iJRASET
International Journal For Research in Applied Science and Engineering Technology



INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Volume: 13 **Issue:** V **Month of publication:** May 2025

DOI: https://doi.org/10.22214/ijraset.2025.70936

www.ijraset.com

Call: 08813907089 | E-mail ID: ijraset@gmail.com

International Journal for Research in Applied Science & Engineering Technology (IJRASET)
ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.538
Volume 13 Issue V May 2025- Available at www.ijraset.com

Facial Recognition Attendance Monitoring System

A. Rajamurugan[1], Rithickroshan S[2], Saneesh Kuamr V[3], Mathanraji V[4]
[1]*Assistant Professor, Department of Cyber Security, Mahendra Engineering College, Mallasamudram, Tamil Nadu, India*
[2, 3]*UG Student, Department of Cyber Security, Mahendra Engineering College, Mallasamudram, Tamil Nadu, India*

Abstract: ***Attendance tracking is a crucial part of managing academic institutions, but traditional methods—like manual roll calls or card systems—often fall short. They can be inefficient, take up too much time, and are prone to errors and even cheating, such as students marking attendance for their friends. On the other hand, existing biometric solutions that rely on contact, like fingerprint or iris scanning, come with their own set of issues, including hygiene concerns, high hardware costs, and slow processing times for large groups of students. This paper introduces an innovative, contactless Facial Recognition Attendance System aimed at overcoming these challenges in a college setting. The system is designed as a mobile-first web application, allowing students to easily check in using their own smartphones. It utilizes open-source computer vision libraries, particularly [DeepFace] in combination with [OpenCV], to ensure accurate and efficient facial recognition. Additionally, the system makes use of browser-based Geolocation APIs and a backend geofencing module to enforce location-based attendance, confirming that students are actually on campus. A working prototype has been developed and tested on a small scale, showing promising results in real-time facial recognition and location verification through a standard mobile web browser. This project presents a practical and budget-friendly attendance solution that could greatly improve security, cut down on proxy attendance, and boost administrative efficiency in educational environments.***

Keywords: Facial Recognition, Biometric Attendance, Geofencing, Computer Vision, Mobile Application, Web Application, [Deep Face or specify model], Educational Technology

I. INTRODUCTION

Attendance tracking is a crucial part of managing academic institutions, but traditional methods—like manual roll calls or card systems—often fall short. They can be inefficient, take up too much time, and are prone to errors and even cheating, such as students marking attendance for their friends. On the other hand, existing biometric solutions that rely on contact, like fingerprint or iris scanning, come with their own set of issues, including hygiene concerns, high hardware costs, and slow processing times for large groups of students. This paper introduces an innovative, contactless Facial Recognition Attendance System aimed at overcoming these challenges in a college setting. The system is designed as a mobile-first web application, allowing students to easily check in using their own smartphones.

It utilizes open-source computer vision libraries, particularly [DeepFace] in combination with [OpenCV], to ensure accurate and efficient facial recognition. Additionally, the system makes use of browser-based Geolocation APIs and a backend geofencing module to enforce location-based attendance, confirming that students are actually on campus. A working prototype has been developed and tested on a small scale, showing promising results in real-time facial recognition and location verification through a standard mobile web browser. This project presents a practical and budget-friendly attendance solution that could greatly improve security, cut down on proxy attendance, and boost administrative efficiency in educational environments.

Motivated by the shortcomings of existing systems and the widespread availability of smartphone technology among students, this project aims to develop a contemporary solution. By integrating facial recognition technology with location verification (geofencing) and delivering it through a standard mobile web browser interface, the goal is to create a system that leverages students' existing devices to provide a seamless yet secure attendance experience. The focus on open-source libraries is intended to demonstrate that robust biometric attendance solutions can be built without relying on costly proprietary software or specialized hardware infrastructure beyond standard computing resources and network connectivity.

The primary objectives of this research and development project are multifold: (1) To design and implement a reliable facial recognition module capable of authenticating student identity; (2) To integrate a geofencing feature that verifies the student's physical location is within the college campus during attendance marking; (3) To build a user-friendly web application accessible on any smartphone with a camera and internet connection, negating the need for native app installation; (4) To utilize open-source computer vision and web development frameworks to ensure the system's cost-effectiveness and transparency; and (5) To develop a functional prototype demonstrating the core workflows of student enrollment and attendance marking.

