**REAL TIME QR CODE DETECTION SYSTEM**



Session 2022-2026

**SUBMITTED BY:**

Noor Asghar 2022-CS-140

Ghania Khan Niazi 2022-CS-156

**SUPERVISED BY :**

Sir Samyan Qayyum Wahla

**Department of Computer Science**

**University of Engineering and Technology, Lahore,Pakistan**

**Table of Contents**

[**1. Introduction** 3](#_Toc185197883)

[**3. System Features** 8](#_Toc185197885)

[**4. External Interface Requirements** 13](#_Toc185197886)

[**5. System Attributes** 14](#_Toc185197887)

[**6. Other Non-Functional Requirements** 15](#_Toc185197888)

[**7. Appendices** 16](#_Toc185197889)

[**7. References** 17](#_Toc185197890)

# **1. Introduction**

**1.1 Purpose**

The purpose of this document is to provide a comprehensive **Software Requirements Specification (SRS)** for the **Real-time QR Code Detection System**. The system is designed to scan QR codes in real-time using a webcam, analyze the data contained within the QR code, and determine if the QR code is **malicious** or **benign**. Additionally, the system will provide information about the **source** of the QR code and display the **data** embedded within it.

The system aims to enhance security by alerting users about potentially harmful QR codes that could lead to phishing attacks, malware installation, or other security breaches. The solution will utilize image processing for real-time QR detection and employ machine learning or predefined rules for maliciousness detection.

**1.2 Scope**

The Real-time QR Code Detection System will include the following features:

* **Real-time scanning**: The system will continuously capture video from a webcam and analyze frames to detect QR/barcode codes in real time.
* **Maliciousness detection**: The system will classify the detected QR code as **malicious** or **benign**. This detection will be based on predefined malicious URL patterns, machine learning classification, or any heuristic rules.
* **Source and Data extraction**: Upon detecting a QR code, the system will extract the **source URL** or other embedded data (e.g., text, contact details) and display it to the user.
* **User Interface**: A graphical interface will be provided for the user to interact with the system, including features such as camera control, status display, and alerts for malicious QR codes.
* **Security Alerts**: The system will alert the user if a malicious QR code is detected, notifying the user about the potential threat.
* **Platform Support**: The system will be developed to work on common operating systems such as **Windows**, **Linux**, and **macOS**.

The boundaries of the project include:

* **Video stream capture**: The application will be restricted to using a webcam for real-time video capture and will not support other devices such as mobile phones or security cameras.
* **QR code format**: The system will primarily handle QR codes, and barcode detection is not part of the project unless required in future iterations.
* **Internet dependency**: Maliciousness detection may require an internet connection to verify URLs or fetch security threat databases.

**1.3 Definitions, Acronyms, and Abbreviations**

* **QR Code (Quick Response Code)**: A type of matrix barcode that encodes data such as URLs, contact information, or text.
* **Malicious QR Code**: A QR code that links to a harmful website or contains data designed to exploit vulnerabilities or deceive the user (e.g., phishing sites, malware downloads).
* **Benign QR Code**: A QR code that links to a legitimate and safe website or contains harmless data.
* **Webcam**: A small camera that connects to a computer to capture real-time video and images, used for scanning QR codes in this system.
* **Machine Learning (ML)**: A field of AI that allows the system to learn from data and make predictions or decisions based on patterns identified in data (used in maliciousness detection).
* **OpenCV**: An open-source computer vision and machine learning software library used for real-time image processing (used for detecting QR codes).
* **TensorFlow**: An open-source machine learning framework, which may be used to implement the malicious QR code detection model.
* **PyQt5**: A Python library for creating graphical user interfaces (used for the user interface of the QR code detection system).
* **URL (Uniform Resource Locator)**: The address of a resource on the internet (e.g., a website or a page).

**1.4 Overview**

This Software Requirements Specification (SRS) document provides detailed descriptions of the system's functionalities, performance requirements, design constraints, and other attributes. The document is divided into the following sections:

* **Section 2**: **Overall Description** – Provides an overview of the system, including its perspective, key features, operating environment, and design constraints.
* **Section 3**: **System Features** – Details the features and functionalities of the QR code detection and maliciousness classification system.
* **Section 4**: **External Interface Requirements** – Describes user interface specifications, hardware and software interfaces, and other system dependencies.
* **Section 5**: **System Attributes** – Outlines key system characteristics such as reliability, availability, security, and maintainability.
* **Section 6**: **Other Non-Functional Requirements** – Lists performance, scalability, and compatibility requirements.
* **Section 7**: **Appendices** – Includes any relevant appendices such as glossaries, acronyms, and system diagrams.

