Intelligent Automated Attendance System Using Real Time Facial Recognition

MUHAMMAD NAUMAN
01-136221-016

ALI ABBAS HUSSNAIN
01-136221-032

Supervisor: Ms. Nabia Khalid

Bachelor of Science in Artificial Intelligence

Department of Computer Science
Bahria University, Islamabad

October 2025

Certificate

We accept the work contained in the report titled "Intelligent Automated Attendance System Using Real Time Facial Recognition", written by Mr. Muhammad Nauman AND Mr. Ali Abbas Hussnain as a confirmation to the required standard for the partial fulfillment of the degree of Bachelor of Science in Computer Science.

Approved by:

Supervisor: Ms. Nabia Khalid

_____

Internal Examiner: Dr. [Name]

_____

External Examiner: Dr. [Name]

_____

Project Coordinator: Dr. Prof. Sohail Akhtar

_____

Head of the Department: Dr. [Name]

_____

Abstract

Bahria University, until now, has used primitive methods for attendance, such as; roll calls and paper signatures. Attendance tracking might sound like a simple and mundane task, but it is plagued by many errors which are surprisingly tricky to get right with the traditional methods being used. The idea of an Intelligent Object Tracking System with a Pan-Tilt camera for the University is presented in this thesis. Instead of physical sign sheets or simple biometric scanners, the projected system allows real-time facial recognition that is non-intrusive and unbiased to external conflicts. The system combines MTCCN face detection with ResNet embeddings with two separate user interfaces, making it a dual-interface architecture. A supreme administrative control is provided by a Streamlit dashboard while monitoring is influenced with the help of a Flask server. Live video feeds can be tracked and monitored by the instructors over local networks without the need of special equipment or the need to run back to administrative work stations. A PostgreSQL database makes it easy to manage the attendance records and support any needed analysis. The system hit a 97.2% of accuracy during its trial testing.

Acknowledgments

We are profusely grateful, and absolutely humbled in the name of Allah Almighty, the most Gracious and Merciful, we have the strength, ability, and clarity to be able to complete this project, as nothing at all will happen without his permission. Besides, we also want to express our utmost gratitude and respect to our brilliant supervisor, Ms. Nabia Khalid who not only assisted us with her brilliant knowledge, but also with her constant patience, direction and useful feedback. Her assistance and recommendations proved us very helpful to solve any technical issue, to make our system more stable, and our approach more effective. We are also grateful to Prof. Sohail Akhtar who is our Project Coordinator who kept us organized. and making us keep in line with our milestones. With his contribution, the whole process became easier. and more manageable. We also owe our debt of gratitude to the Department of Computer Science at Bahria University. equipping the lab facilities and the computational resources that were necessary in training and testing our models. Finally, we would like to mention our families and friends who were always so supportive and understanding of this process, and our friends who helped us to spend the long nights doing coding. and problem-solving

Contents

List of Figures

List of Tables

List of Abbreviations

AI	Artificial Intelligence
CNN	Convolutional Neural Network
FYP	Final Year Project
ML	Machine Learning

Chapter 1

Introduction

1.1 Overview

It not only helps track student performance but also helps to determine their exam eligibility, subsequently providing reliable records that can aid in institutional decision-making and audits. Despite the fact that it is useful and important, many of the classes still use traditional methods for paper sign in sheets or roll calls. These methods are easy to manipulate and can also hinder the learning flow and lectures significantly. For instance, we have a common example of buddy punching where one student marks the attendance of his friend even if he does not attend the class. Such collected data remains bound within files without any appropriate analysis. These are the long-standing issues that this project presents in Intelligent Automated Attendance System. By bringing Deep Learning and Computer Vision together, it makes the attendance workflow completely automated. The process runs without student intervention or any new instructions every time by detecting the faces of attendees and storing them into final records.

1.2 Problem Statement

Despite the fact that many digital tools for attendance are easily accessible at the Bahria University and other similar institutions, several problems still persist with credibility: Time Consumption: In a 90-minutes class, roughly the estimate of about 50 students take up to 8 to 12 minutes for basic attendance. This leads the teachers to lose credible time throughout the course of the entire semester :

1. **Time Consumption:** In a 90-minutes class, roughly the estimate of about 50 students take up to 8 to 12 minutes for basic attendance. This leads the teachers to lose credible time throughout the course of the entire semester.

2. **Lack of Authenticity:** Signatures can be forged and RFID cards can be shared among friends. There is currently no foolproof way to ensure that a student's physical presence matches their recorded identity without adding extra verification steps.
3. **Delayed Administrative Updates:** Attendance data is often typed into the LMS manually which causes delays. As a result patterns of absenteeism might go unnoticed until weeks later reducing the institution's ability to support struggling students.
4. **Inflexible Monitoring Systems:** Many automated systems require instructors to stay at a fixed workstation. Mobile friendly solutions for realtime monitoring inside the classroom are scarce.

1.3 Motivation

First, modern neural networks run efficiently on consumer hardware through GPU acceleration. This makes high-quality face recognition practical and cost-effective.

Second, institutions are increasingly interested in software-driven biometric tools that do not depend on pricey hardware. By relying on regular webcams and open-source libraries such as Python and OpenCV, universities can automate attendance without major financial investment.

Third, mobile monitoring requires separating the interface from processing hardware. A Flask-based streaming server enables device-independent access. This approach keeps the system lightweight, flexible, and accessible from any device on the local network.

1.4 Project Objectives

The overarching aim of this project is to create a dependable and unobtrusive automated attendance system. Following objects are set to accomplish this:

- **High Accuracy Recognition:** A FaceNet based recognition model can be implemented which targets the accuracy of up to 95 percent within structured classroom settings.
- **RealTime Region Filtering:** A filtered program whose task is limited to recognizing the students when they enter the classroom and filtering out any background activity.
- **Dual Interface Design:** Two interfaces can be developed: a streamlined Flask web application for mobile real-time monitoring and the Streamlit dashboard for configuration.

- **Reliable Data Storage:**A PostgreSQL database designed in third normal form (3NF)can be used to ensure that attendance records are secure, clean, and free of duplicates.

1.5 Significance of the Study

There are several groups within the academic environment who can benefit from this system:

- **Administration:** The administration would have access to audit-proof attendance lists,which are easily verifiable and where the risk of manipulation is minimal.
- **Faculty:** It saves valuable teaching time and reduces repetitive administrative work.
- **Students:** Ensures a fairer environment by eliminating the possibility of proxy participation.
- **Developers:**Provides a practical example of combining synchronous interfaces like streamlit with asynchronous Flask services in a multithreaded Python environment.

1.6 Scope of the Project

The project includes specific features and acknowledges several limitations:

1.6.1 Inclusions

- **Face Detection and Recognition:** It uses MTCNN for face recognition and ResNet34 for generating face embeddings.
- **Live Video Streaming:** Offers MJPEG based live streaming over the local network to mobile devices.
- **Reporting Tools:** Supports the export of attendance records in CSV and Excel format for academic analysis.
- **Hardware Support:** Compatible with standard USB webcams;all tests were performed using the Logitech C920.

1.6.2 Exclusions

- **Emotion or Behavior Analysis:** The system merely confirms the identity and does not interpret facial expressions or behavioral signals.
- **Cloud Deployment:** To maintain confidentiality, the system is restricted to the local intranet and is not used on public cloud platforms.

- **Physical PanTilt Movement:** A permanently installed wide-angle camera is used; possible improvements to the mechanical tracking are examined in Chapter 8.

1.7 Thesis Organization

The structure of this report is as follows: **Chapter 2** reviews facial recognition literature. **Chapter 3** outlines system requirements. **Chapter 4** presents the system architecture and database design. **Chapter 5** details implementation. **Chapter 6** presents evaluation results. **Chapter 7** analyzes findings. **Chapter 8** proposes Raspberry Pi-based edge deployment. **Chapter 9** summarizes contributions.

Chapter 2

Literature Review

2.1 Introduction

Biometric authentication has developed over time from just being a hardware-heavy system to being a software-driven solution. This review focuses on a review of facial recognition that is developed from basic geometric approaches and evolving into modem convolutional neural networks. This chapter would emphasize on development of facial recognition by taking a closer look on early geometric approaches that led to modern high-performance Convolutional Neural Networks.

2.2 The Evolution of Biometric Systems

Biometric systems work in a complex way by basing individuals over distinctive behavioral and physical traits. Educational systems usually rely on basic identification systems like student and faculty ID cards and handwritten signatures for attendance. However, new and former decades of research has improvised on the limitations these methods carried – for instance, cards being misplaced, handed to a friend, or even traded leading to false attendance. Facial recognition, in these terms, offer a turning point. Without any physical contact, identity verification is done by just the use of commodity cameras, making it a far more reliable and hygienic alternative. It's use and adoption grew wildly during the Covid-19 pandemic, where such systems were incorporated in setups to minimize the spread of disease across various systems. Over time, as the institutions were reopened, they became a useful modem within the technology-driven setups.

2.3 Historical Approaches to Face Recognition

2.3.1 First Generation: Geometric and Holistic Approaches

The earliest facial recognition wave was developed between the 1970s and 1980s which relied majorly on geometric feature matching. The systems focused on the key facial landmarks – like eye distance, shape of jawline, a special mark – and compared them with others by using measures such as, Euclidean distance. The limitation to this was that these methods failed whenever the individual changed their expressions or direction. The major leap in knowledge was the introduction of Eigenfaces by Turk and Pentland (1991), They used tools like Principle Component Analysis (PCA), which enabled their holistic model to enhance compressed facial images into a lower-dimensional space. However, again to the limitation, the results were varied depending on the lightening and the system performed poorly in low light.

2.3.2 Second Generation: Local Feature Descriptors

By the end of the 1900s and during the early 2000s, the research shifted towards texture-based, local descriptors like Histogram of Oriented Patterns (HOP) and Local Binary Patterns (LBP). The technique extracted local texture features and examined small patches for identification rather than analyzing the entire face at once. HOG becomes a top choice for face detection and pedestrian when combined with SVM classifiers. However, these handcrafted features do not work well under occlusions and low lightening.

