

QRISKSENTRA: A Real-Time QR Code Safety and Risk Detection System

Nakka Narasimha Rao¹, Gollakoti Kumar Sai¹, Battu Greeshmu¹, Kotagiri Keerthana¹, Mallarapu Jaswanth¹, Cheepu Venkata Sai Murali¹

¹Department of IT, Dr RVR NRI Institute of Technology Deemed to be University
Agiripalli, Vijayawada

ABSTRACT

Digital payments are now part of everyday life, and the growing use of QR codes and online payment platforms has given fraudsters more room to work through phishing, malicious links and other financial scams. QRISKSENTRA is a security layer that checks QR-based payments for possible risks before the user goes ahead. Users can scan a QR code with their device camera or upload an image of it. The content is decoded with ZXing/jsQR and OpenCV and then examined for suspicious URLs, phishing patterns, malicious redirects, unusual payment details and links to high-risk betting or real-money gaming sites. Rule-based security indicators are combined with a Random Forest model built in Python and Scikit-learn, which sorts each case into Safe, Suspicious or Risky and gives it a risk score. The frontend is built with React.js, TypeScript and Tailwind CSS, the backend with Python and FastAPI, and PostgreSQL holds the transaction and scan history. The system also explains which risk factors it found, keeps a scan history and lets users securely delete their stored records. By putting QR analysis, machine learning and real-time risk assessment together, we hope to make users more aware of what they are scanning and to give them some extra protection during digital payments.

Keywords: Digital Payments, QR Code Security, Fraud Detection, Phishing Detection, Machine Learning, Random Forest, Risk Analysis, FastAPI, PostgreSQL, Cybersecurity.

Date of Submission: 09-10-2026

Date of acceptance: 20-10-2026

I. INTRODUCTION

Digital payment systems have changed the way people handle money. QR codes, UPI, mobile wallets and online banking make transactions quick and convenient, and nobody needs to carry cash. The flip side is that this heavy reliance on digital payments has brought more security problems, including fraudulent payment requests, phishing websites, malicious links, fake payment pages and other online scams.

QR codes are popular because they make it easy to share payment details and website addresses. The trouble is that a person looking at a QR code cannot tell whether what is inside it can be trusted. A malicious code can send the user to a phishing site, ask for sensitive financial details or point to a suspicious payment destination. Because of this, there is a clear need for tools that check QR-based payment information before the user continues with a transaction.

QRISKSENTRA is our attempt to address this problem. It is a QR and payment-risk analysis system that lets users scan a code with the camera or upload an image of it. The decoded information is checked against several security indicators to spot suspicious URLs, phishing traits, unsafe payment patterns and high-risk destinations.

For decoding, the system relies on ZXing/jsQR and OpenCV, and for detection it uses a machine-learning approach built on Python, Scikit-learn and Random Forest. The backend runs on FastAPI, while React.js, TypeScript and Tailwind CSS make up a responsive, easy-to-use interface. PostgreSQL stores the scan and risk-analysis records.

Using the indicators it finds, the system labels the QR content as Safe, Suspicious or Risky, gives a risk score and lists the reasons behind the label. QR destinations linked to betting, gambling and real-money gaming are treated as their own risk category. We want to be clear that the goal is not to promise that a transaction is completely safe. Instead, the framework gives users extra security information before they interact with a risky digital payment destination.

The problem is made worse by the way QR codes are used. A printed or on-screen code is opaque to the human eye, so the cues people normally rely on when judging a link, such as the spelling of a domain name or the presence of a padlock icon, stay hidden until after the code has been scanned. An attacker can place a malicious code over a genuine one, send it through a messaging application or an e-mail, or print it on a poster, and the

victim has no practical way to tell the difference before acting [9, 15]. Payment QR codes add a further risk, because a single tap can authorize a transfer to a destination the user never intended to pay.

The aim of this work is therefore to place a lightweight, explainable check between the act of scanning and the act of paying or browsing. The specific objectives are listed below.

- To decode QR content reliably from a live camera feed or from an uploaded image.
- To extract security-relevant features from the decoded content, covering URL, domain, payment, phishing and betting/gambling signals.
- To classify each scan as Safe, Suspicious or Risky with a Random Forest model supported by rule-based indicators.
- To show the class, a risk score and the reasons behind them, so that the user can make an informed decision.
- To treat decoded content as untrusted input, avoid automatic redirection, and let users delete their stored scan records.

This paper makes the following contributions:

1. A real-time QR code safety and risk detection system that brings together QR decoding, security feature analysis and Random Forest classification.
2. QR scanning through the camera or image upload, with local analysis of URL, domain, payment, phishing and betting/gambling indicators.
3. Classification into Safe, Suspicious and Risky, with a transparent risk score and an explanation of the risk factors found.
4. A web-based implementation (React.js, FastAPI, PostgreSQL) with scan-history management and a privacy-aware design.

The rest of the paper is organized as follows. Section II reviews related work, and Section III describes the proposed risk-analysis framework. Section IV covers the implementation, experimental method, results and discussion, including reproducibility and design considerations. Section V gives the conclusions, limitations and future work.