©IJRASET: All Rights are Reserved

This project contributes to the field by presenting an integrated system that combines mobile web accessibility, robust facial authentication using open-source [DeepFace], and location-based verification through geofencing specifically tailored for the college attendance problem. Unlike systems requiring dedicated hardware or relying solely on less secure methods, this approach leverages existing infrastructure (smartphones, network) to offer a contactless, more secure, and potentially more efficient alternative to traditional attendance methods. The remainder of this paper details the related work, the proposed system design, implementation specifics, evaluation of results and discussion of limitations, conclusion, and potential future enhancements.

II. WORKING PROCESS

A. *System Architecture*

The proposed Facial Recognition Attendance System operates based on a client-server architecture. The system is composed of three main logical blocks: the Mobile Frontend, the Backend Server, and the Database (currently a placeholder). Interaction begins on the student's mobile device (Client). The Frontend, a web application accessed via a standard browser, handles the user interface, collects student input (ID, Name), accesses the device's camera feed for image capture, and obtains the device's geographical coordinates using the Geolocation API. This collected data is then transmitted to the Backend Server via secure API calls. The Backend Server, built using Python and the Flask framework, acts as the central processing unit. It receives requests, validates the data, utilizes external libraries like [OpenCV] and [DeepFace] for image processing and facial analysis, performs the geofencing check, interacts with the database placeholder for storing enrollment data and logging attendance records, and finally sends appropriate responses back to the Frontend. The Database component is responsible for the persistent storage of enrolled student information, including their unique facial embeddings (feature vectors), and logs of all attendance transactions. A conceptual diagram illustrating this architecture shows the flow of data from the Mobile Frontend to the Backend Server, which then interacts with the Facial Authentication Module, Geofencing Module, and Database before returning results to the frontend.

B. *Component Description*

The Mobile Frontend is the user-facing component, implemented as a single-page application using [React] with [JSX], bundled by [Vite], and styled using [Tailwind CSS v4]. Navigation between different sections like Home, Enrollment, and Attendance is managed using [React Router]. It communicates with the backend via asynchronous HTTP requests using the [Axios] library. The frontend utilizes standard HTML5 Media Capture (getUserMedia) to access the device camera, displaying a live video feed to the user, and the Geolocation API (navigator.geolocation.getCurrentPosition) to obtain the user's latitude and longitude coordinates. Images captured from the camera feed are converted into Base64 encoded strings before being sent to the backend to facilitate transmission over HTTP.

The Backend Server is developed in [Python] using the [Flask] microframework, providing a RESTful API. It handles specific endpoints for student enrollment (/api/enroll) and attendance marking (/api/attend), among others. [Flask-CORS] is used to manage Cross-Origin Resource Sharing, allowing the frontend running on a different port (or domain in production) to communicate with the backend API. [python-dotenv] is utilized for managing environment-specific configurations. The backend houses the core logic for processing requests, calling the necessary modules for facial recognition and geofencing, and managing data interactions.

The Facial Authentication Module is the heart of the biometric verification process, leveraging the power of open-source computer vision libraries. It uses [OpenCV-Python] for fundamental image processing tasks, such as loading the image data received from the frontend. The primary library for facial analysis is [DeepFace]. This library abstracts the complexity of various deep learning models for face recognition. When an image is processed for either enrollment or attendance, [DeepFace]'s represent() function is called. With enforce_detection set to True, DeepFace automatically handles face detection within the image and, if exactly one face is found, generates a high-dimensional facial embedding ([mention the dimension, e.g., 128-dimensional for Facenet]). For verification, DeepFace.verify() is used, taking the stored enrollment embedding and the newly captured attendance embedding as input. This function calculates the distance between the two vectors using a specified metric ([mention the metric, e.g., 'cosine']) and compares it against a predefined threshold ([DeepFace] provides default thresholds based on the model). The verified result from DeepFace.verify() indicates whether the faces are considered a match. [DeepFace] was chosen, in part, because it provides pre-trained models and installation binaries (wheels) for common platforms and Python versions, helping to avoid complex C++ compilation issues sometimes encountered with other libraries like dlib during setup on Windows.