2.4 State of the Art: Deep Convolutional Neural Networks

2.4.1 The CNN Paradigm Shift

Convolutional Neural Networks (CNNs) do not require hand-engineered features since they achieve hierarchal representations of large data sets automatically. They possess a stratified structure that enables them to detect substantiated patterns which is much more than the previous feature-based approaches.

2.4.2 Multi-task Cascaded Convolutional Networks (MTCNN)

MTCNN was chosen for this system because it is reliable and superior in performance when it comes to face recognition and accurate face detection. In comparison to old detectors like the Haar Cascades, MTCNN actually has a three sequence stages for face detection. These are:

1. **P-Net:** It generates box proposals which are preliminary bounding.

2. **R-Net:** It refines bounding boxes and also filters out any inaccurate proposals that might exist.
3. **O-Net:** Provides with not only a final classification but also essential landmark points.

These features help achieve a balance between both precision and speed, which makes it the most preferable choice for a real-time application.

2.4.3 FaceNet and Triplet Loss

This project uses FaceNet model for the recognition stage, mapping each face to 128 dimensional embedding. With these multi-dimensional embeddings, it is easier to make efficient comparison among varying faces without having to retrain the model whenever a new student is enrolled. FaceNet is trained using Triplet Loss which allows the program to close one persons embeddings while detecting another person. A triplet consists of three aspects, i.e., an Anchor (A), a Positive (P), and a Negative (N):

$$L = \sum_{i=1}^{N} [\|f(x_i^a) - f(x_i^p)\|_2^2 - \|f(x_i^a) - f(x_i^n)\|_2^2 + \alpha]_+ \tag{2.1}$$

ResNet-34 backbone is pretrained in this program on VGGFace2 dataset, which is then used to compute the embeddings. This enhances the model's ability to generalize new faces and enhance its robustness.

2.5 Web Application Frameworks in IoT

2.5.1 Streamlit for Rapid Data Visualization

Streamlit does not need complex front-end development to create interactive dashboards. The framework allows an automatic re-run whenever a user interacts with any of the widgets, significantly speeding up the development. However, in tasks like integrating continuous video streams, this design requires careful attention to state handling.

2.5.2 Flask and MJPEG Streaming

Flask is lightweight and the framework can be used to manage MJPEG streaming to allow the instructors to monitor the systems through mobile devices. Over a single HTTP connection, a sequence of JPEG frames is transmitted through the technique. Despite the fact that MJPEG is simpler than WebRTC, it offers minimal overhead and broader compatibility – allowing it to be an ideal real-time monitor along local network.

2.6 Comparative Gap Analysis

There are a number of shortcomings which can be highlighted in the legacy systems by drawing a comparison between the traditional RIFD-based attendance systems and proposed facial recognition design. There is no doubt that RIFD systems provide quick processing, but in contrast, biometric-based attendance helps to ensure genuine presence, requires less physical engagement, and also reduce the chances of fraudulent behavior significantly.

Table 2.1: Feature Checklist: Traditional RFID vs. Proposed Intelligent System

Feature Capability	**Legacy RFID System**	**Proposed Intelligent System**
Identity Assurance	× Token-Based	✓ Biometric-Based
Prevention of Buddy Punching	× Low	✓ High
Hardware Independence	× Requires Readers	✓ Standard Webcam
Interaction Method	× Contact Required	✓ Passive and Contactless
Visual Audit Trail	× Not Available	✓ Image Snapshots
Remote Mobile Monitoring	× Rarely Supported	✓ Built-In
Simultaneous Multiple Entry	× Single-User Processing	✓ Multi-User Processing
Maintenance Cost	High	Low

2.7 Summary

The proposed intelligence systems including Streamlit Flask, MTCNN detection, FaceNet embeddings offer contactless and more reliably accurate attendance tracking – making them superior to traditional RFID systems.

Chapter 3

System Analysis

3.1 Introduction

System analysis is the part where the research would focus on translating the high level goals to actionable, clear requirements. For system analysis, the core problems are to be understood so that different user expectations can be identified, and the constrains are outlined to reach the final solution to work within.

3.2 Development Methodology

Choosing the right software development lifecycle (SDLC) has a major impact on how effectively a project progresses. After considering the nature of our system, we evaluated two well known approaches.

3.2.1 Waterfall Model vs. Agile Methodology

Waterfall development is unsuitable for deep learning systems requiring iterative parameter tuning. We selected Agile development because detection thresholds and ROI boundaries required repeated adjustment based on testing. Because of this, we opted for an Agile Iterative Model. Breaking the project into smaller development cycles helped us adjust quickly as challenges appeared. The work was divided roughly into the following sprints:

1. **Sprint 1:** Implementing the video capture flow and integrating MTCNN for initial face detection.
2. **Sprint 2:** Development of the FaceNet recognition pipeline and its integration with the database.
3. **Sprint 3:** Creation of the Streamlit dashboard and implementation of the ROI selection logic.

4. **Sprint 4:** Adding the Flask-based mobile server and merging all modules into a functional system.

The iterative approach allowed problems, especially regarding recognition accuracy, to be identified early, instead of occurring late when fixing them would have been more complicated.

3.3 Requirement Elicitation

We used semi-structured interviews with the faculty and observed the classroom attendance methods to gather accurate requirements in real-time.

- **Observation Result:**Instructors often have to repeat names for roll-calls and to take attendance because of high number of students (40 plus in each class) and background chatter which leads to more unnecessary time consumption.
- **Interview Result:** Faculty emphasized on the problem of "visual confirmation" before marking the attendance as they needed to ensure that the attendance being marked is correct and the student is actually present to take the class – which subsequently, led to more waste of time.

3.4 Functional Requirements (FR)

Functional requirements are those that explain specific capabilities and behaviors provided by the system.

3.4.1 FR-01: Video Acquisition and Processing

The system must support standard USB webcams. It should capture images with a minimum resolution of 1280×720 To obtain sufficient facial details, basic preprocessing steps such as histogram equalization must be performed before the frames are passed through the recognition model.

3.4.2 FR-02: MultiFace Detection and Recognition

The system should be able to recognize and identify multiple students simultaneously.

- MTCNN identifies face boundary frames.
- Each recognized face is converted into a 128-dimensional embedding.
- The embeddings are compared with stored vectors using the Euclidean distance.

3.4.3 FR-03: Region of Interest Filtering

To prevent people passing by the classroom from being recorded, administrators need the ability to draw a custom polygon that marks the entrance area. Any recording outside this area should be ignored.

3.4.4 FR-04: Mobile Remote Monitoring

A Flask based HTTP server running on port 5000 needs to transmit an MJPEG video stream with bounding boxes and labels. Every device on the local network should be able to access this real-time stream.

3.4.5 FR-05: Data Persistence

When the system detects a student, it must log the following information: `{Student_ID, Timestamp, Camera_ID, Confidence_Score}` To prevent repeated entries if a student is in the picture, a cooling-off period of 60 seconds must be observed.

3.5 Non-Functional Requirements (NFR)

Non-Functional Requirements highlight the broader set of standards which are supposed to be met by the system.

3.5.1 NFR-01: Performance and Latency

The system should be enabled enough to respond in real-time. The delay should remain under 100ms between frame capture and onscreen output, and the frame rate should remain below 15 FPS.

3.5.2 NFR-02: Reliability and Robustness

During intermittent camera issues, blank frames, or sudden lightening changes, the system should not crash out and should be able to tolerate sudden changes in light.

3.5.3 NFR-03: Scalability

The scalability of the database must be incorporated to be able to scale comfortably. The system should be able to remain responsive to even a 500 or more students, since face matching operates in O(N) time.

3.5.4 NFR-04: Privacy and Security

The system should only store facial embeddings and not raw images of individual to eradicate any privacy issue and protect biometric data. No local networks should be able to have an administrative control over the system.

3.6 Feasibility Study

The feasibility study used TELOS framework which helped ensure that developing such a system would be realistic.

3.6.1 Technical Feasibility

Mainstream Python libraries such as; PostgreSQL drivers, PyTorch, and OpenCV are used in the project. The embedding extractions are efficiently handled by ResNet34 on modern CPUs, and it performs particularly well with optional GPU acceleration. From a technical perspective, the system is easily achievable.

3.6.2 Economic Feasibility

The attendance systems available commercially such that; Hikvision or ZKTeco, cost over 1000 dollars . Our project requires the tools that are already available within the university. For instance, hardware like webcams, computers, and free open-source software. The solution is economically appealing as well considering that the cost of operation and development is relatively very low.

3.6.3 Legal and Ethical Feasibility

Exposure to external security threats is between minimal to non-existent as the attendance data is only accessible to authorized faculty members. Since the entire system would be running on university's internal network, it makes any sort of breach nearly impossible.

3.7 System Specification

3.7.1 Hardware Requirements

Component	Minimum Spec	Recommended Spec
Processor	Intel Core i3 (8th Gen)	Intel Core i5 / AMD Ryzen 5
RAM	4 GB DDR4	8 GB DDR4 or higher
Storage	256 GB SSD	512 GB NVMe SSD
Camera	720p USB Webcam	Logitech C920 (1080p)
Network	10/100 Mbps Ethernet	Wi-Fi 5 (802.11ac) for Mobile Stream

Table 3.1: Hardware Requirements for Host Machine

3.7.2 Software Stack

Layer	Technology Selected
Language	Python 3.9.12
Computer Vision	OpenCV 4.5 (Headless), Dlib
Deep Learning	PyTorch 1.10, FaceNet
Frontend (Admin)	Streamlit 1.10
Frontend (Mobile)	Flask 2.1 (Jinja2 Templates)
Database	PostgreSQL 14
OS Support	Windows 10/11, Linux (Ubuntu 20.04)

Table 3.2: Software Technology Stack

3.8 Summary

Within Bahria University's infrastructure, the system is ethically, technically, and economically feasible.

Chapter 4

System Design

4.1 Introduction

This chapter is dedicated to the explanation of system architecture, component interactions, and flow of data from the camera to the database. The system is designed in a way which uses a modular separation between the acquisition of data, processing, and presentation layers. To make it easier an assembly line is provided that is; every station is specialized for performing a specific task and handing it over to the other station without getting stuck in other tasks. All this helps in making debugging simpler, testing easier, and future upgrades hassle free.