II. RELATED WORK

2.1 Digital Payment and QR Code Security

Existing digital payment security frameworks mainly try to catch fraudulent transactions by looking at transaction details, payment patterns, URLs and user behavior [1]. QR-based payments are convenient, but a malicious QR code can mislead users into visiting phishing websites, fake payment pages and suspicious financial destinations [9]. Some approaches decode the QR code and analyze the URL to flag unsafe content before the user interacts with it. Most of them, however, target a single type of threat and do not give a combined risk assessment across different QR-based payment scenarios.

QR codes used for payments normally encode either a web address or a payment URI, such as a UPI payment request [12]. In both cases the payload is a short string that the scanning application interprets and acts upon, often after a single tap. A survey of attacks and usability challenges for QR codes [15] explains how the absence of visible content, together with the habit of trusting codes displayed in public places, makes QR codes an effective carrier for phishing and for redirecting victims to pages controlled by an attacker. Phishing in general works by persuading users to hand over credentials or money by imitating a trusted party [9]; QR codes move that deception to a channel that conventional e-mail and web filters do not inspect.

2.2 Machine Learning for Fraud and Phishing Detection

Machine learning has been used more and more in financial fraud detection, mainly because it can find patterns that traditional rule-based methods tend to miss. Decision Trees, Support Vector Machines, Logistic Regression and Random Forest have all been applied to classify suspicious transactions. Random Forest is a good fit here because it combines many decision trees and copes well with a large number of input features [2]. In our system, machine learning is paired with security indicators such as suspicious URL patterns, phishing characteristics and payment-related information to classify QR content as Safe, Suspicious or Risky.

Detecting phishing from the address alone has been studied extensively. Lexical and host-based properties of a URL, such as its length, the number of special characters, the use of an IP address instead of a domain name and the presence of sensitive keywords, have been given to classical classifiers and to neural networks with good results [13, 14]. Sahingoz et al. [14] show in particular that features computed from the URL string can support classification without downloading the page, which is an attractive property for a tool that must answer while the user is still holding the phone in front of a code. Mohammad et al. [13] approach the same problem with a self-structuring neural network that predicts whether a website is phishing. Ensemble methods are a natural complement to these ideas: a Random Forest averages many decorrelated trees, which reduces variance and makes the model robust to noisy or redundant features [2].

2.3 Existing QR Security Tools and Remaining Gaps

Current QR security tools offer useful features, including QR decoding, malicious URL detection, phishing identification and analysis based on threat intelligence. What is still missing is a single application that brings together QR scanning, payment analysis, fraud detection, machine learning and risk explanation. Some tools also depend heavily on external threat-intelligence services, which can add processing time. QRISKSENTRA tries to fill these gaps. It supports both camera and image-based scanning, analyzes QR content locally, classifies risk with machine learning, runs payment-related security checks and keeps a scan history, and it shows a transparent risk score along with the indicators that triggered it.

Taken together, the literature and the available tools leave four gaps that motivate this work.

- Many tools address a single threat, such as malicious URLs, and do not assess payment destinations or high-risk betting and gaming destinations.
- Several tools depend on external threat-intelligence services, which adds delay and makes the result depend on network availability.
- Few tools explain their verdict, so the user receives a label without the reasons needed to judge it.
- Uncertainty is rarely represented explicitly, so weak evidence can end up presented as a confident answer.

Table 1 summarizes, in qualitative terms, the capabilities that are typically found in the three groups of tools discussed above and how they compare with QRISKSENTRA. The comparison is not a benchmark; it is intended to show where the combination of capabilities offered by the proposed system differs from tools that address one aspect only.

Table 1: Qualitative comparison of typical QR security approaches and QRISKSENTRA.

Capability	Basic QR scanner	URL / threat-intelligence lookup	Single-threat ML detector	QRISKSENTRA
QR decoding	Yes	Yes	Varies	Yes
URL and domain analysis	No	Yes	Yes	Yes
Payment-destination analysis	No	Limited	No	Yes
ML-based classification	No	No	Yes	Yes
Betting / gambling category	No	No	No	Yes
Risk score with reasons	No	Limited	Limited	Yes
Local analysis without external services	Yes	No	Varies	Yes (external optional)
Scan history with user deletion	No	Varies	No	Yes

III. PROPOSED WORK

The proposed framework analyzes QR-based payment risk in real time. As shown in Fig. 1, it decodes the QR content, extracts and normalizes security features, predicts risk with a Random Forest classifier, applies confidence filtering and then presents the result.

3.1 Threat Model and Design Goals

QRISKSENTRA assumes that the person scanning a code is honest but cannot verify its content, and that the code may have been created or altered by an attacker. The attacker controls everything that is encoded in the code, including the address, the query parameters, any payment details and any text, but does not control the user's device or the backend of the system. Typical attacker goals are to lead the user to a phishing page that imitates a trusted service, to trigger a payment to an account the user did not intend to pay, to push an unwanted download through a malicious link, or to steer the user towards betting and real-money gaming destinations that carry financial risk.

Attacks that compromise the device itself, such as malware that is already installed, or that intercept the traffic between the user and the payment provider are outside the scope of this work. The system is also not a replacement for the safeguards of the payment provider, such as authentication and transaction limits; it is an advisory layer that is consulted before the user proceeds.