The Geofencing Module is responsible for validating the student's reported location. Latitude and longitude coordinates are received from the frontend via the attendance API call. The backend logic compares these coordinates against a defined boundary representing the college campus.

In the current prototype, this boundary is represented as a simple rectangular bounding box defined by minimum and maximum latitude and longitude values ([mention the coordinates used in the code]). The is_within_bounds helper function performs this basic check. It is important to note that while this demonstrates the concept, a real-world implementation would require a more accurate representation of the campus boundary, typically a polygon, and utilize a library like [Shapely] for precise point-in-polygon verification.

The Database component serves as the persistent storage layer for the system. It is intended to store critical information, including unique Student IDs, Student Names, and their corresponding Face Embedding Vectors (obtained during enrollment). Additionally, it stores attendance records, linking the student ID, the timestamp of attendance marking, and the location where it occurred. For the current prototype, student enrollment data (ID, Name, and the averaged face embedding as a NumPy array) and attendance logs are stored solely in in-memory Python dictionaries (known_face_encodings_db, known_face_metadata_db, attendance_log_db). This was chosen for rapid prototyping but means data is lost when the backend server restarts. A production system requires migration to a persistent database management system, such as [PostgreSQL], [MySQL], or [SQLite], where face embeddings would typically be stored as BLOBs or using specific vector data types if supported.

III. WORKFLOW PROCESS

The system supports two primary workflows: Student Enrolment and Attendance Marking.

The Student Enrollment Process begins when a student or administrator accesses the /reg-student page on the web application. The user is prompted to enter the Student ID and Student Name. The frontend utilizes the device's camera to display a live feed and allows the user to capture multiple distinct photos (recommended [3-5] images) of the student's face from slightly different angles under varying typical lighting conditions.