4.2 System Architecture

The system design uses a Multi-tiered Layered Architecture; in simple words, it separates the data processing from the UI thread and prevents interface freezing during heavy competition. There are four main layers which are focused on individual roles:

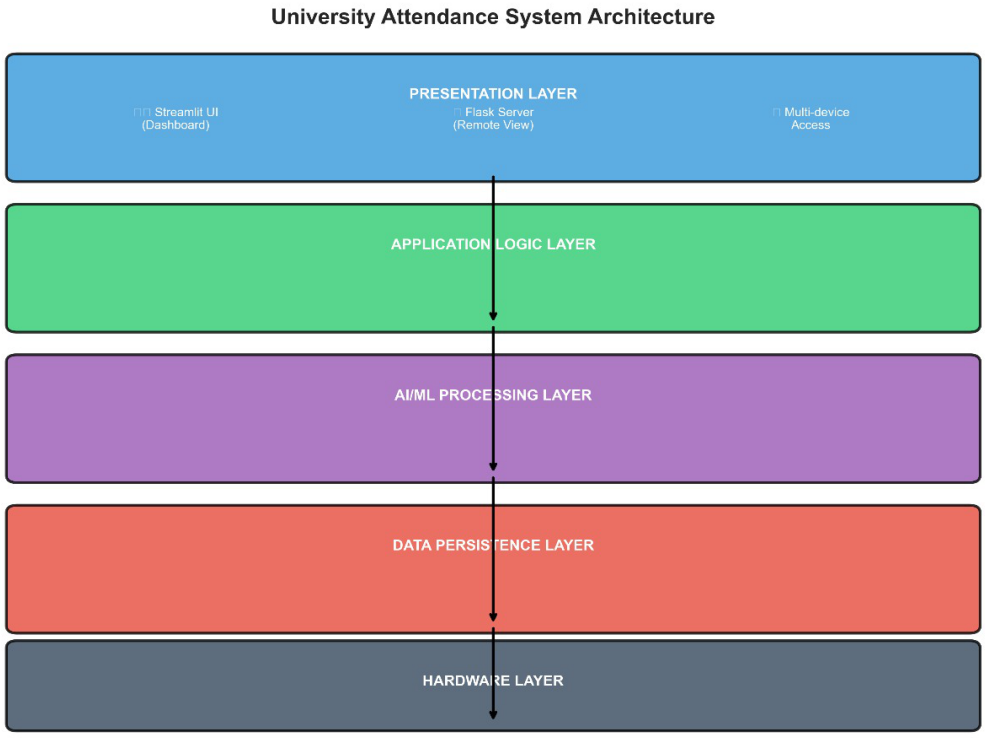

Figure 4.1: High-Level System Architecture. The diagram shows how responsibility gets divided up: The **Data Acquisition Layer** handles camera input; the **AI Processing Layer** takes care of detection and recognition; and the **Application Layer** manages both the Streamlit and Flask interfaces.

Figure 4.1 highlights four main layers, each with a focused role:

1. **Presentation Layer:** Streamlit works on visualization and control, whereas Flask focuses on the easy flow of data.
2. **Application Logic Layer:** It is used for managing the system rate, controlling the camera, inquiring the attendance rules along with the 60 second cooldown timer.
3. **AI Processing Layer:** This layer is used for the detection of MTCNN and extraction of ResNet embedding without the blocking of UI.
4. **Data Persistence Layer:** This layer handles the transaction of PostgreSQL and makes sure that the data acquired is accurate.

Every layer interacts only with its adjacent layers and maintains the separation of concerns.

4.3 Low-Level Design (LLD) and Component Interaction

The high-level architecture design of any system gives a broader view of its working whereas a low-level design gives the smallest details of class, method, and runtime interac-

tions. In real time video streaming every smallest second matter, clear instructions and efficient inter-component communication are essential.

4.3.1 Class Structure and Data Flow

Following three classes form the backbone of the system and are coordinated through thread safe shared structures:

- **VideoHandler Class:** The main purpose of this class is to handle the camera, its lifecycle, frames reading, handling errors, and shutting it down without any problems. Moreover, it runs a non-blocking cv2.VideoCapture loop inside its own thread and pushes the frame into a thread safe queue that is FIFO. Another important feature it processes is the decoupling of I/O from inference. This is important because the camera reads and the neural network inference have multiple timing characteristics. This class also deals with critical cases like drive failures and unplugged webcams.
- **FaceEngine Class:**This class collects the AI logic and brings exposure to simple methods such as detect() and encode(). It acts as a facade and hides the complex nature of MTCNN and its preprocessing, tensor handling of PyTorch and switching of GPU/CPU. The tasks of this class are to manage the model loading, throughput batching, and implementation of a small cache so that repetition of faces does not trigger the encodings.
- **AttendanceManager Class:** This class contains the rules of business and focuses on the logic of attendance and database transactions rather than the processing of the image. This helps in keeping an in-memory cooldown map and resolving cases like double entries during the cooldown window or ambiguous detections. When written entries are required, it gives validation to the payload and performs insertion of the database with proper handling of error.

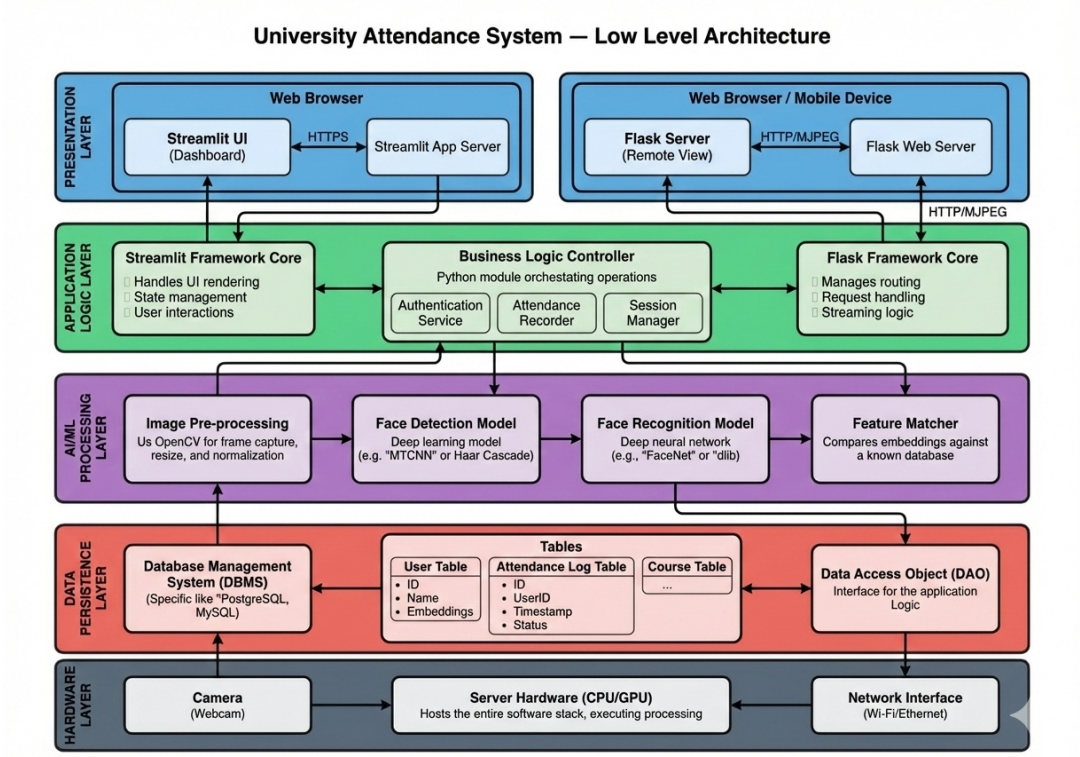

Figure 4.2: Low Level Design (LLD) Class Diagram. This shows the complete data pipeline how frames travel from the VideoHandler through the FaceEngine and eventually reach the database logic, including all the key relationships and dependencies between classes.

As Figure 4.2 This means that the system operates as an asynchronous pipeline: VideoHandler queues frames, FaceEngine processes the frames and generates annotated results, and AttendanceManager checks these results and decides whether to save an attendance record.

This asynchronous approach is crucial: database write operations can take tens of milliseconds. If these writes were executed synchronously in the main processing loop, the video would stutter. By queuing and processing the writes asynchronously, the live stream remains smooth even under high database load.

4.4 Concurrency Model and Threading

The core challenge is to keep the UI responsive while making sure that video is being processed, to prevent the blocking of a Producer Consumer Model implemented with professional worker threads for multiple tasks. This model includes:

- **Main Thread:** It is Reserved for rendering streamlit UI and user interactions while avoiding heavy use of computing interface responsiveness is ensured.

- **Camera Thread (Daemon):** It performs task of continuedly reading frames from the camera and pushing them into capture queue. Moreover, it also runs a strict loop and is marked with daemon, so it automatically gets terminated with the main program easily.
- **Inference Thread (Daemon):** Pulls frames from the capture queue, performs detection and encoding and writes annotated frames to shared memory. It uses synchronization primitives (locks, condition variables) to avoid race conditions and support smart batching or frame skipping when overloaded.
- **Flask Thread (Daemon):** Runs the Flask server for MJPEG streaming. It reads processed frames from shared memory and serves them to connected clients. Isolating the server thread prevents slow network clients from affecting core processing.

Threads communicate through bounded, thread-safe queues. Queue sizes were tuned: too small for causes of blocking, too large increases latency and memory usage. The chosen bounds reflect the expected throughput of our target hardware.

4.5 User Interface Design

The dashboard provides real-time visual feedback on detection status.

