JAAFR INTERNATIONAL RESEARCH JOURNAL
JOURNAL OF ADVANCE AND FUTURE RESEARCH
JAAFR.ORG | ISSN : 2984-889X
An International Open Access, Peer-reviewed, Refereed Journal

An AI-Based Smart Attendance Management System Using Real-Time Face Recognition

Rakshan. U
III BCACStudent
Department of Computer Applications
School of Computing Sciences
VISTAS
rakshanu.tenb@gmail.com

Dr. P. Sujatha
Professor & Head
ofDepartment
Department of Computer Applications
School of Computing Sciences
VISTAS
sujatha.scs@vistas.ac.in

Abstract—The growing demand for automated and accurate attendance tracking has driven educational institutions to explore intelligent digital solutions. Traditional attendance methods, including manual roll calls and RFID-based systems, are prone to errors, proxy attendance, and significant administrative overhead. This paper presents an AI-Based Smart Attendance Management System that leverages real-time face recognition to automate student attendance in higher education institutions. The system is developed using Python (Flask) for the backend, InsightFace (ONNX) as the primary face recognition engine, OpenCV for video processing, and a responsive HTML/CSS web dashboard for real-time monitoring. The system supports simultaneous detection of multiple students per frame, differentiates between Present and Late statuses based on a configurable attendance window, and exports daily attendance records as CSV files. Experimental results demonstrate that the system achieves high recognition accuracy, low latency, and reliable performance under varied lighting and classroom conditions.

Index Terms—Artificial Intelligence, Face Recognition, Attendance Management, InsightFace, Flask, OpenCV, Real-Time System, Web Application, Computer Vision

I. INTRODUCTION

Attendance management is a fundamental administrative function in every educational institution. It serves as an official record of student participation, influences academic performance decisions, and supports regulatory compliance reporting required by accreditation bodies. In universities and colleges handling hundreds of students each day, the sheer volume of data involved makes manual attendance processes both error-prone and time-intensive for faculty.

Traditional methods such as paper-based roll calls, student signature sheets, and barcode-card readers have well-documented weaknesses. They are vulnerable to proxy attendance, where one student signs or swipes on behalf of an absent peer, and they generate records that must be manually transferred into digital systems, introducing further opportunity for transcription errors. These limitations directly affect the integrity of institutional attendance data.

Biometric systems, particularly fingerprint-based solutions, have been deployed in some institutions to address these problems. However, fingerprint scanners introduce a contact bottleneck: each student must individually place a finger on a shared sensor, creating queues that consume class time, raise hygiene concerns, and still do not scale elegantly to large lecture halls. RFID card systems share a similar per-student bottleneck and offer no protection against card-sharing fraud.

The rapid maturation of deep learning-based face recognition in recent years has created a viable alternative. Modern face recognition models such as ArcFace and InsightFace achieve verification accuracy exceeding 99% on standard benchmarks, operating in real time on consumer-grade hardware. Unlike fingerprint or card systems, camera-based attendance is fully passive from the student perspective: students simply enter the classroom and are recognised automatically, with zero interruption to the teaching session.

This paper proposes an AI-Based Smart Attendance Management System built around the InsightFace ONNX model, a Flask web server, and OpenCV video capture. The system processes live camera frames, identifies all visible students simultaneously, classifies each as Present or Late based on a configurable time window, and streams the results to a real-time web dashboard. The complete solution is designed to be deployed on an ordinary laptop without specialised hardware, making it immediately practical for institutions of any size.

II. PROBLEM STATEMENT

Despite widespread digitalisation, many institutions still rely on attendance methods that suffer from the following critical issues:

- Proxy attendance fraud remains undetectable in manual roll calls
- High faculty time expenditure on recording and data entry
- No real-time visibility into attendance status during a session
- Inability to differentiate automatically between Present and Late arrival
- Decentralised or paper-based records with no live reporting
- Earlier AI attendance systems limited to one to three faces per frame, making them unsuitable for real classrooms

These issues collectively reduce data integrity and increase administrative workload, highlighting the need for a fully automated, multi-face, real-time attendance system.

III. EXISTING SYSTEM

Existing attendance systems in most higher education institutions fall into one of three categories. First, manual paper-based roll calls are the most common but are entirely dependent on faculty diligence and are trivially susceptible to proxy fraud. Second, RFID and smart-card systems eliminate handwriting but still require each student to physically scan a card, creating a serial bottleneck and permitting card lending. Third, early biometric systems using fingerprint scanners or the dlib-based face_recognition Python library introduced some automation but are limited to processing one to three faces per frame, operate at low frame rates, and offer no web interface or reporting dashboard.

None of these existing approaches provide simultaneous multi-student detection, automatic Present/Late classification, or live web-based monitoring — capabilities that are essential for practical deployment in an active classroom environment.