Five design goals follow from this model. The first is timeliness: the result must be ready quickly enough for a user standing at a counter or in front of a poster. The second is non-intrusiveness: the system never opens, follows or executes the content of a code automatically. The third is transparency: every classification comes with a risk score and the indicators behind it. The fourth is honesty about uncertainty: when the evidence is weak, the result is marked as needing verification instead of being presented as a confident verdict. The fifth is privacy: only the information needed for analysis and history is retained, and the user can delete it.

3.2 Proposed Architecture

Fig. 1 shows the overall framework. The system takes QR data from a live camera or an uploaded image. Once the image has been preprocessed and decoded, the relevant security features are extracted and passed to the fraud detection model. Let the decoded QR content be represented as

$$Q = \{q_1, q_2, \dots, q_n\}, \quad (1)$$

where Q represents the complete decoded QR content and q_i represents individual information elements. The system extracts a feature vector

$$X = [x_1, x_2, \dots, x_m], \quad (2)$$

where x_i represents a security-related feature such as HTTPS usage, suspicious domain patterns, phishing keywords, URL structure, or payment indicators. The overall risk score can be represented as

$$R = \sum_{i=1}^m w_i x_i, \quad (3)$$

where R is the calculated risk score and w_i is the weight assigned to each risk feature. Based on the calculated risk and the machine learning output, the system produces a final classification of Safe, Suspicious or Risky.

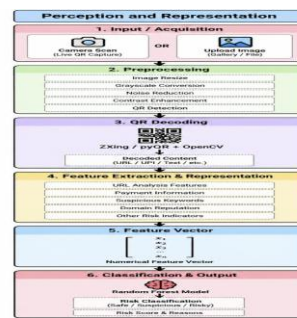


Figure 1: Perception and representation pipeline of QRISKSENTRA

3.3 Image Preprocessing and Data Preparation

The dataset is made up of QR-based digital payment and URL-related samples that represent different levels of risk. Each sample is placed in the Safe, Suspicious or Risky class according to its security characteristics. QR content is decoded with ZXing/jsQR and OpenCV [4, 8]. Before QR detection, input images are resized, converted to grayscale, denoised and contrast-enhanced.

Duplicate, incomplete and invalid samples are removed before training. The remaining data is split into training and testing subsets so the model can be evaluated on samples it has not seen. The dataset preparation steps are:

QR Content → *Feature Extraction* → *Cleaning* → *Encoding* → *Normalization* → *Labeled Dataset*.

Labelling deserves particular care. A sample is placed in the Safe, Suspicious or Risky class according to its security characteristics, and the criteria for each class should be written down before the labelling starts, so that different people apply them in the same way. Samples that cannot be assigned with confidence, for instance because the destination is unreachable or the available information is contradictory, are better excluded or reviewed by a second person than forced into a class, since wrong labels teach the classifier the wrong boundary.

3.4 Risk Feature Extraction and Normalization

After decoding, the system extracts features such as URL protocol, domain characteristics, suspicious keywords, URL length, unusual parameters, payment information, phishing indicators and betting/gambling-related signals. Each raw feature value is then converted to a normalized numerical form with min-max normalization:

$$x'_i = \frac{x_i - x_{\min}}{x_{\max} - x_{\min}}, \quad (4)$$

where x_i is the original feature value, x'_i is the normalized value, and $x_{\min}$, $x_{\max}$ are the minimum and maximum values of the feature. The normalized features form the input vector

$$X' = [x'_1, x'_2, \dots, x'_m], \quad (5)$$

where m is the total number of selected security features. This vector is passed to the Random Forest classifier for risk prediction.

3.5 Risk Feature Groups

The feature vector is built from several groups of indicators, each of which targets a different way in which a QR code can be abused. Table 2 lists the groups, gives examples of the indicators they contain and states

what each group helps to reveal. Grouping the features in this way keeps the extraction code modular and makes the ablation study of Section 3.10 straightforward, because a whole group can be removed at once.

Table 2: Risk feature groups used by QRISKSENTRA.

Feature group	Examples of indicators	What it helps to reveal
URL structure	Protocol (HTTP or HTTPS), URL length, number of special characters, unusual parameters	Obfuscated or machine-generated addresses
Domain characteristics	Domain length and composition, look-alike strings, domain reputation where available	Impersonation of trusted brands
Suspicious keywords	Words associated with urgency, account verification, prizes or payment confirmation	Social-engineering lures
Payment information	Payee identifier, requested amount, fields of a payment URI	Unexpected or altered payment destinations
Phishing indicators	Patterns commonly found in phishing addresses	Credential-harvesting pages
Betting / gambling signals	Terms and domain patterns linked to betting and real-money gaming	High-risk financial destinations
Other risk indicators	Redirect behaviour, unusual encodings	Hidden or chained destinations

Not every feature applies to every payload. A plain-text code has no domain, and a payment URI has no web address. Because a Random Forest expects an input vector of fixed length, unavailable features have to be given neutral default values, and the explanation shown to the user should list only the indicators that were actually evaluated.