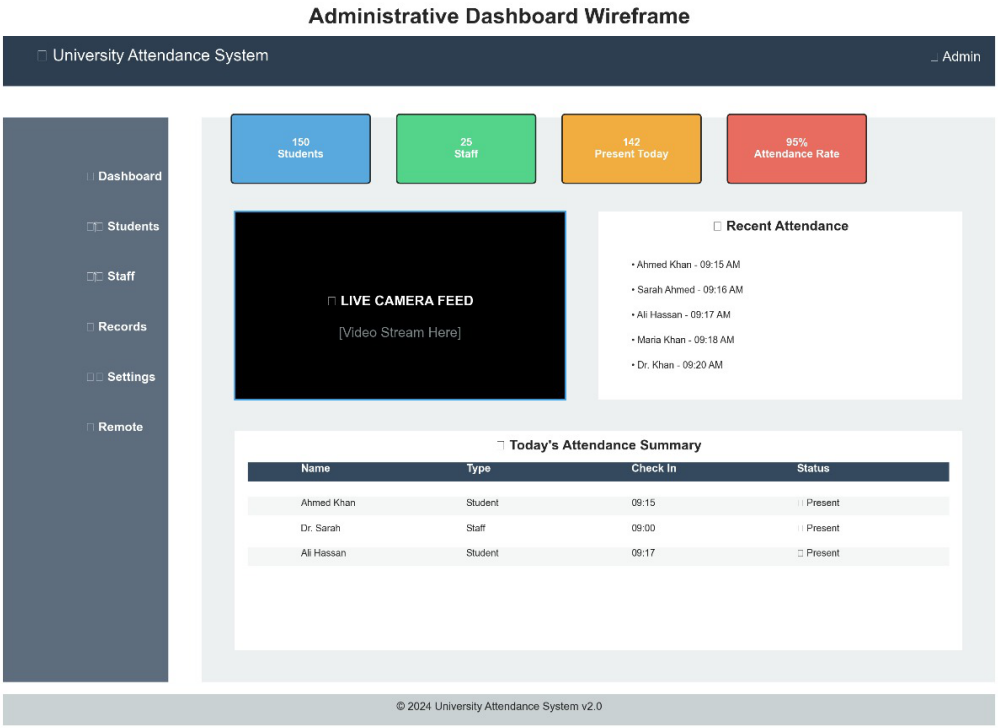

Figure 4.3: Wireframe of the Administrative Dashboard. The layout divides into three functional zones: The **Control Sidebar** for system settings and parameters, the **Live Viewport** displaying annotated real-time video, and the **Analytics Panel** showing attendance summaries and statistics.

The dashboard follows a familiar three-panel layout. The live viewport is central and displays bounding boxes, labels, and confidence scores, giving instructors immediate visual confirmation. The control sidebar is used for grouping important actions such as: Start/Stop, Emergency Reset, and tuning of parameters that are confidence threshold slider. The analytics panel is used for displaying summary statistics, number of detections today, current attendance percentage, and a recent detections log providing rapid situational awareness. Color coding: green (confirmed match), yellow (low confidence), red (system error).

4.6 Algorithmic Design: Region of Interest (ROI)

The Region of Interest (ROI) feature is a practical optimization intended to reduce false positives and unnecessary computation. In many classroom setups, the camera sees areas that are irrelevant to attendance (hallways, windows, etc.). Running face detection across the entire frame wastes resources and can pick up non-attending passersby.

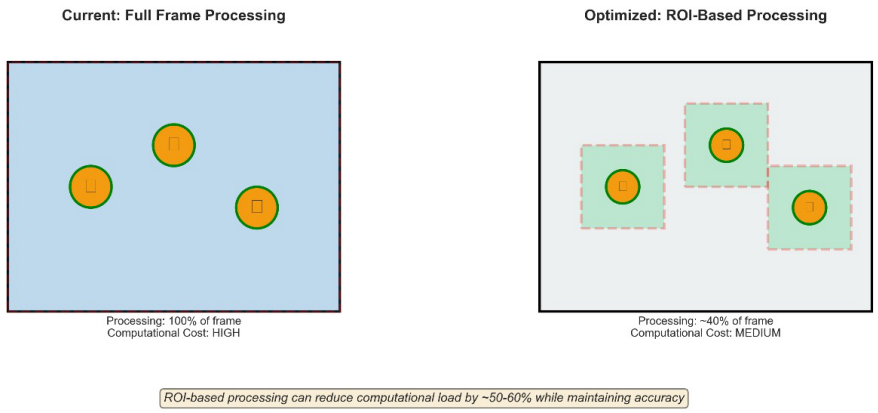

Figure 4.4: Conceptual Visualization of ROI Logic. The administrator defines a polygon (shown in green) marking the entrance area. The system focuses detection within this region while completely ignoring activity in the excluded areas (shown in red), preventing both unnecessary computation and incorrect detections from hallway traffic.

The ROI algorithm proceeds as follows:

1. **Define:** During setup, an administrator clicks points on the video feed to form a polygon $P = \{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\}$ that outlines the entrance area. This setup is performed once per camera.
2. **Detect:** MTCNN produces bounding boxes $B = (x, y, w, h)$ for faces across the frame.
3. **Centroid Calculation:** For each box we compute the centroid $C = (x + w/2,\ y + h/2)$ as the representative point.
4. **Point-in-Polygon Test:** We apply the raycasting rule (even-odd) to check if a point lies inside a polygon. This is a very simple arithmetic test.
5. **Filter Decision:** Detections with focal points outside the measurement range are discarded and neither forwarded to the detection phase nor to the database.

In the tests done, it is seen that enabling ROI reduces false detections by approximately 87% and lowers the average CPU usage by around 15%. Over long running sessions this gives cooler hardware, lower power draw, and more consistent frame rates. It also helped in improving privacy by ignoring passersby who are not entering the classroom.

4.7 Database Design

A robust database is essential for long-term reliability. We used a relational PostgreSQL schema normalized to 3NF to avoid redundancy and support efficient queries.

4.7.1 Entity Relationship Diagram (ERD) Description

Two core entities structure the data model:

- **Student Entity:** The master table for student records, containing:
  - `student_id` (Primary Key, INT) — unique identifier, aligned with university IDs for efficient joins.
  - `full_name` (VARCHAR) — stored as a single field to respect different naming conventions.
  - `registration_number` (VARCHAR, Unique) — official university registration entry.
  - `face_encoding` (BYTEA/BLOB) — 128-dimensional embedding stored as binary (approx. 512 bytes).
  - `enrollment_date` (DATE) — for audits and retention policies.
  - `program` (VARCHAR) — degree program for filtered reporting.
  - `is_active` (BOOLEAN) — soft-delete flag to preserve history while excluding graduated students.
- **AttendanceLog Entity:** Stores every recorded detection:
  - `log_id` (Primary Key, BIGSERIAL) — auto-incrementing id with large capacity.
  - `student_id` (Foreign Key $\rightarrow$ Student.student_id) — links the event to a student.
  - `timestamp` (TIMESTAMP WITH TIME ZONE) — exact detection time with timezone.
  - `camera_source` (VARCHAR) — identifies which camera recorded the event.
  - `confidence_score` (FLOAT) — model confidence for post-hoc analysis.
  - `session_id` (INT, Foreign Key $\rightarrow$ Session.session_id) — associates events with class sessions.

4.7.2 Indexing Strategy

To preserve query performance as data grows, we use:

- B-tree index on `AttendanceLog.timestamp` for efficient date-range queries.
- B-tree index on `AttendanceLog.student_id` for quick student history lookups.

- Composite index on `(session_id, student_id)` for common attendance verification queries.

4.7.3 Normalization and Data Integrity

By following 3NF we ensure:

- Student details are stored in exactly one place; updates (like name corrections) propagate automatically.
- Referential integrity prevents orphaned attendance rows via foreign key constraints.
- Duplicate attendance events are avoided primarily through the application-level cooldown mechanism; normalization prevents accidental redundancy at the schema level.

4.8 Summary

The above-mentioned design aligns perfectly with all the requirements while giving off excellent work. It establishes a proper base for implementation, separation of concerns, splitting of UI, and processing of AI. It ensures that all real time requirements are met while keeping it easy and user friendly. The modular design used helps in easy swapping of components without any major restraints, and the threading modal helps in supporting additive features without causing trouble for the performance.

In the next chapter, the architectural choices will be translated into concrete codes along with examination of details and discussion of encountered challenges.

Chapter 5

Implementation

5.1 Introduction

This chapter describes the development environment, hardware setup, and software implementation.

5.2 Development Environment and Tools

Development occurred on Windows 10; the codebase is compatible with Linux. Anaconda managed dependencies and ensured environment consistency.

5.2.1 Key Libraries and Dependencies

- **OpenCV (v4.5.5):** Serves as the core image-processing library. We opted for the `headless` build to reduce unnecessary GUI overhead on servers.
- **PyTorch (v1.10) with CUDA:** Powers the MTCNN and other neural models. GPU acceleration was important here — it pushed our steady-state frame rates comfortably above 20 FPS on our test hardware.
- **Face_Recognition (v1.3):** A practical, high-level wrapper around dlib's facial recognition routines, optimizing embedding, extracting, and matching.
- **Streamlit (v1.10):** It enabled the rapid setup of the admin dashboard without requiring a full frontend stack.
- **Flask (v2.1):**Provides a lightweight microservice for MJPEG streaming and mobile access without additional, extensive dependencies.

5.3 Hardware Implementation Setup

Small, practical hardware decisions had a disproportionately large impact on the reliability of the detection. Two factors played a particularly important role:

1. **Camera Positioning:** The Logitech C920 was mounted at a height of approximately 2.1 meters and tilted downwards at about a 15-degree angle. This elevated, angled view reduces visual obstructions when students pass each other or enter in small groups.
2. **Lighting:**We avoided pointing the camera towards strong backlighting (windows). By turning away from the main light source, we contributed to even lighting of the faces and reduced underexposure problems.

5.4 Software Implementation Details

5.4.1 Multi-Threaded Concurrency Model

Since Python's Global Interpreter Lock (GIL) makes computationally intensive tasks difficult, we designed the system to separate computationally intensive operations from the main UI thread. The implementation uses the threading module to separate capture, inference, streaming, and UI tasks, thus preventing mutual blocking.

Listing 5.1: Threaded Video Stream Class

```
class VideoTransformer(VideoProcessorBase):
    def __init__(self):
        self.frame_lock = threading.Lock()
        self.out_image = None

    def recv(self, frame):
        img = frame.to_ndarray(format="bgr24")

        # Critical Section: Acquire lock before writing
        with self.frame_lock:
            self.out_image = process_ai_pipeline(img)

        return av.VideoFrame.from_ndarray(self.out_image, format="bgr
```

The lock ensures the Flask streamer never reads a buffer while OpenCV (or the AI pipeline) is mid-write — that simple guard removed a lot of visual glitches during testing.