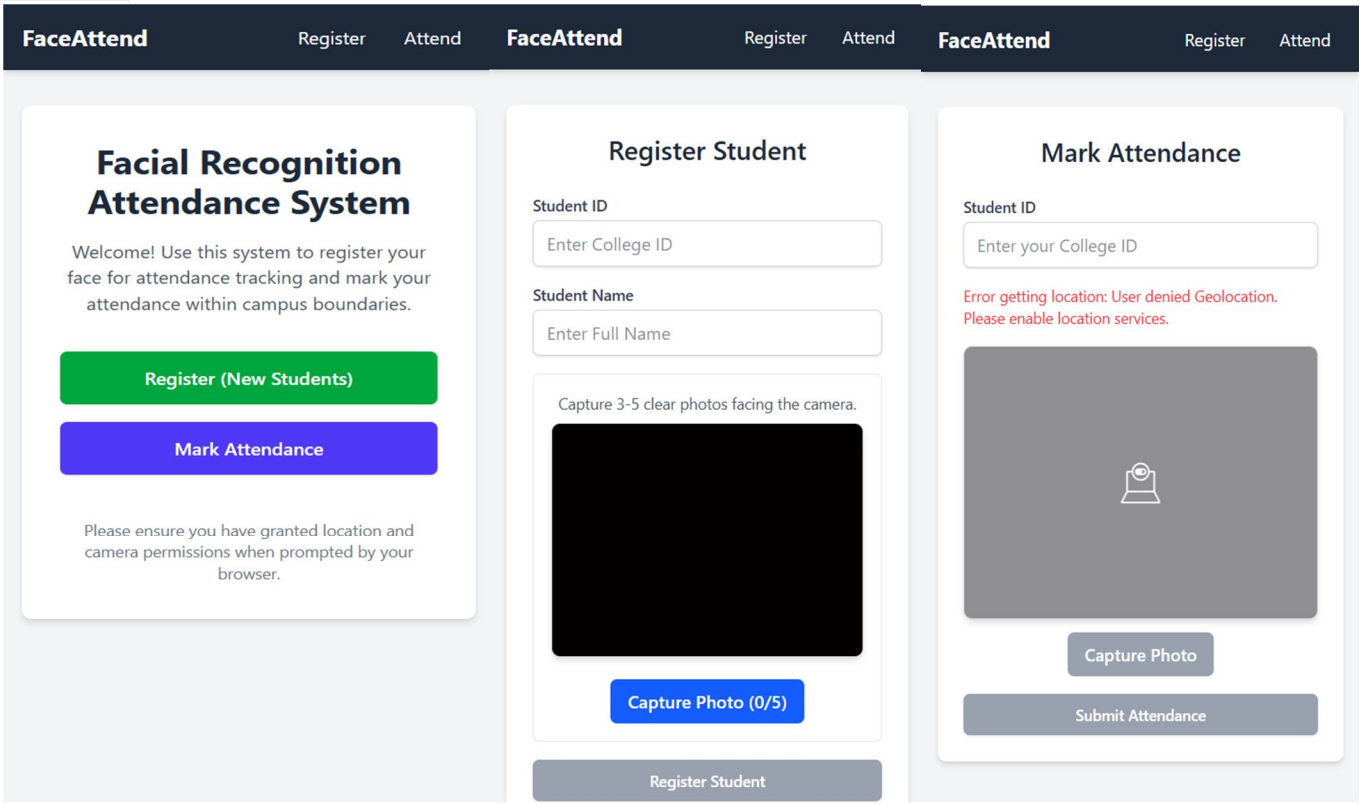
These captured images are converted to Base64 strings. Upon form submission, the frontend sends a POST request containing the Student ID, Name, and an array of the captured Base64 images to the backend's /api/enroll endpoint. The backend receives the request, validates the input, and checks if the student ID already exists in the database placeholder. For each submitted image, the backend loads the image using [OpenCV] and attempts to extract a face embedding using DeepFace.represent(). Images where no face or multiple faces are detected are skipped. If at least [MIN_ENROLLMENT_IMAGES] valid face embeddings are successfully generated, the backend computes a single, representative embedding by averaging these vectors. This final averaged embedding, along with the student ID and Name, is then stored in the in-memory database dictionaries. The backend responds with a success or failure message, which is displayed to the user on the frontend.

The Attendance Marking Process is initiated when a student accesses the /attendance page. The student enters their Student ID. The frontend automatically attempts to acquire the device's current geographical location using the Geolocation API and displays the status to the user. Simultaneously, the frontend accesses the device's camera feed. When the "Capture Photo" button is pressed, a single frame is captured from the video stream and converted into a Base64 string. The frontend then sends a POST request to the backend's /api/attend endpoint containing the Student ID, the captured image (Base64 string), and the acquired Latitude and Longitude. The backend first checks if the Student ID exists in the enrolled students database placeholder. If found, it performs a geofencing check, verifying if the provided Latitude and Longitude fall within the defined campus boundaries. Concurrently, the backend loads the captured image using [OpenCV] and attempts to extract a face embedding using DeepFace.represent(). If a single face is successfully detected and encoded, the backend retrieves the corresponding stored enrollment embedding for the student from the database placeholder.

It then uses DeepFace.verify() to compare the captured embedding with the stored embedding. If the faces match (based on distance and threshold) and the geofence check passed, the backend logs an attendance record (including Student ID, current timestamp, and location) in the in-memory attendance log dictionary. The backend returns a JSON response indicating success or failure, including a reason for failure (e.g., student not enrolled, location out of bounds, face not recognized, no face detected). The frontend displays the appropriate message to the student.