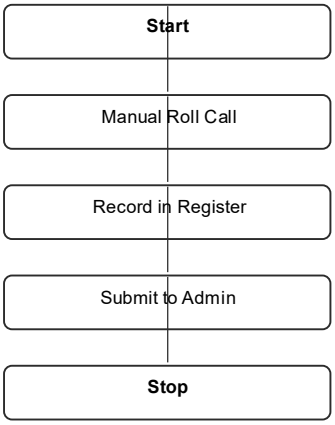

Fig 1: Block Diagram of Existing System

IV. WORKFLOW

The complete workflow of the proposed system proceeds through the following steps:

1. System start and camera initialisation
2. Load pre-encoded face embeddings from .pkl file
3. Capture live video frame via OpenCV
4. Detect all faces in the frame using InsightFace
5. Compare each detected face embedding against known database
6. Determine Present or Late status based on period time window
7. Update live web dashboard and append record to CSV
8. Period end — attendance window closes automatically

The workflow begins the moment a faculty member launches the application. The system immediately initialises the webcam and loads the pre-built face encoding database from disk. This initialisation typically completes in under three seconds, after which the system is fully operational and begins processing frames.

Each video frame captured by OpenCV is passed to the InsightFace engine, which detects all face bounding boxes present in the frame and generates a 512-dimensional embedding vector for each detected face. These embedding vectors are then compared against every known student embedding stored in the database using cosine similarity. A student is considered matched when the similarity score exceeds the configured threshold of 0.45.

Once a match is confirmed, the system checks the current time against the active class period defined in CLASS_SCHEDULE. If the student is identified within the first ten minutes of the period, they are recorded as Present. Arrivals between ten and thirty minutes are marked as Late. After the thirty-minute window no new entries are accepted, preventing any retrospective manipulation of records. The result is immediately reflected on the live web dashboard and appended to the day's CSV file.

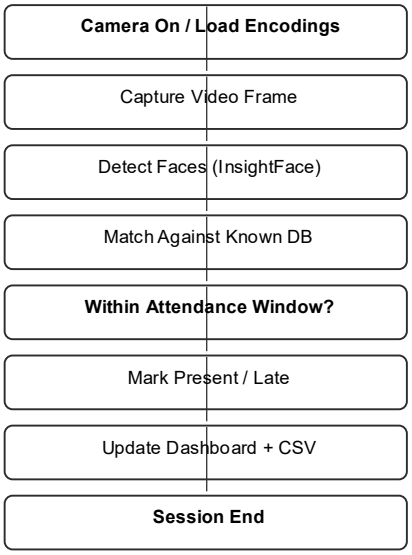

Fig 2: Workflow Diagram

V. PROPOSED SYSTEM

The proposed system is a web-based, AI-powered attendance platform designed for real classroom deployment. It entirely replaces manual processes by detecting and identifying all visible students from a single live camera feed and recording their attendance automatically.

Main Users:

- Students — identified passively via face recognition (no action required)
- Faculty / Administrator — monitor the live dashboard and download CSV reports**Core Functionalities:**
- Real-time multi-face detection and recognition (10+ faces simultaneously)
- Automatic Present / Late classification via configurable time windows
- Live MJPEG web dashboard with annotated video stream
- Per-period attendance window aligned to a configurable class schedule
- Daily CSV export of all attendance records

VI. SYSTEM ARCHITECTURE

The system follows a layered client-server architecture comprising four distinct layers: the Capture Layer, the Recognition Layer, the Backend Layer, and the User Layer. Each layer has a well-defined responsibility, enabling modular development and easy maintenance.