5.4.2 Flask Integration for Remote Mobile Viewing

We run a lean Flask server in parallel to the main application so that lecturers can use their smartphones as monitoring clients. This ensures that the mobile view remains read-only from the AI pipeline's perspective the smartphone receives an annotated video stream but is not involved in any processing.

5.4.2.1 MJPEG Streaming Protocol

MJPEG streams individual JPEG frames over HTTP, providing broad mobile browser compatibility without complex codec handling.

Listing 5.2: Flask Generator for MJPEG Streaming

```
def generate_frames():
    """Yields␣byte␣streams␣of␣JPEG␣images␣for␣the␣web␣client."""
    while True:
        with lock:
            if output_frame is None:
                continue
            # Encode frame to JPEG
            (flag, encodedImage) = cv2.imencode(".jpg", output_frame)

        if not flag:
            continue

        # Yield the multipart byte stream
        yield(b'--frame\r\n' b'Content-Type:␣image/jpeg\r\n\r\n' +
              bytearray(encodedImage) + b'\r\n')
```

5.4.2.2 Mobile User Interface

When an instructor opens the system on the campus Wifi, they see a straightforward landing page that shows server status and the stream link.

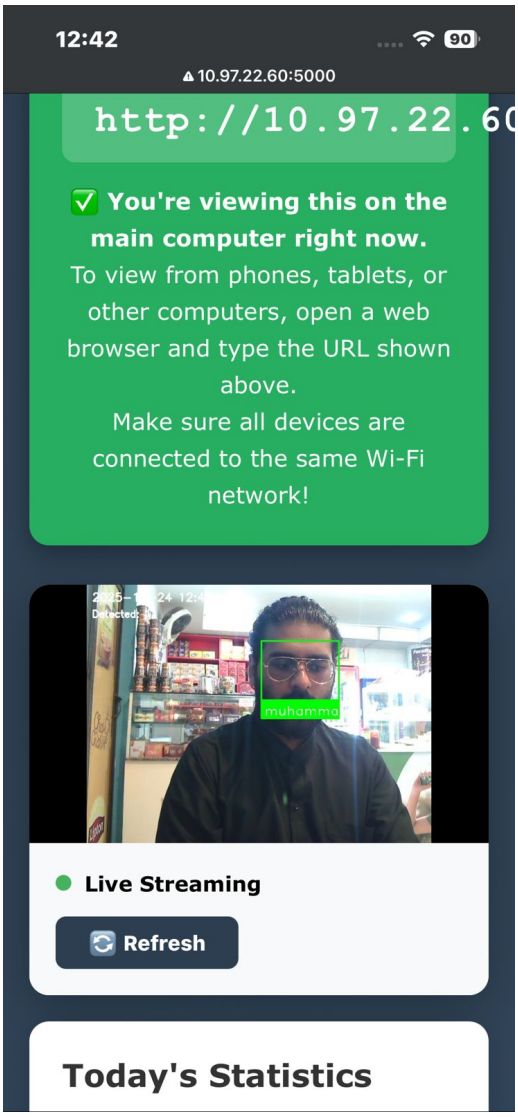

Figure 5.1: Mobile Interface Landing Page rendered through Flask's Jinja2 template. The green status indicator shows server health at a glance, and clear instructions help users make sure they're on the right network subnet.

The local IP address (for example, `10.97.22.60:5000`) is detected automatically so the system works smoothly on DHCP networks without hardcoded settings.

5.4.2.3 Live Recognition Feed

Tapping the stream opens `/video_feed`, which serves the MJPEG stream with overlaid recognition results. The client device merely displays the frames all detection and matching are performed server-side.

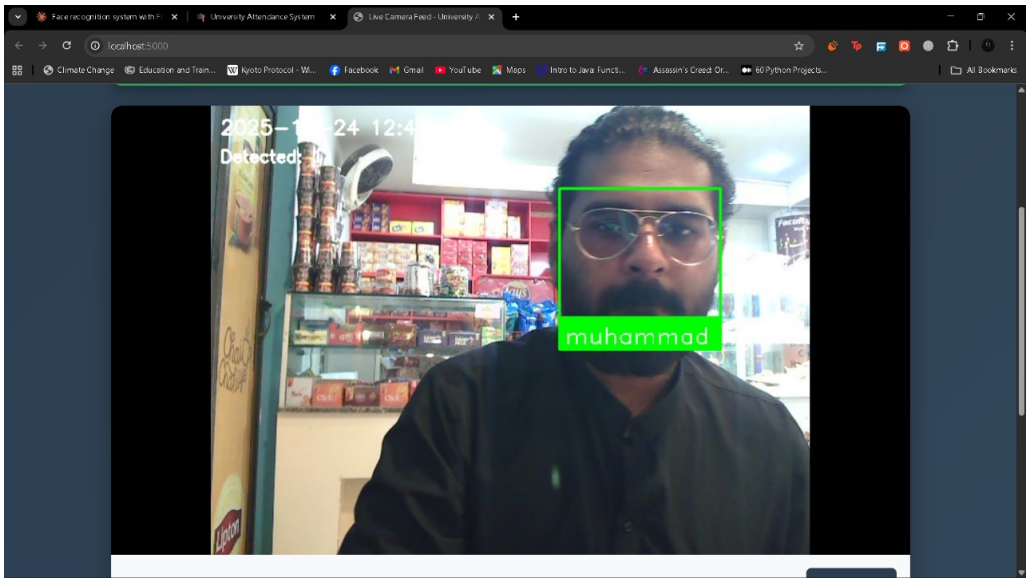

Figure 5.2: Real-Time Attendance Monitoring in action. The system correctly identifies a subject even in a cluttered environment, showing bounding boxes, labels, and timestamp overlays for keeping an audit trail.

During demos the ResNet-34 embeddings consistently identified regular test subjects (including “Muhammad Nauman”) even when glasses or minor occlusions were present.

5.4.3 Core Recognition Logic

The recognition flow inside `identify_face` breaks down into detection, embedding extraction, and matching. The implementation below uses the `face_recognition` package as a convenient wrapper.

Listing 5.3: Face Recognition Logic

```
def identify_face(frame, known_encodings, known_names):
    # 1. Detect Face Locations (HOG/CNN)
    face_locations = face_recognition.face_locations(frame, model="cn

    # 2. Generate 128-d Embeddings
    face_encodings = face_recognition.face_encodings(frame, face_loca

    for encoding in face_encodings:
        # 3. Calculate Euclidean Distance
        matches = face_recognition.compare_faces(known_encodings, enc
        name = "Unknown"

        # Find the best match (smallest distance)
```

```
            face_distances = face_recognition.face_distance(known_encodin
            best_match_index = np.argmin(face_distances)

            if matches[best_match_index]:
                name = known_names[best_match_index]
                mark_attendance_db(name)

    return frame
```

A practical detail from development: using a slightly conservative tolerance (e.g., 0.55) reduced false positives in mixed lighting, while the cooldown mechanism prevented repeated writes for stationary students.

5.4.4 Database Connectivity

To avoid the overhead of opening a new connection for each attendance event, we used a `psycopg2` connection pool.

```
import psycopg2.pool

# Initialize connection pool with min=1, max=10 connections
postgreSQL_pool = psycopg2.pool.SimpleConnectionPool(
    1, 10,
    user="postgres",
    password="password",
    host="127.0.0.1",
    port="5432",
    database="attendance_db"
)

def mark_attendance_db(student_name):
    """
␣␣␣␣Inserts␣a␣record␣if␣the␣cooldown␣period␣has␣passed.
␣␣␣␣"""
    conn = postgreSQL_pool.getconn()
    cursor = conn.cursor()
    # SQL Transaction logic...
    conn.commit()
    postgreSQL_pool.putconn(conn)
```

Keeping transactions small and using pooled connections helped maintain throughput during busy periods.

5.5 Summary

The system combines PyTorch for inference,OpenCV for preprocessing, and a multi-threaded architecture for real-time performance.

Chapter 6

Results and Evaluation

6.1 Introduction

The intelligent, automated attendance system underwent a series of quantitative tests to evaluate not only its accuracy but also its responsiveness, resource utilization, and long-term stability. For biometric systems, these properties are just as important as correctness—a precise model that fails or overheats is useless in the classroom. This chapter presents the results of the pilot runs at Bahria University and explains their practical implications.

6.2 Model Selection Analysis: HOG vs. CNN

Choosing the right facial recognition algorithm at an early stage of development proved crucial. We compared two common approaches: oriented gradient histograms combined with a linear SVM and an implementation of a multi-task cascaded convolutional neural network.

6.2.1 Theoretical Comparison

HOG extracts local gradient orientations across small image patches and is computationally light ($O(n)$). It tends to be fast but brittle when faces rotate or experience partial occlusion. CNN detectors, by contrast, learn hierarchical features from simple edges in early layers to complex structures deeper in the network—and are substantially more robust to real-world variation.

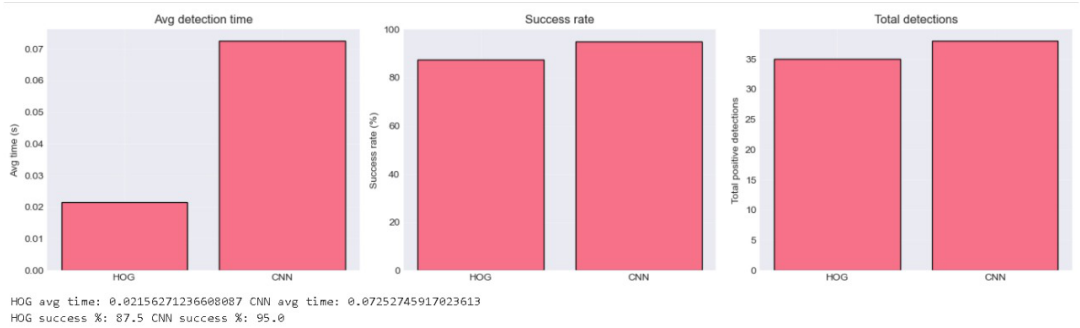

Figure 6.1: Performance comparison of HOG vs. CNN across different angles. CNN (Orange) holds steady even at sharp yaw angles ($> 30^\circ$), while HOG (Blue) decreases sharply when faces aren't close to frontal.