**2. Overall Description**

**2.1 Product Perspective**

The **Real-time QR Code Detection System** is a stand-alone security application designed to scan QR codes in real-time, detect whether the QR code is benign or malicious, and provide additional details about the QR code's source and data. This system fits within the broader context of cybersecurity by helping users identify and avoid security threats that may arise from malicious QR codes. The product aims to provide an intuitive and real-time interface that is user-friendly, making it accessible to both technical and non-technical users.

While the system can function independently, it can be integrated into larger security infrastructures as a tool for individual users or organizations. For example, it could be a useful feature in mobile security apps, workplace security systems, or other devices where QR code scanning is frequently required. The system may also be used in conjunction with other cybersecurity measures, such as phishing protection or malware detection tools, to provide a more comprehensive defense against online threats.

**2.2 Product Features**

The key features of the **Real-time QR Code Detection System** include:

* **Real-time Video Capture and Processing**: The system continuously captures video from a webcam or camera module and processes frames in real-time to detect QR codes.
* **QR/Barcode Detection**: The system can detect QR codes and barcodes from live video streams. It uses image processing algorithms to identify QR patterns and decode them into readable data.
* **Classification of Scanned Data**: Once a QR code is detected, the system processes the data it contains (e.g., URLs, contact information, plain text) and classifies it into benign or malicious categories.
* **Malicious QR Detection**: The system uses a predefined database of malicious URLs, machine learning models, or heuristic algorithms to classify QR codes as malicious if they link to harmful websites, phishing pages, or malware downloads.
* **User Interface with Real-Time Feedback**: A graphical user interface (GUI) will display real-time feedback from the system, showing the live camera feed, QR code information, classification results (benign or malicious), and alerts when a malicious QR code is detected.
* **Option to Start/Stop the Camera Feed**: Users can start or stop the live camera feed, allowing them to control when the system scans QR codes. This feature ensures that the camera is only active when needed, conserving system resources and enhancing user privacy.

**2.3 User Classes and Characteristics**

The system will be used by several classes of users, each with different levels of access and requirements:

* **General Users**: These are everyday users who need to scan QR codes for personal or work-related tasks. They may have limited technical knowledge and will interact with the system through the graphical user interface (GUI). They will use the system to check the safety of QR codes before interacting with them.
  + Characteristics: Basic technical knowledge, casual use, no special privileges.
  + Primary Actions: Scan QR codes, view scanned data, receive alerts, and take appropriate action.
* **Security Administrators**: Security administrators may use the system to scan QR codes in various environments, such as workplaces, organizations, or public areas. They have a higher level of technical knowledge and may require the ability to configure the system or integrate it into larger security protocols.
  + *Characteristics*: Advanced technical knowledge, access to system configuration options, troubleshooting.
  + *Primary Actions*: Configure the system, integrate with other security systems, monitor alerts, and manage database updates for malicious URLs.
* **Developers (Optional)**: Developers may be involved in extending or maintaining the system. They will have full access to the source code and technical aspects of the system.
  + Characteristics: Advanced technical skills, coding knowledge.
  + Primary Actions: Modify or extend system features, maintain codebase, update dependencies.

**2.4 Operating Environment**

The **Real-time QR Code Detection System** will operate in the following environments:

* **Hardware Requirements**:
  + **Camera**: A webcam or camera module capable of capturing video in real-time. The camera should ideally support a resolution of **720p or higher** to ensure accurate detection of QR codes.
  + **Computer**: A personal computer or laptop with a processor capable of handling real-time video processing. Recommended specs include at least **2 GB of RAM** and **1 GHz CPU**.
* **Software Requirements**:
  + **Operating System**: The system will be developed for compatibility with common operating systems, including:
    - **Windows** (Windows 7 or later)
    - **Linux** (Ubuntu 18.04 or later)
    - **macOS** (macOS 10.15 or later)
  + **Programming Languages**:
    - **Python 3.8+**: The main programming language for the system.
  + **Libraries and Frameworks**:
    - **PyQt5**: Used for creating the graphical user interface (GUI).
    - **OpenCV**: Utilized for real-time video capture and image processing, including QR code detection.
    - **TensorFlow or scikit-learn**: For implementing machine learning models that can classify QR codes as benign or malicious based on predefined patterns or learning.
    - **Pillow**: For image manipulation and QR code decoding.
    - **Requests/BeautifulSoup**: For fetching and parsing URLs for maliciousness analysis (optional for web-based maliciousness detection).
* **Third-Party Dependencies**:
  + If using external databases for malicious URL detection, access to these databases may be required.