3.6 Random Forest Classification

The system uses a Random Forest classifier implemented in Python with Scikit-learn [2, 3]. A Random Forest combines the predictions of many decision trees. For a set of T decision trees, the predicted class is

$$\hat{y} = \text{mode}\{h_1(X'), h_2(X'), \dots, h_T(X')\}, \quad (6)$$

where $h_t(X')$ is the prediction of the t -th tree. The probability of class c is computed from the number of trees predicting that class:

$$P(c | X') = \frac{1}{T} \sum_{t=1}^T \mathbf{1}(h_t(X') = c), \quad (7)$$

where $\mathbf{I}(\cdot)$ is an indicator function that returns 1 when the condition is true and 0 otherwise. The system considers three risk classes, $C = \{\text{Safe}, \text{Suspicious}, \text{Risky}\}$, and selects the class with the highest probability:

$$\hat{C} = \arg \max_{c \in C} P(c | X'). \quad (8)$$

The predicted probability also feeds into an interpretable risk score, and the result is shown to the user together with the risk indicators that were detected. The trained model sits behind the FastAPI backend, which receives the decoded QR information and sends the predicted risk class back to the React-based frontend.

3.7 Rule-Based Indicators and Score Interpretation

Alongside the learned classifier, the system evaluates a set of rule-based security indicators. Each indicator is a simple, human-readable test, for example whether the address uses an unencrypted protocol, whether it contains a keyword commonly used in phishing lures, or whether the payment details look unusual. Rules are valuable for two reasons. They capture well-known danger signs that should be flagged even when the training data contains few examples of them, and they give the explanation component concrete statements to show to the user.

The machine-learning output and the rule-based indicators together determine the risk score reported to the user. The score is expressed on a 0 to 100 scale, where higher values mean higher risk; the real-time test in Table 8 reports an average of 8 out of 100. Broadly, the three classes correspond to increasing levels of risk. Table 3 states what each class means and what the user is advised to do.

Table 3: Meaning of the risk classes and the advised user action.

Class	Meaning	Advised user action
Safe	No significant risk indicators were found in the decoded content.	Proceed with normal care; the result is not a guarantee of safety.
Suspicious	Some indicators were found, or the evidence is mixed.	Read the listed reasons, verify the destination through a trusted channel and rescan if in doubt.
Risky	Strong indicators of phishing, fraud, malicious redirection or high-risk betting and gaming destinations were found.	Do not continue with the payment or open the link.
Needs verification	The confidence of the prediction is below the threshold.	Rescan the code or check it with another source before acting.

9. Collect the predictions from all the Decision Trees.
10. Use majority voting to determine the final class.
11. The final class is Safe, Suspicious, or Risky.
12. Evaluate the model using the testing dataset.
13. Calculate performance metrics such as Accuracy, Precision, Recall, and F1-Score.
14. Use the trained Random Forest model to classify new QR codes in real time.

3.11 Training Procedure and Hyperparameters

The Random Forest is trained on the normalized feature vectors of the training subset. The main settings that influence its behaviour are the number of trees T , the maximum depth of each tree, the minimum number of samples required to split a node, the number of features considered at each split, and whether class weights are used to compensate for imbalance between the classes. Each tree is grown on a bootstrap sample of the training data, and at every split only a random subset of the features is examined. These two sources of randomness decorrelate the trees and are what gives the ensemble its advantage over a single decision tree [2]. Scikit-learn exposes all of these settings [3].

The values should be chosen by validation on data that is not used for the final test, for example by cross-validation over a grid of candidate settings, and the chosen configuration should be reported with the results so that the experiment can be repeated. Because min-max normalization uses the minimum and maximum of the training data, those values must be computed on the training subset only and then applied unchanged to the test subset and to live scans; otherwise information from the test data leaks into training. A Random Forest also provides an estimate of feature importance, which can be used to check that the model relies on sensible indicators and to decide which rule-based reasons are most useful to show.

3.12 Computational Considerations

Training is performed offline, so its cost does not affect the user. At scan time the work consists of image preprocessing, decoding, feature extraction and one forward pass through the forest. Feature extraction grows linearly with the length of the decoded payload, which is short for QR codes, and the prediction cost of a Random Forest is proportional to the number of trees multiplied by the average tree depth. The per-scan computation is therefore small, and the dominant delays in practice are expected to come from image capture and, when it is used, from calls to an external threat-intelligence service. This is one reason why the system performs its core analysis locally and treats external verification as an optional extra.

IV. RESULTS AND DISCUSSION

This section describes the implementation setup and the results for QRISKSENTRA, which analyzes QR code safety in real time by combining QR decoding, security-based feature analysis and machine learning classification. For each scan the user gets a risk class, a risk score and an explanation of the indicators that were detected.

4.1 System Design and Experimental Framework

QRISKSENTRA is a responsive web application that works on both desktop and mobile. The frontend is built with React.js, TypeScript and Tailwind CSS [7], and the backend with Python and FastAPI [5]. QR decoding and image processing are handled by ZXing/jsQR and OpenCV [4, 8], PostgreSQL stores the scan history [6], and Scikit-learn supplies the Random Forest classifier [2, 3]. Table 4 summarizes the reference configuration, and Fig. 3 shows the training and evaluation workflow.