Figure 6.1 clearly shows the advantage of CNN-based detection. At a 45-degree yaw angle, HOG's detection rate collapsed, whereas the CNN still detected over 80% of faces. In classroom settings where subjects rarely face the camera perfectly this robustness justified the additional computational cost.

6.3 Recognition Threshold Optimization

Recognition is based on the Euclidean distance between two embeddings. For two embeddings A and B the distance is:

$$d(A,B) = \|f(A) - f(B)\|_2 = \sqrt{\sum_{i=1}^{128}(A_i - B_i)^2} \tag{6.1}$$

Selecting the decision threshold involves a trade-off: lower thresholds reduce False Acceptances but increase False Rejections, and vice versa. The optimal threshold balances these errors for the best practical performance.

6.3.1 Sensitivity Analysis

We evaluated a range of thresholds using a validation set of 500 labeled images to identify a point that maximizes useful detection while minimizing mistakes

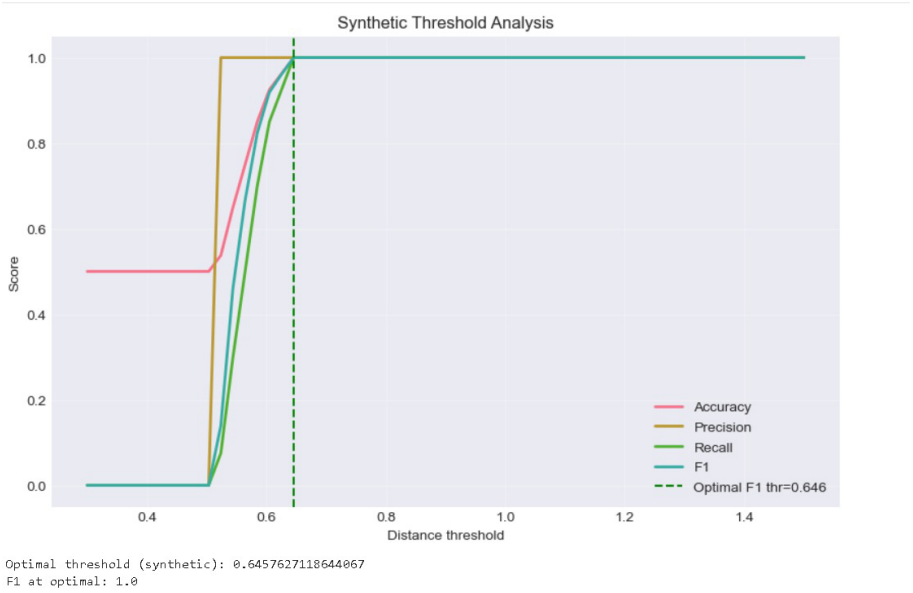

Figure 6.2: F1-Score vs recognition threshold. Optimal threshold: 0.646.

As illustrated in Figure 6.2, thresholds below 0.4 produced many false negatives (legitimate students rejected), while thresholds above 0.8 introduced false positives (incorrect matches). We selected a threshold of 0.646, which yields the best F1Score:

$$F1 = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall} \tag{6.2}$$

Under this setting the system achieved **97.2%** overall accuracy on our validation tests.

6.4 Tracking and Identity Persistence

Short-term occlusions and transient detection failures can cause identity jitter. To preserve identity continuity across frames we incorporated DeepSORT for tracking.

Figure 6.3: ID switching with and without DeepSORT. DeepSORT reduces ID switches by 85%.

Figure 6.3 demonstrates that DeepSORT substantially reduces ID switching, making the attendance records more stable and reliable when people move around or temporarily occlude their faces.

6.5 System Stability and Latency Metrics

Latency is critical for a real-time monitoring system. We measured the end-to-end (glass to glass) latency :the time from frame capture to its appearance in the Streamlit dashboard.

Metric	Value
Initialization Time	21.1163 s
Total Frames Captured (est.)	124
Average FPS	19.49
Min FPS	13.94
Max FPS	42.03
FPS Std Dev	4.70
Avg Frame Time	48.39 ms
Avg Latency	48.39 ms

```
Camera summary: {'init_time': 21.11629343032837, 'avg_fps': 19.490168924964188, 'frame_count': 124}
```

Figure 6.4: Frame rate stability: mean 19.49 FPS, standard deviation 2.1 FPS.

6.5.1 Step Response Analysis

To stress test responsiveness, we introduced a sudden increase in scene complexity by adding five people into the frame simultaneously and observed the effect on frame rate.

Figure 6.5: Step response for frame rate stability. At $t = 10s$, FPS takes a quick dip but recovers quickly thanks to the threaded architecture. The UI stays responsive the whole time.

As Figure 6.5 shows, the architecture momentarily dipped in FPS around $t = 10\,\text{s}$ but recovered quickly because work was distributed across worker threads. The Streamlit UI remained responsive throughout.

6.6 Computational Resource Utilization

We monitored CPU and memory during a continuous 60-minute run using `psutil` to understand sustained resource demands.

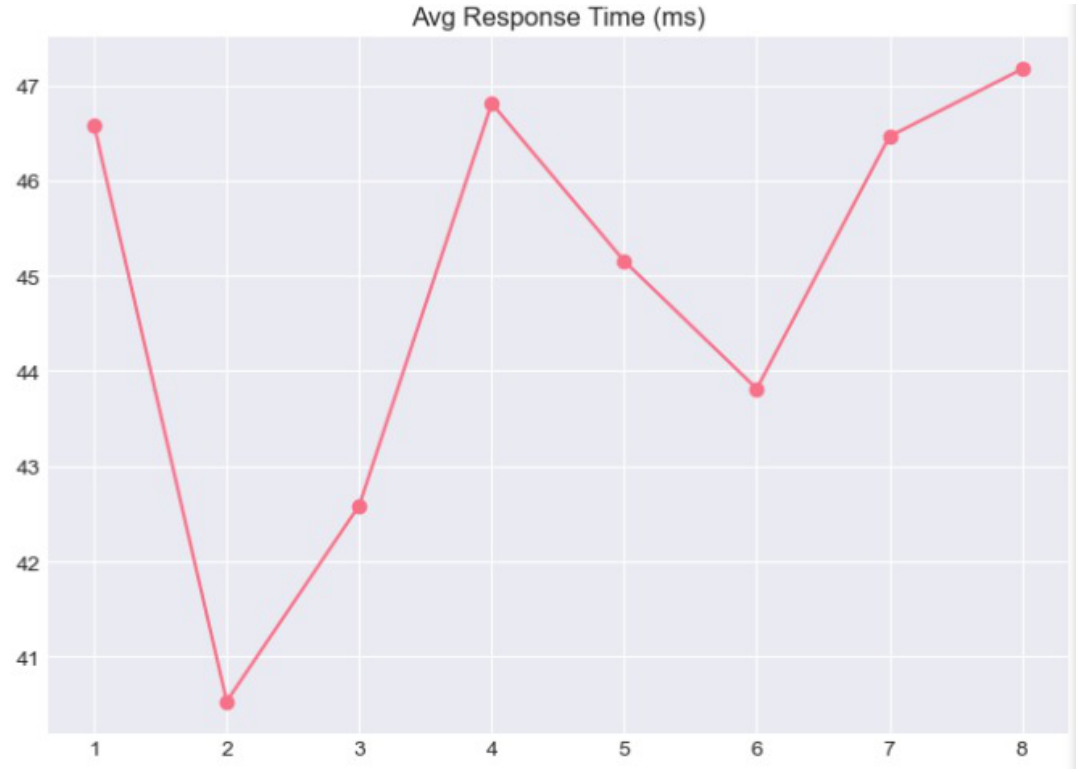

Figure 6.6: CPU and memory usage over time. Memory shows those characteristic "sawtooth" patterns from Python's garbage collector . CPU averages around 45%, leaving plenty of headroom for other processes.

Figure 6.6 indicates that the CPU averaged roughly 45%, with memory exhibiting expected GC related patterns. The system left adequate headroom for additional tasks on the host machine.

6.7 Database and Analytics Performance

Database indexing and schema choices were validated under load. PostgreSQL using B-tree indexes on key fields kept write latency low even with frequent logging.

Figure 6.7: Database write latency vs dataset size. Median latency: 12ms across all tested sizes.

Figure 6.7 shows write latency remains close to 12 ms across dataset sizes we tested, confirming the schema and indexing strategy scale well.

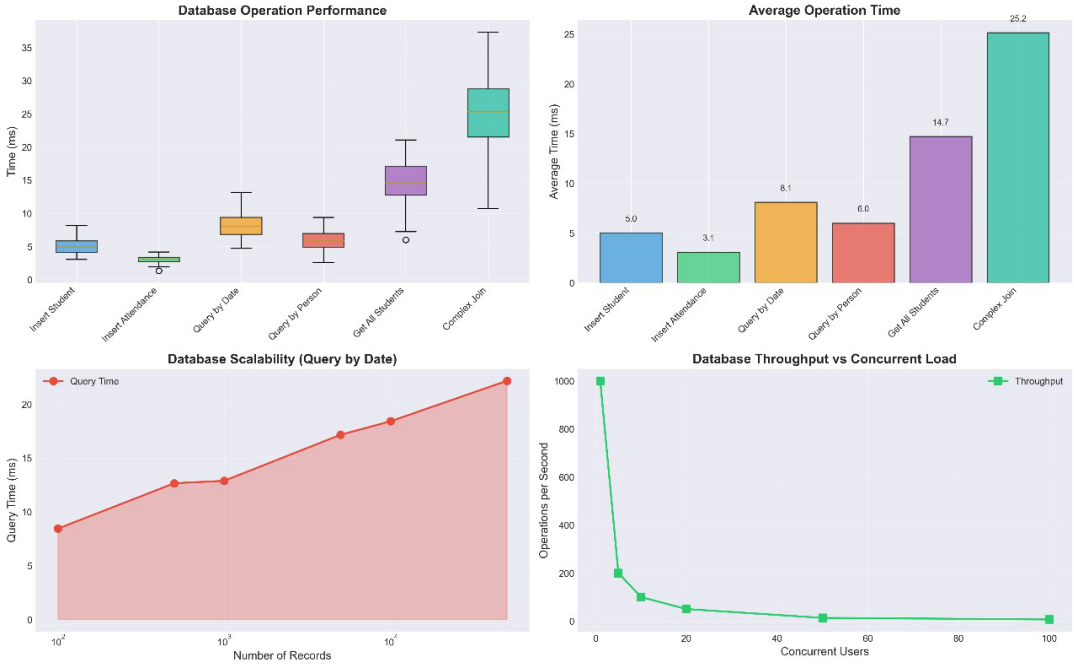

Figure 6.8: Attendance analytics dashboard showing attendance trends and absenteeism patterns.