**2.5 Design and Implementation Constraints**

The **Real-time QR Code Detection System** must operate within the following constraints:

* **Camera Resolution**: The accuracy of QR code detection can be influenced by the quality of the camera feed. Cameras with lower resolutions may affect detection performance. The system should be tested with at least **720p resolution** cameras, but support should be available for **1080p** and higher.
* **Real-time Processing**: The system must process video frames in real-time. This requires efficient image processing algorithms to ensure that QR codes are detected quickly, without noticeable lag. The system must be able to scan QR codes at a frame rate of **15 FPS or higher** for smooth user experience.
* **System Performance**: The system must run efficiently on typical hardware configurations without consuming excessive CPU or memory resources.
* **Maliciousness Detection Time**: The time required for maliciousness detection should not exceed **2 seconds** after QR code data is extracted, ensuring that users receive quick feedback.

**2.6 Assumptions and Dependencies**

* **Assumptions**:
  + The user has access to a functional webcam or camera module.
  + The user has an internet connection for downloading the maliciousness detection model, if required.
  + The QR code data is encoded in standard formats that can be decoded by the system.
* **Dependencies**:
  + **External Libraries**: The system depends on third-party libraries such as **OpenCV**, **PyQt5**, and **TensorFlow** for its implementation. These libraries must be compatible with the chosen operating system and Python version.
  + **Internet Connection**: If the system uses online databases or services to check the maliciousness of URLs, an internet connection will be required for those functionalities. Alternatively, an offline model could be developed, but it would require larger databases of malicious URLs.
  + **Hardware**: The system relies on the availability and quality of the webcam. The performance of QR detection may degrade on low-quality or incompatible came

# **3. System Features**

**3.1 QR Code and Barcode Detection**

**Description**:  
The QR Code and Barcode Detection feature allows the system to recognize and decode QR codes and barcodes from the live video feed. This feature is vital for the system to scan and read QR codes in real-time, extracting the data encoded within them, which could include URLs, contact information, or other structured data.

**Functional Requirements**:

* The system should continuously scan the video feed to detect QR codes and barcodes.
* The system must decode the QR data into readable format (e.g., URLs, text, contact information).
* The system should be able to handle QR codes in various orientations and lighting conditions.
* The system should display the extracted data on the user interface for the user to review.

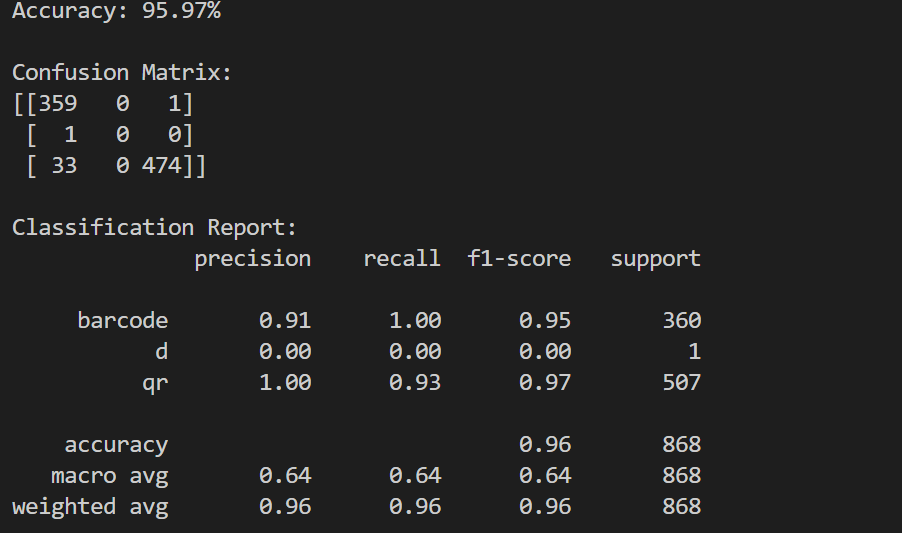
**User Interactions**:

* The user will not need to interact directly with this feature as it runs automatically once the video stream is initiated. The user will see detected QR code data on the display.

**Dependencies**:

* **OpenCV** for video processing and QR/barcode recognition.
* **Pillow** or similar image processing libraries to decode QR data.