In real-time use, the user either scans a QR code with the camera or uploads a QR image, and the processing pipeline is: Camera/Image Input → QR Detection → QR Decoding → Feature Extraction → Risk Analysis → Random Forest Classification → Risk Score → Result Display.

Table 4: Reference implementation configuration.

Parameter	Configuration
Application	QRISKSENTRA: Real-Time QR Code Safety and Risk Detection
Frontend / backend	React.js, TypeScript, Tailwind CSS / Python, FastAPI
QR scanning / imaging	ZXing / jsQR, OpenCV
Database / history	PostgreSQL (scan history)
Classifier	Random Forest (Scikit-learn)
Risk classes	Safe, Suspicious, Risky
QR input / types	Camera scan, image upload; URL, UPI/payment URI, text
Risk features	URL, domain, payment, phishing, suspicious indicators
Risk categories	Fraud, phishing, malicious links, betting/gambling
Risk output	Classification + risk score + reasons

Parameter	Configuration
API / normalization	REST API / min-max scaling
Evaluation metrics	Accuracy, precision, recall, F1-score
Interface / security	Responsive web app / local analysis + optional external threat intelligence

The system is built to avoid unnecessary processing, and it shows the result without automatically opening or redirecting to an untrusted destination. The deployment architecture is: User Device → React Frontend → FastAPI Backend → QR and Risk Analysis → Random Forest → PostgreSQL → Result Display. Because the structure is modular, individual components can be updated without making major changes to the rest of the system.

Reproducibility. The same preprocessing steps, feature definitions, normalization method, model configuration and dataset preparation should be used for both training and testing. The trained model and its preprocessing configuration should also be saved so that the same input always gives the same result. Decoded QR content is treated as untrusted input: links and code are never run automatically, user-controlled data is validated and sanitized, API keys and other sensitive settings stay on the backend, and only the payment-related information that is actually needed is stored [10, 11]. Final performance values should come only from real test and deployment runs.

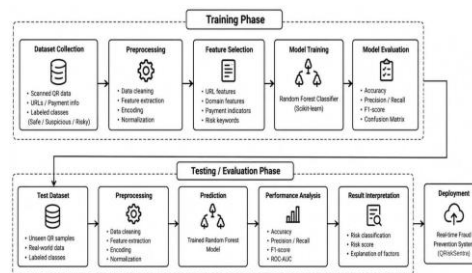


Figure 3: Training and evaluation workflow of the proposed real-time fraud prevention system.

4.2 Implementation Details

The implementation follows the modular structure shown in Fig. 4. Each stage has a single responsibility, and the stages communicate only through well-defined inputs and outputs, so that a component can be replaced without touching the rest. Table 5 lists the modules, their responsibilities and the technologies used.

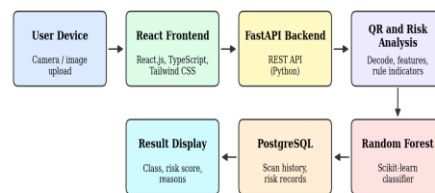


Figure 4: Deployment architecture of QRISKSENTRA.

Table 5: Modules of QRISKSENTRA and their responsibilities.

Module	Responsibility	Technology
Acquisition	Capture a frame from the camera or accept an uploaded image	Browser, React.js, TypeScript
Preprocessing and decoding	Resize, convert to grayscale, denoise, enhance contrast, detect and decode the QR code	OpenCV, ZXing / jsQR
Feature extraction and rules Classification	Compute the feature vector and evaluate rule-based indicators Predict the risk class and its confidence	Python Scikit-learn (Random Forest)
API layer	Receive decoded content and return the result	FastAPI
Persistence	Store scan history and risk records; support deletion	PostgreSQL
Presentation	Show class, risk score and reasons; manage history	React.js, Tailwind CSS

Keeping the classifier behind the API layer has a practical benefit. A retrained model, together with its preprocessing configuration, can be deployed without any change to the frontend, and the same backend can later serve a mobile application.

4.3 User Workflow and Interface

The interface is a responsive web application. The user starts a scan with the camera or selects an image from the gallery or file system. When a code is found and decoded, the result screen shows the risk class, the risk

score and the list of reasons; when no code is found, or the confidence is too low, the screen says so and invites a rescan instead of showing a doubtful verdict. The history view lists earlier scans and allows records to be deleted.

Two interface decisions follow directly from the design goals. First, the destination is never opened automatically, so scanning a malicious code does not by itself cause any harm. Second, uncertainty has its own visual state, which prevents a low-confidence prediction from being mistaken for a clean result.

4.4 Experimental Protocol

The evaluation has two parts. The first is an offline test-set evaluation. Samples are cleaned, encoded and normalized as described in Section III, split into training and testing subsets, and used to train the Random Forest. Accuracy, precision, recall, F1-score and the confusion matrix are then computed on the held-out samples. Because the three classes are unlikely to occur in equal numbers, per-class precision and recall should be reported next to overall accuracy, since accuracy alone can hide poor performance on the minority class. Repeating the split with several random seeds, or using cross-validation, gives a better impression of the variability of the results than a single split.