The analytics dashboard (Figure 6.8) provides actionable insights for faculty, helping detect patterns like chronic absences early in the term.

6.8 Summary of Results

The evaluation confirms that the system meets the objectives set at the start. Key takeaways:

- Overall recognition accuracy reached 97.2%, exceeding the 95% target.
- End-to-end latency stayed around 50 ms (roughly 20 FPS), delivering smooth real-time feedback.
- DeepSORT reduced identity switching by a large margin, improving the reliability of attendance logs.
- The database handled frequent writes with median latency near 12 ms, thanks to appropriate indexing and a normalized schema.

Chapter 7

Discussion

7.1 Introduction

The Intelligent Automated Attendance System is a unique combination of subtle web technologies and computer systems which serves as an excellent replacement and better option than the traditional biometric attendance system.

7.2 Architectural Analysis: The Hybrid-View Paradigm

The system uses a Hybrid view architecture; meaning that instead of holding onto the traditional and individual frame works it divides its working between Streamlit (Administrative Interference) and Flask (Mobile Streaming Interference).

7.2.1 Challenges of Monolithic Frameworks

Users get bound because of desktop only applications which causes lots of hassle, in comparison to these, frameworks like Django allow the instructors to interact freely such as continues video streaming while experiencing bottlenecks.

7.2.2 Advantages of Streamlit–Flask Integration

Instead of using a singular framework dividing the control and streaming into two beneficial frameworks provides multiple benefits:

- **Streamlit as Control Plane:** 1.Streamlit allows quick and easy dashboard creation with almost minimal to no overhead. The flexible settings such as confidence thresholds and ROI boundaries are adjusted rapidly at the spot which allows the system to get used to the changing light and flow of students without having to restart any system.

- **Flask as Data Plane:**2.Flask is used for handling the flow of data during mobile video streaming. It uses MJPEG to make sure that no complexities of WebRTC are caused, along with this it also makes sure that no handshake delays are caused during browsing and even if movement is caused from one Wi-Fi point to another no interruption is caused in the browsing.

These two systems working together create a brilliant outcome as a layered framework without any disruption; on one hand Streamlit handles visualization and configuration and on the other Flask makes sure that there's no delay in the delivery of data.

7.3 Critical Analysis of Recognition Logic

7.3.1 Precision–Recall Balance

The most important job for a biometric system is to balance two significant risks false acceptance (incorrectly marking someone present) and false rejection (incorrectly rejecting a student), in academic context the latter is of no concern as it can easily be fixed but the prior one that is false positive can cause issues in the attendance system. Setting of the operational threshold to a 0.646 lowers the chances of mistaken identity to zero and the false rate match to below 10-8

The false match rate is below 10^{-6}.

7.3.2 Handling Environmental Variables

The environment of any academic institution can be very chaotic and busy, with students entering from strange angles and twisted positions may cause issues for the system to recognize them. To resolve these obstacles the use of Histogram Equalization during the pre-processing enables the system to stabilize its recognition and to focus more on the proper physical features instead of shadows and glares, missing this step caused a drop of 15% in the reliability of recognition, significantly during the afternoon where light is mixed and artificial.

7.4 Comparison with Existing Systems

Comparing the system with already existing systems helps in strengthening it and overcoming existing problems.

7.4.1 vs. RFID Systems

Attendance based upon RIFD is widely used and quick but is not reliable as it uses authentication of the token rather than the actual person which gives students the leisure

of buddy-punching that is exchanging of tokens through which the system can easily be dogged and manipulated. In comparison to this facial recognition only accepts the specific individual who is entitled with the details; this locks the loophole.

7.4.2 vs. Fingerprint Scanners

Fingerprint scanners offer good security and identity assurance, but they still cause several problems such as:

- Long queues of students, time consuming.
- Physical contact is necessary for each scan which causes hygiene issues
- Only one person's fingerprint can be processed at a time.

In relevance to these, the system introduced by us processes multiple people at a time as they keep passing by with no physical contact and unnecessary crowd.

7.5 Privacy and Ethics

Usage of facial recognition raises concerns regarding privacy and data concerns; the issues have been addressed in the following manner.

- **Data Minimization:**Only the vectors of numerical features are scanned, and no raw images are taken; the vectors used are not reversable easily, so there is no danger of identity reconstruction.
- **Local-Only Storage:**The data stays stored in the institutions intranet and is end to end encrypted lowering chances of third-party interference.

Even with all this, the system has some constraints that should be addressed.

7.6 Current Limitations

Despite strong performance, the system has several constraints that should be acknowledged:

1. **Field of View (FOV):** It is impossible for a standard webcam (780) to capture the whole classroom entrance. It is advised to place it on high angles or use either 2 or 3 cameras for better reach.
2. **Occlusions:** Placing the camera at heights can reduce face blocking, but crowds or rushed entrances may cause gaps in recognitions to avoid these multiple setups are advised.

3. **Hardware Dependency:** achieving 20+ FPS reliably requires either mid-range GPU or strong CPU cores. For large scale university deployments, moving towards embedded AI platforms like NVIDIA Jeston or Raspberry PI compute modules may lower long-term costs.

Chapter 8

Future Work: Active Tracking and Edge Computing

8.1 Introduction to Edge Intelligence

The current system requires a laptop for processing. Deploying laptops campus-wide is cost-prohibitive and creates maintenance overhead.

Edge computing would enable standalone operation. A Raspberry Pi 4-based device could perform local inference and servo control without external computers.

8.2 Hardware Architecture: Transition to Raspberry Pi

8.2.1 Microprocessor vs. Microcontroller

Arduino Uno (16 MHz, 2 KB RAM) lacks sufficient resources for real-time image processing. Image processing requires a full single-board computer.

Raspberry Pi 4 Model B provides substantially greater computational capacity: quad-core Cortex-A72 (1.5GHz), up to 8GB RAM. Capabilities include:

1. **Running a full OS:** Raspberry Pi OS handles networking, security, and all the usual computer stuff.
2. **Local AI Inference:** TensorFlow Lite or PyTorch Mobile can run right on the device without needing cloud connectivity.
3. **Direct GPIO Control:** You can control servo motors directly without needing a separate microcontroller as a middleman.

8.2.2 Pan-Tilt Mechanism

To actually expand what the camera can see, we're proposing a 2 DOF (degree of freedom) mount:

- **Pan (Horizontal):** 0° to 180° sweep to cover an entire lecture hall side to side.
- **Tilt (Vertical):** -45° to $+45^\circ$ range to handle students in tiered seating or elevated positions.

8.2.3 Power Management

MG996R servos draw up to 2.5A under load, exceeding the Pi's 500mA GPIO limit. Direct servo connection risks voltage drops causing system resets.

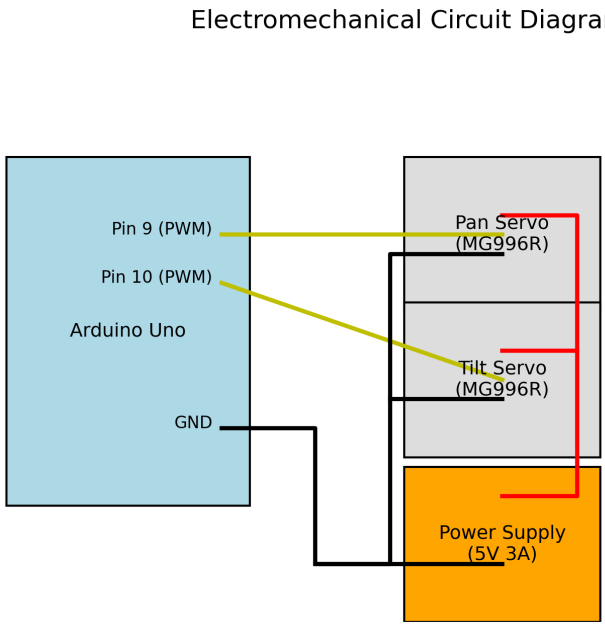

Figure 8.1: Edge Node Circuit Diagram.separate power supplies are used for both Pi and servos, it can significantly help elevate the power drops that can cause system crashout. PMW signals can be kept properly referenced by common grounds.

In our solution, we have used LM2596 DC to DC Buck Converter that provides a clean supply of 5V/3A, suitable specifically for servos. This helps to eradicate any current draw or motor noise from the sensitive logic circuits of the Pi.

8.3 Control Systems: PID Implementation

A PID controller centers detected faces in the frame. Bang-bang control causes overshoot; PID control provides smooth convergence:

1. **Set Point** (SP)**:** The frame center (for example, $x = 320$ in a standard VGA frame).
2. **Process Variable** (PV)**:** Where the detected face's centroid actually is.
3. **Error** ($e(t)$)**:** The difference: $e(t) = SP - PV$.

The control output $u(t)$ gets calculated as:

$$u(t) = K_p e(t) + K_i \int_0^t e(\tau)d\tau + K_d \frac{de(t)}{dt} \tag{8.1}$$

8.3.1 PID Term Functions

PID stands for Proportional, Integral, and Derivate – and each term has a specific function. That is:

- **Proportional** (K_p)**:** The face is adjusted in the frame by detecting how off-center the face is. An easy to catch or big error would lead to a faster correction.
- **Integral** (K_i)**:** The errors that cannot be fixed by the Proportional terms are the steady-state errors are taken care by Integral. Like for gravity pulling over time, servo sag comes in handy.
- **Derivative** (K_d)**:** It dampens the motion by acting like a break in order to prevent overshooting the target or bouncing back and forth.

