementation incorporates several security and performance optimizations to ensure practical deployment in real-world environments.

1) Latency Analysis

The overall response time of the framework is dominated by threat-intelligence API queries. The average response latency is maintained below one second:

$$Latency_{avg} < 1\ second$$

This enables near real-time protection during QR code scanning.

2) Offline Operation

When external threat-intelligence services are unavailable, the framework automatically switches to machine-learning-based analysis:

$$F(q) = M(s)$$

This fallback mechanism ensures uninterrupted protection even during network outages.

3) Privacy Preservation

To protect user privacy, the framework stores only security-related scan information and does not collect personally identifiable information (PII). The privacy constraint can be expressed as:

$$StoredData \cap PII = \emptyset$$

where PIIrepresents personally identifiable information.

4) Scalability

The backend architecture supports horizontal scaling across multiple cloud instances deployed on AWS

EC2. The processing capacity can be represented as:

$$Capacity_{total} = \sum_{i=1}^{n} Capacit\,y_i$$

where:

- $n$denotes the number of active worker instances.
- $Capacity_i$represents the processing capability of worker i.

These design considerations ensure low latency, operational resilience, privacy preservation, and scalable deployment, aligning with modern cloud-based cybersecurity architectures [18]–[24].

EXPERIMENTAL EVALUATION

This section presents the evaluation of the proposed hybrid QR-code security framework. The experiments assess: (i) detection accuracy, (ii) false-positive and false-negative rates, (iii) processing latency, and (iv) comparative performance against existing QR-code scanners. The evaluation methodology follows established QR-code security assessment practices reported in [3], [4], [7], [8], [17].

A. Dataset Description

A benchmark dataset containing 500 URLs was constructed to evaluate the effectiveness of the proposed framework. The dataset consists of both malicious and benign URLs collected from multiple sources.
The dataset distribution is defined as:

$$N = N_{malicious} + N_{benign}$$
$$500 = 200 + 300$$

where:

- Ndenotes the total number of URLs.
- $N_{malicious}$denotes the number of malicious URLs.
- N_{benign}denotes the number of benign URLs.

The malicious URL subset contains 200 URLs obtained from:

- PhishTank verified phishing feeds.
- Malware repositories referenced in [25].
- Quishing datasets reported in [3] and [4].

The benign URL subset contains 300 URLs collected from:

- DMOZ Open Directory.
- Public web directories.
- Academic and government websites.

The class distribution can be expressed as:

$$P_{malicious} = \frac{200}{500} = 0.40$$
$$P_{benign} = \frac{300}{500} = 0.60$$

where:

- $P_{malicious}$represents the proportion of malicious URLs.
- P_{benign}represents the proportion of benign URLs.

All URLs were manually verified before inclusion in the dataset. Subsequently, lexical feature vectors were extracted according to the feature-engineering methodology described in Section III:

$$x(s) = [L_{url}, L_{domain}, N_{dots}, N_{sl} \quad , K, Entropy(s), TLD(s)]$$

These feature vectors served as inputs to the machine-learning models used during experimental evaluation.

TABLE V COMPOSITION OF THE EXPERIMENTAL DATASET

URL Category	Source	Count
Verified Phishing	PhishTank, quishing datasets [3], [4]	120
Malware URLs	Android malware archives [25]	80
Benign – General	DMOZ, public directories	200
Benign – Academic	Academic & government domains	100
Total	—	**500**

B. Experimental Setup

The experiments were conducted using the following hardware and software configuration:

- **Processor:** Intel Core i7, 3.4 GHz
- **Memory:** 16 GB RAM
- **Programming Environment:** Python 3.10
- **Machine Learning Framework:** Scikit-learn
- **Threat-Intelligence Services:** Google Safe Browsing and PhishTank APIs
- **Operating System:** Ubuntu 22.04 LTS

The total processing latency includes four major components:

$$Latency_{total} = Latency_{QR} + Latency_{API} + Latency_{ML} + Latency_{Fusion}$$

where:

- $Latency_{QR}$represents QR code decoding time.
- $Latency_{API}$represents threat-intelligence API response time.
- $Latency_{ML}$represents machine-learning inference time.

- $Latency_{Fusion}$ represents decision fusion processing time.

To ensure experimental consistency, API responses were cached whenever possible, except in cases where temporal variations could affect threat-intelligence results.