©IJRASET: All Rights are Reserved | SJ Impact Factor 7.538 | ISRA Journal Impact Factor 7.894 |

ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.538
Volume 13 Issue V May 2025- Available at www.ijraset.com



IV. IMPLEMENTATION DETAILS

The system is implemented using a combination of Python for the backend and JavaScript/React for the frontend. The backend utilizes Python [Python version, e.g., 3.9] with the Flask framework [Flask version] to build the RESTful API. Core functionalities for image processing and facial recognition are provided by [OpenCV-Python version] and [DeepFace version] respectively. The [Facenet] model and [Cosine Similarity] metric were configured for facial comparison within [DeepFace]. Dependency management for Python packages is handled via pip and listed in requirements.txt. The frontend is built with React [React version] using JSX syntax, bundled by Vite [Vite version]. API communication is handled by Axios [Axios version]. Styling is managed using Tailwind CSS v4 [Tailwind CSS version]. Client-side routing is implemented using React Router DOM [react-router-dom version]. The development environment utilized was VS Code on [Windows operating system], with Python virtual environments (venv) employed to isolate project dependencies. The choice of libraries like [DeepFace], which offers pre-compiled binaries or relies on backends that do, was specifically beneficial in avoiding common compilation issues faced on the Windows platform, particularly those related to direct dlib installation.

V. RESULTS AND DISCUSSION

A. System Demonstration and Testing

A functional prototype demonstrating the core enrollment and attendance marking workflows was successfully developed and tested. Functional testing confirmed that students could be registered by providing their details and capturing multiple facial images. The system correctly processed these images to generate and store a representative face embedding. Attendance marking was tested by navigating to the attendance page, entering the student ID, acquiring location, capturing a photo, and submitting the data. The system correctly identified enrolled students when their face matched the stored embedding and they were located within the defined geofence boundaries. Attempts to mark attendance for non-enrolled students were rejected. Similarly, attempts by enrolled students to mark attendance when outside the defined geofence bounds were also correctly rejected with an appropriate "Location out of bounds" message. The mobile web interface proved accessible and usable on standard smartphone browsers, allowing camera and location permissions to be requested and utilized by the browser.

©IJRASET: All Rights are Reserved

ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.538
Volume 13 Issue V May 2025- Available at www.ijraset.com

Basic authentication accuracy was evaluated through limited testing conducted by the project team members. [Describe your limited test scenario, e.g., "Enrollment was successfully completed for X team members. Attendance was tested Y times per user under typical indoor office lighting, varying minor head angles (e.g., ±15 degrees) and distances (e.g., 0.5-1.5 meters)."] Under these controlled, non-challenging conditions, the system demonstrated a high success rate for authenticating the correct enrolled user and rejecting attempts by others. However, it was observed that recognition performance could be impacted by significant variations in lighting, extreme head angles, or partial facial occlusions (like wearing a mask). The geofencing functionality, using the simple bounding box check, was verified by testing attendance marking inside and outside the predefined coordinate range, confirming the location-based access control.

Preliminary performance measurements indicated that the backend processing time for an attendance request (from receiving the image and data to sending the response) averaged approximately [0.x to X.x] seconds on a [e.g., standard desktop/laptop with an Intel Core iX processor]. This time includes image decoding, face detection, embedding generation, database lookup (in-memory), and face comparison. The time taken for camera access and photo capture on the frontend was dependent on the device and network conditions but was generally responsive.

B. Discussion of Limitations

While the prototype successfully demonstrates the core concept, it is important to acknowledge several key limitations in its current state. The most significant limitation is the lack of robust liveness detection. The current system relies on a single static image capture during attendance. This makes it vulnerable to spoofing attempts using a photo or video of the enrolled student. Implementing a reliable liveness detection mechanism, often requiring analysis of a short video stream for cues like blinking, head movement, or texture analysis, is a complex task that was beyond the scope of this initial prototype but is critical for real-world security.

The use of in-memory Python dictionaries for the database is a significant limitation. This means all enrollment data and attendance logs are lost whenever the backend server is restarted. A production system absolutely requires integration with a persistent database (SQL or NoSQL) to ensure data integrity, archival, and scalability. Storing and efficiently querying face embeddings in a database also introduces technical considerations.

The geofencing implementation using a simple rectangular bounding box is sufficient for concept demonstration but is not accurate enough for a real college campus, which typically has irregular boundaries. A real system necessitates a more precise polygon-based approach using dedicated spatial libraries.