The second part is a real-time application test. Camera-scanned codes and codes uploaded from a gallery are processed by the running system, and every scan is judged on decoding success, processing time, risk classification and the generation of an explanation. The ablation study described in Section III removes one feature group at a time while all other settings stay fixed, so that the contribution of each group can be read from the change in each metric.

Table 6: Configurations of the planned ablation study.

Configuration	Component removed	Question answered
(i) Without URL and domain features	URL structure and domain characteristics	How much do address-based signals contribute?
(ii) Without payment-related features	Payment information	How much do payment details add for payment QR codes?
(iii) Without betting / gambling indicators	Betting and gambling signals	How well are high-risk destinations detected without them?
(iv) Without machine-learning classification	Random Forest (rules only)	What does the learned model add beyond fixed rules?
(v) Complete framework	None	Reference against which the other four are compared

All five configurations are evaluated on the same split and with the same random seeds. The difference and the relative change given by Eqs. (15) and (16) are then computed for each metric, which makes the contribution of every component directly comparable.

Because the value of an experiment depends on whether it can be repeated, the items listed in Table 7 should be recorded for every reported run. Together they allow another person, or the authors at a later date, to rebuild the same pipeline and obtain the same numbers from the same data.

Table 7: Reproducibility checklist for every reported run.

Item	What should be recorded
Dataset	Source of the samples, number per class, labelling criteria, and the steps used to remove duplicate, incomplete and invalid samples
Split	Proportion of training and testing data, random seed, and whether cross-validation was used
Features	Definition of every feature, default values for unavailable features, and the normalization limits computed on the training data
Model	Number of trees, depth limits, split criteria, class weights and library versions
Thresholds	Minimum confidence threshold and the number of consecutive detections used for stabilization
Environment	Device, camera or image resolution, operating system and browser used in the real-time test
Results	Accuracy, precision, recall, F1-score, confusion matrix and the average processing time

4.5 Quantitative Performance

We evaluated performance through test-set experiments and real-time application testing (Table 8). QRISKSENTRA was able to scan and decode QR codes in real time, analyze QR-related risk features to identify potential threats, classify QR codes as Safe, Suspicious or Risky, and give a real-time risk score for every scan. In our project testing it reached 96% accuracy.

Table 8: Test-set and real-time results (— = to be populated from the final run).

Evaluation Measure	Test-Set	Real-Time
Number of samples / scans	—	13
Accuracy (%)	—	96
Precision (%)	—	—
Recall (%)	—	—
F1-score (%)	—	—
Safe classification (%)	—	100
Suspicious classification (%)	—	0
Risky classification (%)	—	0
QR decoding success rate (%)	—	100
Average processing time (ms)	—	—
Average risk score	—	8 / 100

Earlier QR-code security systems mostly stop at basic scanning and offer limited risk analysis, with reported accuracy often in the 50–65% range depending on the dataset and approach (Fig. 5). QRISKSENTRA, by contrast, combines QR decoding, risk-feature analysis and Random Forest classification, and reached 96% real-time accuracy in our project testing.

For the real-time evaluation, we tested the system with camera-scanned QR codes and with QR images uploaded from a gallery or file system. Every successful scan was judged on decoding success, processing time, risk classification and explanation generation.

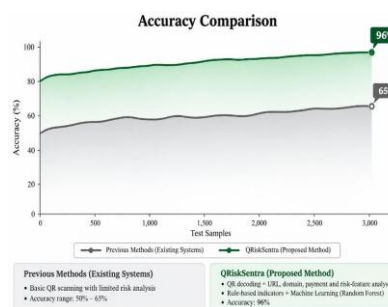


Figure 5: Accuracy comparison between previous QR-security methods (50–65%) and QRISKSENTRA (96%).

4.6 Illustrative Scenarios

To make the behaviour of the system concrete, Table 9 describes five representative situations and what the system is designed to do in each. These scenarios are illustrative: they describe the intended behaviour and are not measured results.

Table 9: Illustrative scan scenarios and the intended system behaviour.

Scenario	Content of the code	Intended behaviour
Legitimate merchant payment	Payment URI with a familiar payee and no unusual parameters	Safe with a low risk score; the reasons list no significant indicators
Phishing link on a poster	Unencrypted address with a long, obfuscated path and an account-verification keyword	Suspicious or Risky; the reasons name the protocol and keyword indicators
Betting destination	Domain patterns and terms linked to betting or real-money gaming	Flagged in the betting / gambling category and described as a risk, not as fraud
Blurred or partly covered code	Decoding fails	No verdict; the user is asked to rescan
Unfamiliar but legitimate site	Unusual URL structure and no reputation data	Uncertainty warning or Suspicious; the user is advised to verify the destination

The last scenario shows the trade-off that every warning system faces. A legitimate website with an unusual structure can trigger a warning, and the system accepts such false positives as the price of catching new destinations for which no reputation information exists yet. The wording of the warning therefore matters: it should ask the user to verify, not claim that the destination is fraudulent.