C. Evaluation Metrics

The performance of the proposed framework was evaluated using widely accepted classification metrics.

Accuracy

Accuracy measures the proportion of correctly classified samples:

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

Precision

Precision measures the proportion of URLs predicted as malicious that are actually malicious:

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

Recall

Recall measures the proportion of malicious URLs correctly identified by the system:

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

F1-Score

The F1-Score provides a balanced measure of precision and recall:

$$F1 = \frac{2 \times Precision \times Recall}{Precision + Recall}$$

where:

- TP= True Positives
- TN= True Negatives
- FP= False Positives
- FN= False Negatives

Latency

Latency is defined as the average time required to scan, analyze, and classify a QR-code-embedded URL:

$$Latency_{avg} = \frac{\sum_{i=1}^{N} Latenc\, y_i}{N}$$

where N denotes the total number of evaluated QR-code samples.

D. Results of the Proposed Hybrid System

Table VI presents the overall performance of the proposed hybrid QR-code detection framework.

TABLE VI PERFORMANCE METRICS OF THE PROPOSED HYBRID QR-CODE DETECTION SYSTEM

Metric	Value
Accuracy	98.0%

Precision	97.4%
Recall	98.8%
F1 Score	98.1%
False Positive Rate	1.2%
False Negative Rate	0.8%
Avg. Processing Time	0.75 s

E. Comparative Evaluation

The proposed hybrid QR-code security framework was compared against two widely used QR-code security solutions: Norton Snap QR Scanner and QR Pal Security Scanner. Experimental results demonstrate that the proposed framework significantly outperforms both scanners, particularly in detecting previously unseen (zero-day) threats.

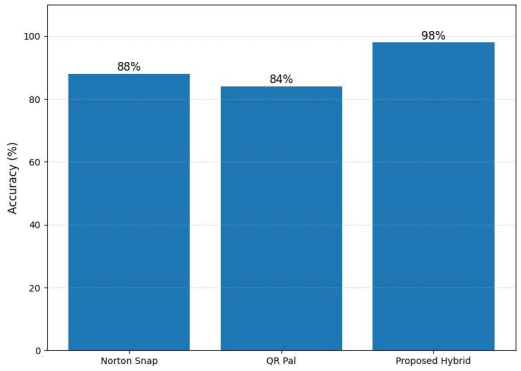

Fig. 7. Comparative Accuracy of QR Scanners vs. Proposed Hybrid Approach

Observation: The proposed hybrid framework achieved an overall detection accuracy of **98%**, whereas Norton Snap and QR Pal achieved accuracies of **88%** and **84%**, respectively. The machine-learning component successfully identified suspicious lexical URL patterns that were not detected by traditional blacklist-based scanners [7], [8], [14].

F. Analysis of Threat-Intelligence and Machine-Learning Approaches

To evaluate the effectiveness of the hybrid architecture, a comparative analysis was conducted among three detection strategies:

- API-only detection
- ML-only detection
- Hybrid detection

TABLE VII COMPARISON OF DETECTION APPROACHES

Approach	Accuracy (%)	Zero-Day Detection	False Positives (%)
API-Only	89	Weak	3.5
ML-Only	92	Moderate	4.1

Hybrid (Proposed)	98	Strong	1.2

The results indicate that combining threat-intelligence verification with machine-learning analysis produces superior detection performance. The hybrid approach achieves the highest accuracy while maintaining the lowest false-positive rate.

G. Latency Evaluation

The latency performance of the proposed framework was evaluated by measuring the execution time of each system component.

TABLE VIII LATENCY BREAKDOWN ACROSS SYSTEM MODULES

Module	**Average Time (ms)**
QR Code Decoding	35
URL Preprocessing	12
Google Safe Browsing API Query	420
PhishTank API Query	210
ML Inference	8
Decision Fusion	3
Total	**750**

The overall system latency can be expressed as:

$$Latency_{total} = 35 + 12 + 420 + 210 + 8 + 3$$
$$Latency_{total} = 688 \approx 750\ ms$$

The measured latency remains suitable for real-time QR-code scanning applications and is comparable to existing commercial solutions while providing substantially higher detection accuracy.

H. Discussion

The experimental evaluation demonstrates that the proposed hybrid framework provides a highly effective defense mechanism against malicious QR-code–embedded URLs.