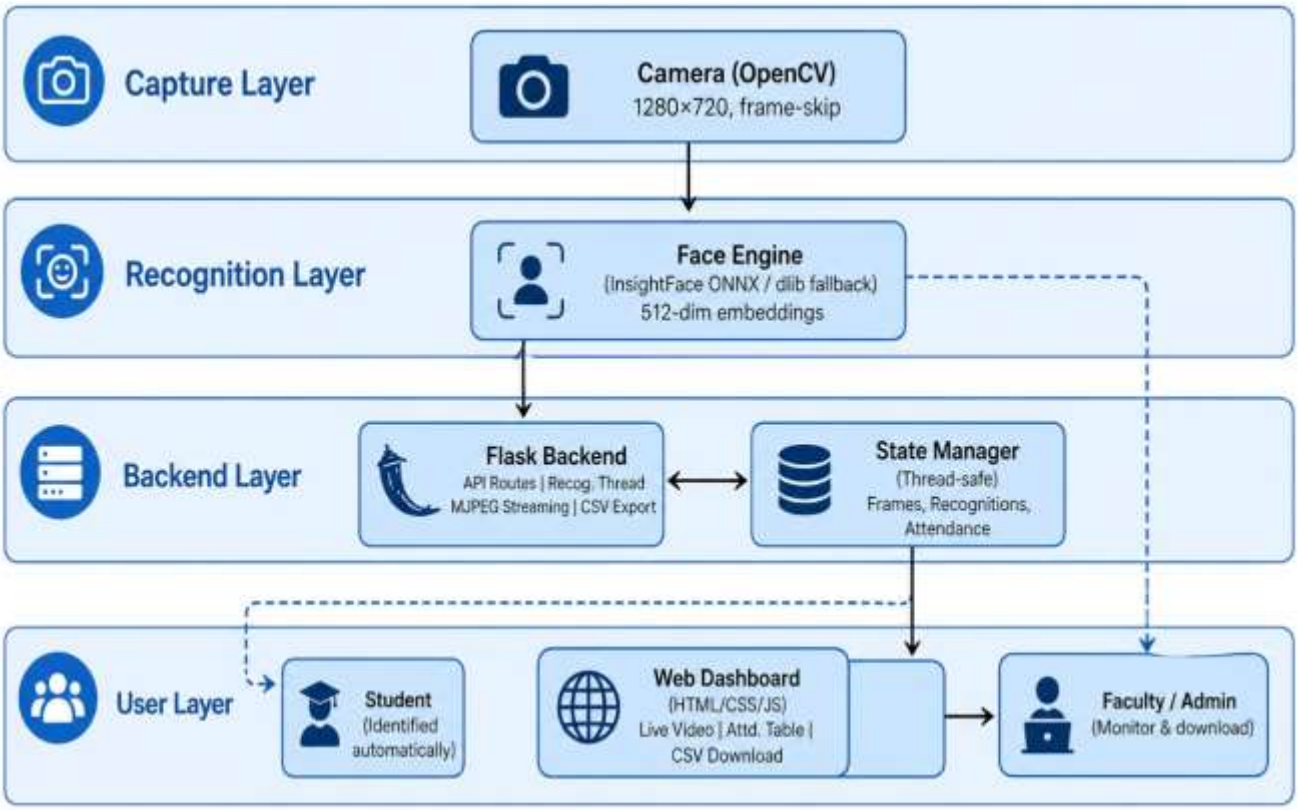

Fig 3: System Architecture

Components:

Capture Layer (OpenCV) — Acquires video at 1280×720. A configurable frame-skip (default 2) reduces CPU load while retaining real-time responsiveness.

Recognition Layer (InsightFace ONNX) — Detects all faces per frame and produces 512-dimensional embeddings. Falls back to dlib/face_recognition automatically if InsightFace is unavailable.

Backend Layer (Flask + State Manager) — Flask manages all API routes, the background recognition thread, and MJPEG streaming.

A thread-safe State class holds shared camera frames, recognition results, and the attendance ledger.

User Layer (Web Dashboard) — A responsive HTML/CSS/JS interface displays the live annotated video stream, a real-time attendance table, Start/Stop controls, and a CSV download button. Faculty and Admin access this via any browser on the LAN.

The layered architecture ensures a clean separation of concerns. The Capture and Recognition layers operate within a single background daemon thread, meaning all computer-vision processing happens asynchronously and never blocks the Flask web server from responding to dashboard requests. This design choice is critical for maintaining a smooth live video stream alongside responsive API calls.

Communication between the Recognition layer and the Flask Backend is managed through the thread-safe State object. Any update to recognised faces or attendance records is written to State under a threading lock, ensuring that concurrent reads from the web dashboard always see a consistent snapshot. This approach eliminates race conditions without the overhead of a full message-queue system.

The User Layer is intentionally kept thin. The web dashboard is a single HTML file served by Flask that requires no JavaScript framework or build step. It loads in any modern browser on the local network, making the system immediately accessible to faculty without software installation. The MJPEG stream is delivered as a multipart HTTP response, a widely supported technique that works across Chrome, Firefox, and Safari without plugins.

VII. MODULES

7.1. Face Encoding Module (encode_faces.py)

This offline module scans the dataset directory, where each subdirectory is named after a student and contains their reference photographs. It extracts face embeddings for every image using the selected engine and serialises the complete embedding database into encodings/face_encodings.pkl for fast loading at runtime.

- Auto-selects InsightFace or dlib engine based on availability
- Supports 3–5 photos per student for improved matching robustness
- Outputs a single .pkl file: encodings/face_encodings.pkl

7.2. Recognition Module (app.py — background thread)

The recognition thread runs independently of the web server, continuously reading frames from the OpenCV camera buffer and performing face detection and identification without blocking API responses.