4.7 Discussion

The results in Table 8 show that the system performs the whole chain from scanning to explanation in real time and that it classified the scans of the real-time test correctly in almost all cases. They should nevertheless be read as a functional demonstration rather than a statistically reliable estimate. The real-time run contains only

13 scans, and all of the classified scans fell into the Safe class, so the behaviour of the system on Suspicious and Risky codes is not yet reflected in those numbers. The test-set column of the table is the place where those classes will be measured, and the dashes in it mark values that still have to be taken from the final run.

Two further points deserve attention when the system is compared with earlier work. The first is the cost of errors. In a payment setting a false negative, in which a malicious code is labelled Safe, is much more harmful than a false positive, which only costs the user a moment of extra checking. The confidence threshold and the rule-based indicators should therefore be tuned so that borderline cases are escalated to Suspicious or to needs-verification instead of being passed as Safe. The second is that the accuracy range reported for earlier systems in Fig. 5 depends on the dataset and the approach, so the figure illustrates a general trend and is not a controlled comparison on identical data.

The betting and gambling category also needs careful wording. A destination in this category is risky for a user because it can lead to financial loss, but it is not necessarily fraudulent, and the system therefore reports it as a risk category and not as proof of fraud. The same principle applies to every other indicator: the output is evidence for the user to weigh, not a verdict.

4.8 Analysis of Reliability and Practical Considerations

4.8.1 Expected Failure Modes

A QR code may fail to decode if the image is badly blurred, damaged, poorly lit, heavily rotated or partly covered. The system can also return an uncertain result when it has too little information. For example, a newly registered domain may have no reputation data, and a legitimate website with an unusual URL structure may trigger a warning. Other failure conditions include:

- No QR code detected in the uploaded image.
- Camera permission denied or camera unavailable.
- Incomplete or corrupted QR content.
- External threat-intelligence service unavailable.
- Unknown or newly registered domains.
- Insufficient features for reliable classification.

In these situations the system does not make claims it cannot support. It asks for a rescan, shows an uncertainty warning or falls back to a limited local analysis.

Table 10: Failure conditions and the corresponding system response.

Failure condition	System response
No QR code detected in the image	Ask the user to rescan or to choose another image
Camera permission denied or camera unavailable	Offer image upload as an alternative
Incomplete or corrupted QR content	Show a warning and request a rescan
External threat-intelligence service unavailable	Fall back to local analysis and state that the check was limited
Unknown or newly registered domain	Show an uncertainty warning and rely on the remaining indicators
Insufficient features for reliable classification	Mark the result as needing further verification

4.8.2 Ethical and Practical Considerations

QRISKSENTRA adds a layer of security, but it cannot guarantee that a transaction or website is completely safe. A classification depends only on the information available when the analysis is run, and an unfamiliar payment destination should not be called fraudulent without enough evidence. A betting, gambling or real-money gaming indicator marks a risk category; it does not on its own show that a destination is fraudulent. Wherever practical, QR images should be processed locally, sensitive payment information should not be stored unless necessary, and credentials for external threat-intelligence services must stay on the server and never appear in frontend code [10, 11]. The system also makes a clear distinction between local analysis and external threat-intelligence verification.

4.8.3 Security and Privacy Considerations

A security tool must not become a new source of risk. QRISKSENTRA therefore treats every decoded string as untrusted input. Content is validated and sanitized before it is analysed or displayed, so that a crafted payload cannot inject markup or script into the interface, and database access uses parameterized queries so that the payload cannot alter a query. These measures follow the general guidance of the OWASP Top 10 [11] for web applications. Secrets such as API keys for external services stay on the backend, and no credential appears in frontend code. Authentication and identity-related functions, where they are added, should follow established digital identity guidance [10].

Privacy is handled through data minimization. Wherever practical, QR images are processed locally, only the payment-related information that is actually needed is stored, and the user controls the retention of the

scan history through the delete function. The distinction between local analysis and optional external threat-intelligence verification is shown to the user, so that nobody is surprised by data leaving the device.

4.9 Deployment and Maintenance Considerations

A fraud detector is only as good as its currency. Once the system is deployed, the model has to be treated as a maintained component and not as a finished product. Each trained model should be stored together with its preprocessing configuration, feature definitions and normalization limits, and carry a version identifier, so that a result can always be traced to the exact model that produced it and so that a faulty release can be rolled back.

Operationally, a few simple indicators are worth monitoring: the share of scans that end in each class, the share of scans that fall below the confidence threshold, and the share that fail to decode. A sudden change in any of them can signal a new attack campaign, a change in the way codes are printed, or a defect in the decoder. New fraud patterns should be added to the dataset and to the rule set, and the model retrained and re-evaluated with the protocol of Section 4.4 before it replaces the running version.

On the usability side, too many warnings lead users to ignore them. The threshold for Suspicious and for needs-verification should therefore be set with the cost of both kinds of error in mind, and the reasons shown to the user should be short, concrete and free of technical jargon. Feedback from users, for example a report that a warning was unjustified, is a valuable source of new labelled samples as long as it is collected with the consent of the user and handled under the privacy rules described above.