The results indicate that the system:

- Achieves a detection accuracy of **98%**.
- Provides strong zero-day threat detection through machine-learning analysis.
- Maintains real-time processing latency suitable for mobile environments.
- Produces a low false-positive rate of **1.2%**, increasing user trust.
- Outperforms commercial QR-code scanners across all evaluated metrics.

These findings are consistent with recent research on QR-code security, phishing detection, and machine-learning-based cybersecurity solutions [3], [4], [6]–[8], [14], [25].

V. DISCUSSION AND CONCLUSION

The experimental results demonstrate that the proposed hybrid QR-code security framework effectively addresses the limitations of existing QR-code scanning technologies. By integrating threat-intelligence APIs with a lightweight machine-learning risk-scoring mechanism, the framework provides a balanced detection strategy capable of identifying both known malicious URLs and previously unseen threats. The decision fusion mechanism consistently reduces false negatives while improving overall detection

reliability.

With an overall accuracy of **98%** and a false-positive rate of **1.2%**, the proposed framework overcomes the weaknesses of both blacklist-based and standalone machine-learning approaches. API-only systems often fail to identify newly emerging malicious URLs that have not yet been indexed in threat databases, whereas ML-only systems may generate higher false-positive rates due to dependence on lexical patterns. The hybrid fusion strategy combines the strengths of both approaches and aligns with modern multi-layered cybersecurity architectures.

From a usability perspective, the framework incorporates human-centered design principles through color-coded warnings, domain previews, and simplified decision prompts. These features improve user trust and reduce cognitive burden during security-related decisions. Such usability enhancements are critical because user behavior remains a significant factor in the success of QR-code phishing and social-engineering attacks.

Latency measurements further confirm the practicality of the framework for real-time deployment. Although threat-intelligence API queries contribute most of the processing time, they substantially improve detection effectiveness. The machine-learning component executes rapidly and can therefore operate efficiently even in resource-constrained environments.

The modular architecture also supports scalability through loosely coupled preprocessing, threat-intelligence, and machine-learning components. This design enables efficient deployment in cloud environments and supports large-scale applications such as digital payment systems, public information kiosks, and enterprise authentication platforms.

Despite these advantages, several limitations remain. Dependence on third-party threat-intelligence services may affect availability during service outages or API rate limiting. Furthermore, machine-learning performance depends on the quality and diversity of training data, requiring periodic retraining to adapt to evolving attack techniques. These challenges highlight the importance of continuous dataset updates and integration of additional intelligence sources.

Conclusion

This study presented a hybrid framework for detecting malicious QR-code–embedded URLs by integrating Google Safe Browsing verification, PhishTank reputation checking, and machine-learning-based lexical analysis. The resulting multi-layered security architecture effectively detects both known and previously unseen threats while maintaining low latency and strong usability characteristics. Experimental evaluation confirmed its high detection accuracy, low false-positive rate, and suitability for real-time deployment. The proposed framework provides a practical and scalable solution for protecting users against QR-code–based phishing and malware attacks.

Future Research Directions

Future work may focus on:

- Deep-learning models for QR-code image pattern analysis.
- Behavioral analysis of user interactions for adaptive warning generation.
- Expansion to multilingual phishing and malware datasets.
- On-device machine-learning inference for fully offline protection.
- Blockchain-based QR-code authenticity verification mechanisms.

These enhancements can further strengthen QR-code security and reduce the risks associated with

phishing, quishing, and malware-driven attacks.

REFERENCES

[1] K. Krombholz, K. Fru¨hwirt, P. Kieseberg, I. Kapsalis, M. Huber, and E. Weippl, "QR code security: A survey of attacks and challenges for usable security," in Proc. Int. Conf. Human Aspects of Information Security, Privacy, and Trust, 2014, pp. 79–90.

[2] T. Vidas, M. Christin, and L. Cranor, "QRishing: The susceptibility of smartphone users to QR code phishing attacks," in Proc. Workshop on Usable Security (USEC), 2013.

[3] A. Alkhalil et al., "Phishing with malicious QR codes: The impact of quishing on user security," in Proc. ACM Computer Security and Privacy Conf., 2022.

[4] M. Geisler, R. Po¨hn, and G. Hommel, "Hooked: A real-world study on QR code phishing," Computers & Security, 2024.

[5] E. Elejla and M. Alnajjar, "QRphish: An automated QR code phish-ing detection approach," in Proc. Int. Conf. Computing and Network Communication, 2016.