The QR Code and Barcode Detection feature achieved an accuracy of **95.97%** during testing. The following evaluation metrics demonstrate the model's performance . It Include the results of the first model barcode\_qr\_model.h5

****

**3.2 Real-Time Video Stream**

**Description**:  
The system captures real-time video from the webcam or camera module to analyze the frames and detect QR codes. The video stream provides the live data source for QR scanning.

**Functional Requirements**:

* The system must continuously capture video in real-time at a frame rate of at least **15 FPS**.
* The system must display the live video stream within the user interface.
* The system must process the video frames efficiently for QR code detection.

**User Interactions**:

* The user clicks the **"Start Camera"** button to begin the live video feed.
* The user can stop the feed by clicking the **"Stop Camera"** button.

**Dependencies**:

* **OpenCV** for handling video input from the webcam.
* Hardware support for video capture (webcam).

**3.3 Malicious QR Detection**

**Description**:  
This feature checks if the scanned QR code links to a known malicious URL or contains harmful data. The system uses predefined rules, databases, or machine learning models to classify a QR code as either **benign** or **malicious**.

**Functional Requirements**:

* The system should analyze the QR code’s data (e.g., URL) and check it against a known list of malicious websites.
* The system should use machine learning or heuristics to classify unknown QR codes based on features such as URL domain, URL structure, and metadata.
* The system should alert the user if a malicious QR code is detected, providing details of the threat.

**User Interactions**:

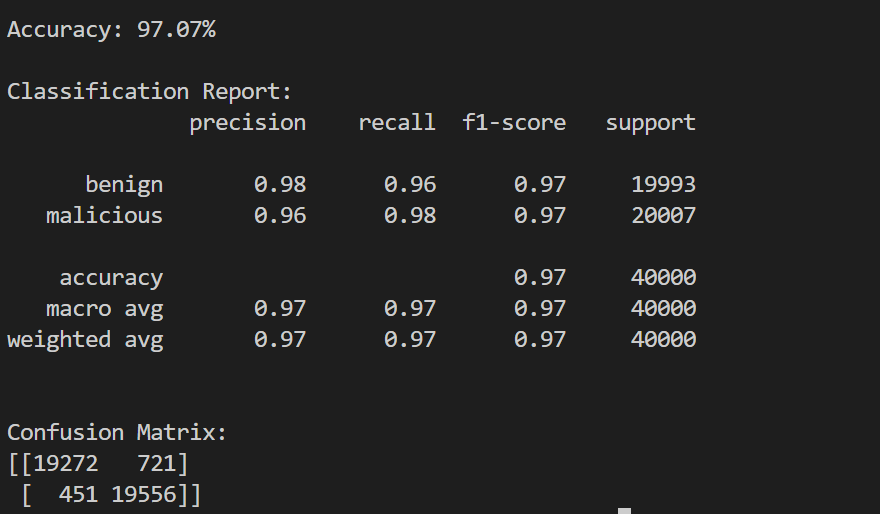
* The user does not need to interact directly with this feature. It operates automatically once a QR code is scanned.
* If the QR code is classified as malicious, the system will display a warning message on the screen and may provide options to either ignore the warning or take appropriate action (e.g., open a safe mode or report the URL).

**Dependencies**:

* **TensorFlow** or **scikit-learn** for machine learning-based maliciousness classification.
* **Custom database or third-party services** for maintaining a list of known malicious URLs.

The Malicious QR Detection feature achieved an accuracy of **97.07%**. The metrics below illustrate the model's ability to distinguish between benign and malicious QR codes:

Here is the results for the second model rf\_qr\_code\_model.pkl .



**3.4 Classification of QR Data**

**Description**:  
After a QR code is detected, the system classifies the data contained within it. The classification could be based on type (URL, contact, plain text, etc.) and the security classification (benign or malicious).

**Functional Requirements**:

* The system must categorize the data into predefined classes (e.g., URL, contact info, text).
* The system must cross-reference URLs with a database or security model to determine if they are benign or malicious.
* The system must display the classification result for the user to review.

**User Interactions**:

* The user will see the classification result immediately after the QR code is detected and decoded.
* The system will present the QR data and the security classification (benign or malicious) as part of the feedback.

**Dependencies**:

* **Machine learning models** for classifying URLs and QR code data.
* **External or internal databases** for malicious URL checks.