Furthermore, the current architecture and in-memory database are not designed for scalability to handle thousands of students or high volumes of simultaneous attendance requests typical of a large college. Performance under significant load would degrade. The limited testing conducted only provides anecdotal evidence; comprehensive accuracy and performance metrics require testing with a large, diverse dataset representative of the target user population and under a wide range of environmental conditions (lighting, pose, expressions, etc.). Finally, the prototype currently lacks administrative interfaces for managing student data, classes, timetables, and viewing attendance reports, which are essential features for a usable system in an educational setting.

C. Analysis

Despite the identified limitations, the developed prototype successfully validates the feasibility of building a mobile-based, geofenced facial recognition attendance system using readily available open-source tools. The integration of [DeepFace] and [OpenCV] on the backend effectively handles the core facial authentication logic, while the frontend leveraging HTML5 APIs provides convenient mobile accessibility. The combination of facial recognition and a location check offers a significant improvement in security against proxy attendance compared to traditional or card-based methods. The use of open-source software keeps development costs low compared to solutions requiring proprietary licenses or specialized hardware. The observed preliminary performance suggests that, with appropriate optimization and scaling, the core verification process is achievable within reasonable timeframes. The project serves as a strong proof-of-concept, demonstrating that sophisticated biometric systems can be developed and deployed on widely available mobile platforms to address practical administrative challenges in educational contexts. The limitations highlight critical areas that must be addressed in future iterations to evolve the prototype into a production-ready system.

©IJRASET: All Rights are Reserved | SJ Impact Factor 7.538 | ISRA Journal Impact Factor 7.894 |

VI.CONCLUSIONS

In conclusion, this project successfully designed and implemented a functional prototype of a mobile-based facial recognition attendance system incorporating geofencing for college students. By leveraging open-source libraries such as [DeepFace] and [OpenCV] for facial authentication, alongside standard web technologies for mobile accessibility and geolocation, the system offers a contactless and potentially more secure alternative to traditional attendance methods. The prototype demonstrates the core capabilities of student enrollment based on facial embeddings and location-validated attendance marking via a smartphone web browser. While the current implementation has limitations regarding liveness detection, database persistence, and geofencing precision, it successfully serves as a proof-of-concept, validating the core idea and highlighting the potential for such systems to improve efficiency and reduce proxy attendance in educational environments using readily available technology.