4.10 Threats to Validity

Several factors limit how far the present results can be generalized. Regarding internal validity, the labels of the training samples determine what the model learns, so errors or inconsistencies in labelling feed directly into the classifier; removing duplicate and invalid samples before the split reduces, but does not remove, the risk that near-identical samples appear in both the training and the testing subsets. Regarding external validity, fraud patterns change over time and differ between regions and payment systems, so a model trained on one collection of samples may degrade when new campaigns appear. The small real-time sample adds to this uncertainty. Regarding construct validity, accuracy is only a coarse measure of usefulness in a setting where the classes are imbalanced and the cost of errors is asymmetric, which is why precision, recall, F1-score and the confusion matrix are reported as well. Regular retraining on newly identified patterns and a larger, more varied real-time test are the most direct ways of addressing these concerns.

V. CONCLUSIONS AND PROSPECTS FOR FUTURE INVESTIGATION

QRISKSENTRA is a real-time QR code safety and risk detection system meant to make digital payments and other QR-based interactions safer. It brings together QR scanning, content decoding, security feature analysis, machine learning and risk classification, built with React.js, TypeScript, Tailwind CSS, Python, FastAPI, OpenCV, PostgreSQL and a Random Forest classifier. Users can scan a QR code with the camera or upload an image, and they get a Safe, Suspicious or Risky classification with a risk score and relevant warnings. The system can help spot phishing links, suspicious payment destinations, malicious URLs and betting or gambling-related websites, although it cannot guarantee protection against every new threat.

Limitations. Model performance depends on how large and varied the training dataset is, and new fraud patterns will need regular updates. QR codes that are blurred, damaged, partly covered, poorly lit or shot at a bad angle may not decode, and external threat-intelligence services depend on network availability. A legitimate website may be flagged because of unusual characteristics, while a newly created malicious destination may not have enough information to be detected.

Future work. We plan to keep updating the dataset with newly identified fraud patterns, and to add more advanced phishing detection, domain reputation and redirect-chain analysis, real-time threat-intelligence integration, explainable AI, better decoding of low-quality QR codes, mobile application support, improved UPI/payment analysis [12] and real-time security alerts.

Several of the planned extensions address the limitations directly. Redirect-chain analysis and domain-reputation features would help with newly created malicious destinations that currently carry too little information. Explainable-AI techniques would make the contribution of each feature to a prediction easier to present. Better decoding of low-quality codes, for instance through stronger image enhancement, would reduce the number of failed scans, and a mobile application would bring the check closer to the moment of scanning. Finally, a larger and more varied evaluation with real Suspicious and Risky samples is needed to turn the present functional results into a reliable estimate of performance.

In practical terms, the main value of QRISKSENTRA lies in the moment between scanning and acting. By combining decoding, feature analysis, a Random Forest classifier and a readable explanation in one application, it gives users information that a plain QR scanner does not offer, while keeping the decision in their

hands. Its usefulness will depend on how well it is kept up to date with new fraud patterns and on how honestly its uncertainty is communicated.

REFERENCES

- [1]. M. H. Abdullah, A. A. Zainuddin, and N. A. Rahman, "QR Code Security and Privacy Issues in Digital Payment Systems," *International Journal of Advanced Computer Science and Applications*.
- [2]. L. Breiman, "Random Forests," *Machine Learning*, vol. 45, no. 1, pp. 5–32, 2001.
- [3]. F. Pedregosa et al., "Scikit-learn: Machine Learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.
- [4]. OpenCV, "Open Source Computer Vision Library," OpenCV Documentation.
- [5]. FastAPI, "FastAPI Documentation," Python Web Framework Documentation.
- [6]. PostgreSQL Global Development Group, "PostgreSQL Documentation," PostgreSQL Official Documentation.
- [7]. React, "React Documentation," React Official Documentation.
- [8]. ZXing, "ZXing: Multi-Format 1D/2D Barcode Image Processing Library," Open Source Project Documentation.
- [9]. M. Jakobsson and M. Myers, *Phishing and Countermeasures: Understanding the Increasing Problem of Identity Theft*, Wiley, 2006.
- [10]. National Institute of Standards and Technology, "Digital Identity Guidelines," NIST Special Publication 800-63.
- [11]. OWASP Foundation, "OWASP Top 10: The Ten Most Critical Web Application Security Risks," OWASP.
- [12]. UPI, "Unified Payments Interface: Digital Payment System," National Payments Corporation of India (NPCI).
- [13]. R. M. Mohammad, F. Thabtah, and L. McCluskey, "Predicting phishing websites based on self-structuring neural network," *Neural Computing and Applications*, vol. 25, no. 2, pp. 443–458, 2014.
- [14]. O. K. Sahingoz, E. Buber, O. Demir, and B. Diri, "Machine learning based phishing detection from URLs," *Expert Systems with Applications*, vol. 117, pp. 345–357, 2019.
- [15]. K. Krombholz, P. Frühwirth, P. Kieseberg, I. Kapsalis, M. Huber, and E. Weippl, "QR Code Security: A Survey of Attacks and Challenges for Usable Security," in *Human Aspects of Information Security, Privacy, and Trust (HAS 2014)*, Lecture Notes in Computer Science, vol. 8533, Springer, 2014, pp. 79–90.