**3.5 User Interface**

**Description**:  
The user interface (UI) is where the user interacts with the system, including starting and stopping the camera feed, viewing QR data, and receiving alerts about malicious QR codes.

**Functional Requirements**:

* The UI should display a live video stream from the camera feed.
* The UI should display detected QR code data (e.g., URL, text).
* The UI should provide a **"Start Camera"** and **"Stop Camera"** button.
* The UI should show a notification or alert if a malicious QR code is detected.

**User Interactions**:

* The user can start or stop the video feed using the **"Start Camera"** and **"Stop Camera"** buttons.
* The user will view real-time feedback regarding the detected QR data.
* The user will receive an alert if a malicious QR code is detected.

**Dependencies**:

* **PyQt5** for developing the graphical user interface (GUI).

**3.6 Start/Stop Camera**

**Description**:  
This feature allows the user to control the start and stop functions of the webcam for scanning QR codes.

**Functional Requirements**:

* The system must allow the user to start the camera feed to begin scanning QR codes.
* The system must allow the user to stop the camera feed to conserve resources or halt scanning.

**User Interactions**:

* The user clicks the **"Start Camera"** button to activate the camera and begin scanning.
* The user clicks the **"Stop Camera"** button to stop the camera feed.

**Dependencies**:

* **OpenCV** for camera control.

**3.7 Exit Application**

**Description**:  
This feature allows the user to exit the application once they are finished using it.

**Functional Requirements**:

* The system must provide a way to safely exit the application, closing all processes and freeing resources.

**User Interactions**:

* The user clicks the **"Exit"** button to close the application.

**Dependencies**:

* **PyQt5** for managing the application window and exit process.

# **4. External Interface Requirements**

**4.1 User Interfaces**

**Design**:

* The main window will display a video feed window showing the real-time camera capture.
* Below the video feed, the extracted QR data will be displayed, including the decoded text or URL.
* The user will see a **"Start Camera"** button to initiate scanning and a **"Stop Camera"** button to end the video feed.
* A status area will show the classification of the QR code (benign or malicious), and any warnings or alerts will be displayed prominently.

**Layout**:

* The video feed occupies the majority of the screen, with buttons located beneath it for controlling the camera feed.
* QR data and classification results will be displayed in the lower section of the window.

**4.2 Hardware Interfaces**

* **Webcam**: The system requires a **webcam** or **camera module** for capturing video in real-time. The camera should support at least **720p** resolution for effective QR code detection.
* **Computer**: The system should run on a **PC or laptop** with a minimum of **2 GB RAM** and **1 GHz CPU** to handle video processing and classification tasks efficiently.

**4.3 Software Interfaces**

* **OpenCV**: Used for capturing and processing the real-time video feed.
* **PyQt5**: Provides the graphical user interface (GUI) framework for user interaction.
* **TensorFlow/scikit-learn**: Used for machine learning models to classify QR code data and detect malicious content.
* **Pillow**: For image decoding and processing.

**4.4 Communication Interfaces**

* **No communication protocols** are explicitly required for this system, unless using an external database for malicious URL checks, in which case **HTTP/HTTPS protocols** will be used to fetch data.

# **5. System Attributes**

**5.1 Reliability**

* The system should have **high reliability**, continuously processing video frames and detecting QR codes without errors or crashes.
* The QR detection and maliciousness classification should work reliably under normal usage scenarios, with minimal downtime.
* The QR Code and Barcode Detection model achieved an accuracy of **95.97%**, and the Malicious QR Detection model achieved an accuracy of **97.07%**. These results demonstrate the system's reliability in detecting QR codes and classifying them as benign or malicious with high precision.

**5.2 Availability**

* The system should be available for use at all times once installed, with minimal downtime for maintenance. It should run 24/7 without requiring constant manual intervention.

**5.3 Security**

* The system should **securely handle** sensitive data, especially when interacting with malicious URLs.
* Ensure that the camera feed is only active when the user chooses to start it, and should terminate the feed when stopped.

**5.4 Maintainability**

* The system should be modular, with separate components for video processing, QR code detection, and maliciousness detection, making it easy to update or maintain each component independently.

**5.5 Portability**

* The application should be compatible across different platforms, including **Windows**, **Linux**, and **macOS**.

# **6. Other Non-Functional Requirements**

**6.1 Performance Requirements**

* The system must operate in **real-time**, processing video frames at **15 FPS or higher**.
* **Maliciousness detection** should occur within **2 seconds** of QR code extraction to maintain a smooth user experience.