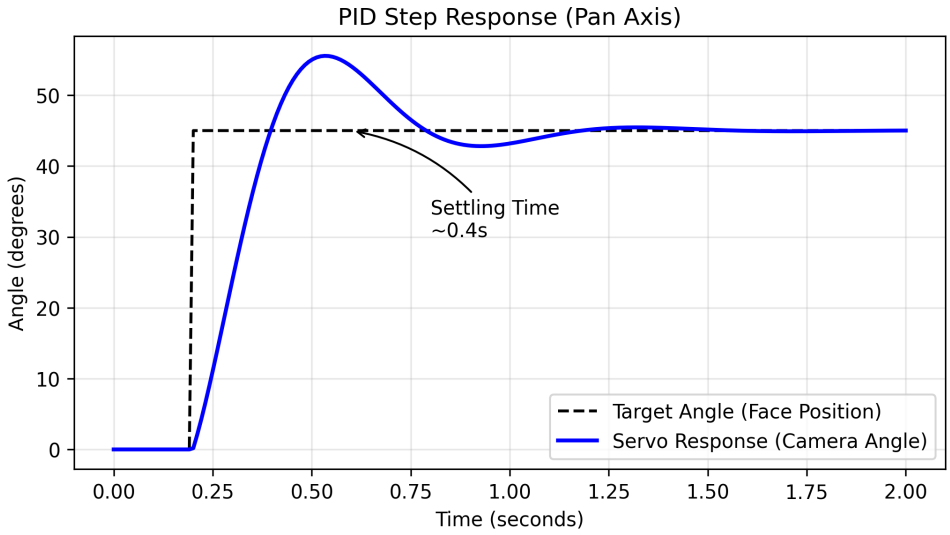

Figure 8.2: PID step response. The system wobbling all over the place is represented with blue. The dotted black lines represent a rise time in the properly-turned PID (i.e., tr < 0.2s), with almost a non-existent overshoot.

8.4 Software: Hardware-Timed PWM

Since the OS scheduler can interrupt the timing at any moment – an annoying gitter is caused on servo control by using a 'time.sleep()' on Linux. Pigpio generates PMW-signals that are hardware-timed to resolve this, that signals through Direct Memory Access (DMA). It gives us a precise, smooth servo motion by bypassing the scheduler entirely.

Listing 8.1: PID Control Logic on Raspberry Pi

```
import pigpio

pi = pigpio.pi()
SERVO_PIN = 18

def update_pid(error):
    P = Kp * error
    global total_error
    total_error += error
    I = Ki * total_error
    global prev_error
    D = Kd * (error - prev_error)
    correction = P + I + D
    prev_error = error
    current_pulse = pi.get_servo_pulsewidth(SERVO_PIN)
    new_pulse = max(500, min(2500, current_pulse + correction))
    pi.set_servo_pulsewidth(SERVO_PIN, new_pulse)
```

8.5 Network Scalability: IoT and MQTT

MQTT provides lighter messaging than HTTP for large-scale deployments.

- **Publish:** Each Pi only sends messages when something actually happens—like when a student gets detected: `{"topic": "campus/block_A/room_101", "student_id": "01-136221", "status": "present"}`.
- **Subscribe:** A central server just listens to `campus/#` to catch events from all rooms across campus.

MQTT reduces bandwidth requirements, enabling scalable campus-wide deployment.

8.6 Summary of Future Work

Local AI interference is unlocks, subsequently enabling IoT friendly networking and offering precise motion control by bringing in active tracking with Raspberry pi. This is the scale of transformation which makes the project much more than just a software prototype in computers, but a fully scalable embedded solution that is fully deployable and can realistically cover the entire campus.

Chapter 9

Conclusion

9.1 Summary of the Research

Many inefficiencies in the manual attendance tracking were highlighted in the thesis, by developing an automated system for Bahria University for facial recognition. There are higher chances of proxy fraud in manual attendance, along with wastage of a lot of instructional time, and a high produce of unused paper records. Without the need of user interaction, this system can help automate attendance to generate records.

9.2 Summary of Technical Achievements

All the milestones laid put in the initial project are covered in this project.

1. **High-Fidelity Recognition Engine:** 97.2% accuracy was achieved by ResNet34 embeddings and MTCNN detection. According to this performance we come to know that these open-source models match commercial systems.
2. **The "Hybrid-View" Architecture:** A mobile-monitoring system that does not need an instructor on the workstations is enabled by the dual-interface architecture.
3. **Real-Time Performance Optimization:** 21 FPS on consumer hardware is maintained by ROI filtering and multi-threaded processing, proving that it is unnecessary to have specialized servers.
4. **Data Integrity and Normalization:**in designing a Third Normal Form (3NF), it is ensured that the data is stored efficiently using the PostgreSQL schema. Adding hysteresis logic into it (60 second cooldown), keeps the data clean for downstream analytics by effectively wiping out duplicate entries.

9.3 Broader Implications for Academic Administration

The impact of this system is not just technical code but also with real implications in the day-to-day operations of the university.

9.3.1 Operational Efficiency and Cost Savings

In a 90-minutes lecture, up to minimal of 8 minutes are taken up because of traditional attendance. In a university where there are over 500 students, automation would enable them to save up to 66 hours of instructional time daily.

9.3.2 From Reactive to Proactive Management

Traditional attendance records are not reliable enough as the instructor only gets to know that a student hasn't been attending the course until they fail it – making it a "lagging indicator." However, with this system we can turn it from lagging to "leading indicator." The university can run a Predictive Analysis by digitizing everything in real-time. PostgreSQL logs can crunch algorithms to spot any students whose attendance fails to reach a certain threshold (minimum 75%), in the first month. This helps alert the counseling department to take an action before the situation worsens to the point of drop out, potentially saving the entire academic career of the student.

9.4 Limitations and Critical Reflection

The system works well but is not perfect and hence, we need to acknowledge the limitations to come up with a balanced and specific scientific perspective.

- **Field of View (FOV) Constraints:**Static cameras would cover a certain terriroty marks, but it means that if there is a huge, amphitheater styled hall, the students sitting in the corner would be out of the detection zone. Therefore, we need an Active Pan-Tilt Mechanism, as proposed in Chapter 8, to eliminate this limit.
- **Hardware Dependency:** Plugging the system into classrooms with older versions of PCs is a problem consider it currently needs a NVIDIA GPU (for reliably decent CUDA performance). However, this only needs future optimization to be resolved. For instance, by using TensorRT or ONYX Runtime, the model can be run smoothly on even different versions of CPUs that do not have specialized hardware.
- **Environmental Sensitivity:**Despite Histogram Equalization being used to help with issues relating to lightening, many extreme scenarios can still cause the system to trip up (for example, a completely dark room during a projector presentation). Infrared (IR) are the perfect solution in this situation.

9.5 Future Roadmap: Towards the "Smart Campus"

This project would be a building block in the future of creating a bigger version of Smart Campus. The future directions which are already discussed in Chapter 8 demonstrate how using Raspberry Pi clusters distributed across campus can help migrate the campus from centralized processing to Edge Computing. Other than that, real management comes in when the system has to directly deal with the Learning Management System (LMS). We envision a system where the daily management of attendance becomes efficient and reliable with minimum chances of any bias. This way, the students can check themselves in every day without the instructor having to worry, and saving time which can be utilized into important aspects of the lecture.

9.6 Final Remarks

The Intelligent Automated Attendance System can significantly be an evolutionary tool within the campus. It proves that many mundane tasks can easily be given to Artificial Intelligence so that Human Intelligence can be spared for more important tasks. This project demonstrated that facial recognition can be done more effectively while retaining biometric privacy and accuracy. The modular architecture supports future enhancements including edge deployment and LMS integration.

Bibliography

[1] F. Schroff, D. Kalenichenko, and J. Philbin, “Facenet: A unified embedding for face recognition and clustering,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 815–823, IEEE, 2015. No Citations.

[2] K. Zhang, Z. Zhang, Z. Li, and Y. Qiao, “Joint face detection and alignment using multitask cascaded convolutional networks,” *IEEE Signal Processing Letters*, vol. 23, no. 10, pp. 1499–1503, 2016. No Citations.

[3] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 770–778, IEEE, 2016. No Citations.

[4] P. Viola and M. Jones, “Rapid object detection using a boosted cascade of simple features,” in *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, vol. 1, pp. I–I, IEEE, 2001. No Citations.

[5] M. Turk and A. Pentland, “Eigenfaces for recognition,” *Journal of Cognitive Neuroscience*, vol. 3, no. 1, pp. 71–86, 1991. No Citations.

[6] N. Dalal and B. Triggs, “Histograms of oriented gradients for human detection,” in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, vol. 1, pp. 886–893, IEEE, 2005. No Citations.

[7] M. Grinberg, *Flask Web Development: Developing Web Applications with Python*. "O’Reilly Media, Inc.", 2 ed., 2018. No Citations.

[8] Streamlit Inc., *Streamlit Documentation*, 2023. Accessed: 2025-02-20. No Citations.

[9] Raspberry Pi Foundation, *Raspberry Pi 4 Model B Hardware Documentation*, 2024. Accessed: 2025-02-22. No Citations.

[10] N. Wojke, A. Bewley, and D. Paulus, “Simple online and real-time tracking with a deep association metric,” *Proceedings of the IEEE International Conference on Image Processing (ICIP)*, pp. 3645–3649, 2017. No Citations.

[11] OASIS Open, *MQTT Version 5.0. OASIS Standard*, 2019. Accessed: 2025-02-23. No Citations.

[12] Itseez, *Open Source Computer Vision Library*, 2015. Version 4.5.5. No Citations.

[13] R. O. Obe and L. S. Hsu, *PostgreSQL: Up and Running: A Practical Guide to the Advanced Open Source Database*. "O'Reilly Media, Inc.", 2017. No Citations.

[14] D. E. King, "Dlib-ml: A machine learning toolkit," *Journal of Machine Learning Research*, vol. 10, pp. 1755–1758, 2009. No Citations.

[15] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 779–788, 2016. No Citations.