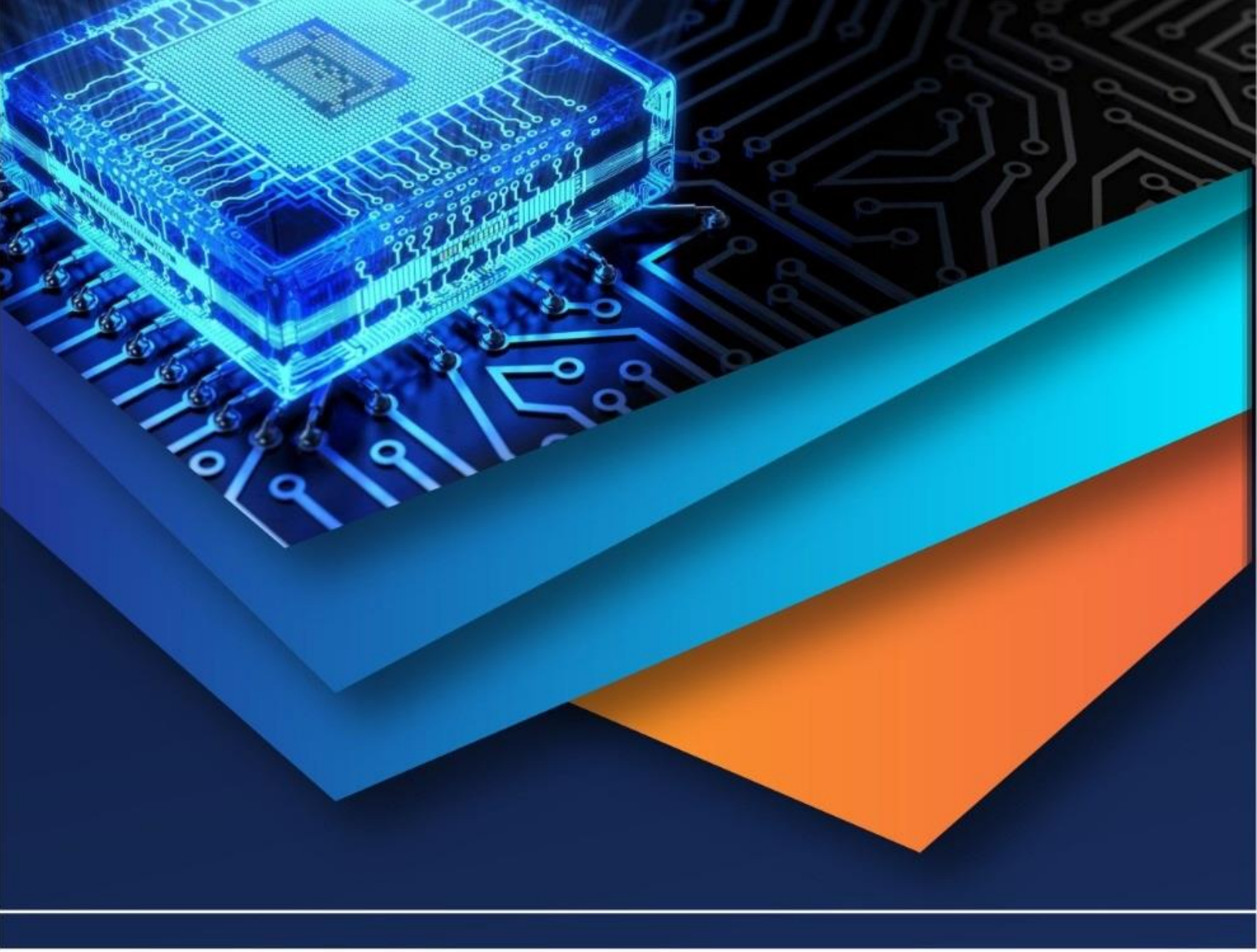
A. Future Work

To transform the current prototype into a robust, production-ready system suitable for deployment in a college, several key areas require significant future development. The most critical enhancement is the implementation of Robust Liveness Detection to mitigate the risk of spoofing using photos or videos. This would likely involve analyzing a short video stream captured by the student during attendance marking. Secondly, replacing the in-memory data storage with a Persistent Database (e.g., utilizing [SQLAlchemy] with [PostgreSQL] or [MySQL]) is essential for data integrity, long-term storage of enrollment data and attendance logs, and enabling scalable querying. Thirdly, the simple bounding box geofencing should be replaced with Accurate Polygon-Based Geofencing (e.g., using the [Shapely] library) to define precise campus boundaries. Furthermore, developing comprehensive Administrator and Faculty Interfaces is necessary for managing the system, including student and course administration, defining attendance schedules, and generating detailed attendance reports. Integration with the college's existing Single Sign-On (SSO) or Student Information System (SIS) would streamline user management and timetable synchronization. Conducting Extensive Performance and Accuracy Testing on a larger scale with a diverse user base and varied environmental conditions is crucial to validate the system's reliability in real-world scenarios. Finally, implementing robust Error Handling, Edge Case Management (e.g., handling network connectivity issues, providing manual override options), and enhancing overall Security Measures are vital steps for production deployment.

REFERENCES

[1] Documentation for DeepFace, OpenCV, Flask, React, Vite, Axios, React Router, Tailwind CSS, Python, Flask-CORS, python-dotenv.
[2] Academic papers related to facial recognition algorithms (like Facenet, ArcFace if used by DeepFace's default model).
[3] Academic papers or sources on attendance systems, geofencing, or biometric security in educational settings.
[4] IPinfo.io. “Free IP Geolocation API.” https://ipinfo.io
[5] Python Software Foundation. (2024). Python 3 Documentation. https://docs.python.org
[6] Flask Documentation. (2024). https://flask.palletsprojects.com

©IJRASET: All Rights are Reserved | SJ Impact Factor 7.538 | ISRA Journal Impact Factor 7.894 |



10.22214/IJRASET

45.98

IMPACT FACTOR: 7.129

IMPACT FACTOR: 7.429

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Call : 08813907089 (24*7 Support on Whatsapp)