**6.2 Scalability**

* The system should be scalable to support additional features in the future, such as support for additional barcode formats, integration with cloud-based databases, or adding new maliciousness detection models.

**6.3 Usability**

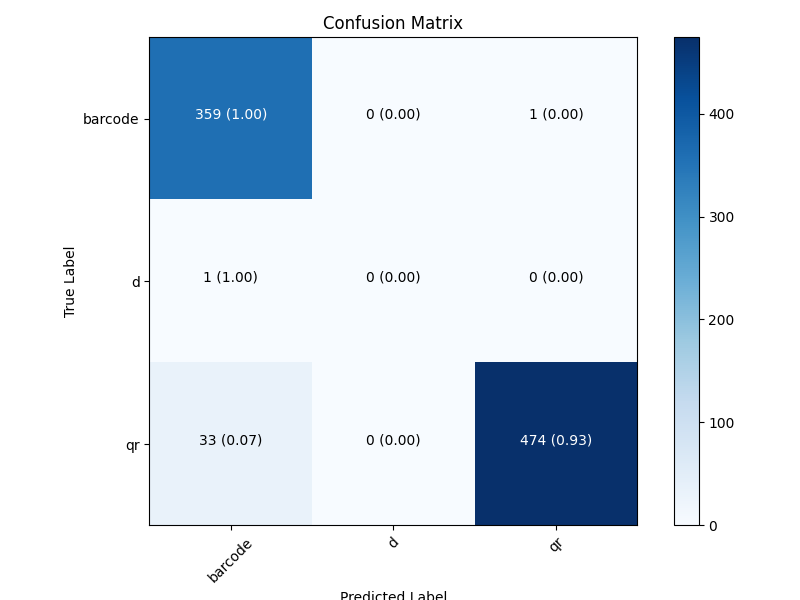
* The system should be intuitive and easy to use, with clear buttons and alerts for scanning, stopping the feed, and handling detected QR codes.

**6.4 Compatibility**

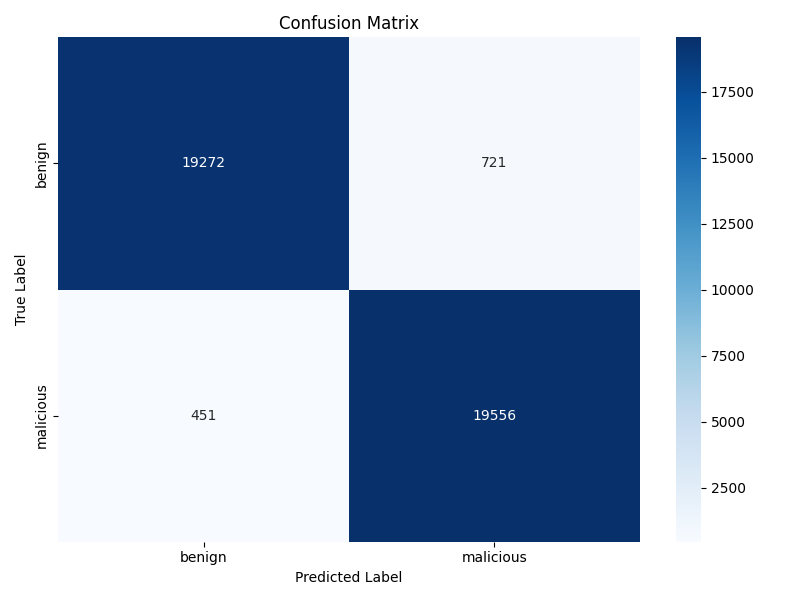
* The system should work seamlessly with standard webcams and be compatible with various operating systems, including **Windows**, **Linux**, and **macOS**.

# **7. Appendices**

**7.1 QR Code Detection Model Results of Confusion Matrix**

****

**7.2 Malicious QR Detection Model Results of Confusion Matrix**



# **7. References**

* **PyQt5 Documentation**: Official documentation for the PyQt5 library used to develop the graphical user interface of the system. https://www.riverbankcomputing.com/static/Docs/PyQt5/
* **OpenCV Documentation**: Official documentation for OpenCV, used for real-time image processing and QR code detection. https://opencv.org/documentation/
* **TensorFlow Documentation**: Documentation for TensorFlow, which may be used to implement a machine learning model for maliciousness detection. <https://www.tensorflow.org/>
* **QR Code Specification**: QR code standard documentation for understanding the structure and encoding of QR codes. https://www.iso.org/standard/62073.html