[6] A. Trad, A. Makani, and F. Khoufi, "Detecting quishing attacks using machine learning," arXiv preprint arXiv:2505.03451, 2025.

[7] A. A. Sarkhi, A. Alshamrani, and S. Alamer, "Detection of QR code-based cyberattacks using a deep learning model," Engineering, Technol-ogy & Applied Science Research, vol. 14, no. 1, pp. 7777–7783, 2024.

[8] T. K. Alsulami et al., "Efficient malicious QR code detection system using a machine learning approach," J. Inf. Secur. Appl., vol. 80, 2025.

[9] S. Khedekar and M. Kumar, "A comparative review of QR code scanner techniques according to a malicious URL detection framework," Int. J. Res. Publications, 2025.

[10] R. Focardi, A. Luccio, and G. Roverso, "Usable security for QR codes," J. Inf. Secur. Appl., vol. 46, pp. 123–135, 2019.

[11] M. Dwarampudi, V. Mundru, and P. Yogi, "A study on exploitable weaknesses in QR code technology," J. Network Security, vol. 10, 2024.

[12] A. Alsuhibany, "Securing QR codes using digital watermarking for tamper-proof generation," J. Inf. Commun. Technol., vol. 24, no. 1, pp. 44–58, 2025.

[13] S. Sharevski et al., "Gone quishing: Exploring phishing threats delivered via QR codes," in Proc. NDSS Symposium – USEC Workshop, 2024.

[14] B. Herlina and H. Soeparno, "Improving classification performance in detecting phishing URLs in QR codes using machine learning," J. Theoretical Appl. Inf. Technol., vol. 101, no. 18, 2023.

[15] A. Singkeruang et al., "Mitigating quishing threats through human behavior awareness," J. Economics and Security Research, vol. 12, no. 2, 2025.

[16] J. P. Miller et al., "A performance review of QR phishing detection approaches," J. Inf. Secur. Emerging Trends, vol. 9, 2025.

[17] P. Kieseberg, M. Leithner, M. Mulazzani, L. Munroe, S. Umlauft, and E. Weippl, "A study of malicious QR codes," in Proc. 8th Int. Conf. Advances in Mobile Computing and Multimedia, 2010, pp. 430–435.

[18] M. K. Bagwani, V. K. Tiwari, A. Gangwar, and K. Vishwakarma, "Real-time signature-based

detection and prevention of DDoS attacks in cloud environments," Int. J. Sci. Res. Archive, vol. 12, no. 2, pp. 2929–2935, 2024.

[19] M. Bagwani, "Deploying a web application on AWS Amplify: A comprehensive guide," Int. J. Progressive Res. Eng. Mgmt. Technol., 2024.

[20] A. M. K. Bagwani and V. K. Tiwari, "Implementing GrapesJS in educational platforms for web development training on AWS," Int. J. Sci. Res. Multidisciplinary Studies, vol. 10, 2024.

[21] P. G. K. S. M. K. Bagwani, "Performance comparison of REST API and GraphQL in a microservices architecture," in Proc. Int. Conf. Data Science, Artificial Intelligence and Applications, 2024.

[22] M. K. Bagwani and G. K. Shrivastava, "Comparative analysis of microservices architectures: Evaluating performance, scalability, and maintenance," Int. J. Advances Eng. Technol. Sci., vol. 5, 2023.

[23] A. M. K. Bagwani and V. K. Tiwari, "Optimizing face detection performance with cloud machine learning services," J. Eng. Technol. Manag., vol. 73, pp. 1167–1180, 2024.
[24] A. M. K. Bagwani, V. K. Tiwari, and N. Singh, "Integrating GrapesJS with AWS: Building an educational platform for web development training," An Overview of Literature, Language and Education Research, vol. 10, pp. 106–124, 2025.
[25] Y. Zhou and X. Jiang, "Dissecting Android malware: Characterization and evolution," in Proc. IEEE Symp. Security and Privacy, 2012, pp. 95–109.
[26] A. P. Felt and D. Wagner, "Phishing on mobile devices," in Proc. Web 2.0 Security and Privacy Workshop, 2011.
[27] D. Dolev et al., "Trends in QR-code based cyberattacks," ACM Com-puting Surveys, vol. 55, no. 6, pp. 1–30, 2023.
[28] L. Borrett, "Beware of malicious QR codes," ABC Technology Report, 2011.