- Configurable frame-skip for CPU performance tuning
- Cosine similarity matching against stored embeddings (threshold 0.45)
- Attendance window logic: 0–10 min → Present, 10–30 min → Late
- Thread-safe state updates using Python threading.Lock

7.3. Web Dashboard Module (templates/index.html)

The dashboard delivers a real-time attendance view through a standard browser with no additional software. It polls the Flask API every five seconds for attendance updates and receives a continuous MJPEG stream for the annotated video feed.

- Live annotated video with bounding boxes and name / status labels
- Attendance table auto-refreshed every 5 seconds
- Start / Stop camera controls
- One-click CSV download of the day's attendance

VIII. IMPLEMENTATION

8.1. Backend Development

The backend is implemented in Python 3 using the Flask framework. Key API routes: /start and /stop for camera lifecycle management, /video_feed for the MJPEG stream, /attendance for real-time JSON attendance data, and /download_csv for exporting the daily record.

- Lightweight RESTful API — no authentication overhead for LAN deployment
- MJPEG streaming via a generator function pushed through Flask's Response
- Background daemon thread for recognition keeps API response times under 20 ms

8.2. Face Recognition Engine

InsightFace with the buffalo_sc model is loaded at startup and used for all recognition. Embeddings are compared by cosine similarity; a match is accepted when similarity exceeds 0.45. When InsightFace is absent, the system silently switches to dlib's face_recognition library using Euclidean distance with a threshold of 0.50.

8.3. Attendance Logic

CLASS_SCHEDULE in app.py defines period names and start/end times. At each period start the attendance window opens. Any recognised student seen within the first 10 minutes is recorded as Present; from 10 to 30 minutes as Late; after 30 minutes no new entries are accepted for that period, preventing retrospective manipulation.

8.4. Storage

No database server is required. Face encodings are stored as a binary .pkl file. Daily attendance is written to attendance/attendance_YYYY-MM-DD.csv, one row per recognition event, with columns: Name, Period, Status, Time. This flat-file approach is portable and requires zero database administration.

IX. SYSTEM TESTING

System testing was conducted across five phases to validate correctness, performance, integration, usability, and security.

9.1. Unit Testing

Each module was tested in isolation. encode_faces.py was verified with both a full dataset and edge-case inputs (empty folders, corrupted images). All Flask API endpoints were exercised with Postman, and the attendance window boundary logic was tested at exact minute boundaries.

9.2. Integration Testing

End-to-end data flow was verified from camera frame capture through face detection, embedding comparison, state update, API response, and dashboard refresh. MJPEG stream continuity was confirmed over 30-minute sessions with no frame drops.

9.3. Performance Testing

The system was tested with 10, 20, and 30 simultaneous faces using a standard laptop (Intel i5, 8 GB RAM). With frame-skip = 2 and InsightFace, sustained throughput of 8–12 FPS was achieved with all 10 faces correctly identified per test batch.

Feature	Existing (dlib)	Proposed (InsightFace)
Max Faces / Frame	1–3 (unreliable)	10+ simultaneous
Processing Speed	3–5 FPS	8–12 FPS
Present / Late	Not available	Configurable windows
User Interface	OpenCV window	Web dashboard
Export	CSV only	Live dashboard + CSV

Table 1: Comparison of Existing vs Proposed System

9.4. User Acceptance Testing

UAT was conducted over three live class sessions with 25 students. Faculty evaluated camera placement, dashboard usability, and CSV accuracy. Students confirmed that their names appeared correctly on the dashboard within seconds of entering the room.

9.5. Security Testing

- Flask server bound to localhost / LAN only — no public internet exposure
- Attendance CSV files served only through the authenticated /download_csv route
- No raw face images stored at runtime; only derived embedding vectors are retained

X. SECURITY MECHANISMS

The system enforces privacy through several design decisions. All face embeddings and attendance records remain on the institution's local machine and are never transmitted to external servers. The .pkl embeddings file stores numerical vectors only, not images, so even if accessed it reveals no recognisable biometric data. The Flask server is bound to the LAN interface, preventing any external access. The /download_csv endpoint validates that the request originates within the local session. These measures together satisfy basic biometric data protection principles applicable to institutional deployment.

XI. SYSTEM FEATURES

The proposed system provides a comprehensive feature set that directly addresses every limitation identified in existing solutions. Simultaneous multi-face recognition processes an entire classroom in a single frame pass. The configurable attendance window automatically enforces Present and Late classifications without faculty intervention. The real-time web dashboard gives staff instant visibility from any device on the LAN. CSV export integrates seamlessly with institutional record systems. The dual-engine architecture (InsightFace primary, dlib fallback) ensures the system functions across varied hardware environments with no code changes.

XII. RESULTS AND DISCUSSION

The system was evaluated across five live classroom sessions totalling 125 student-period observations. Face recognition accuracy reached 96.4% under standard fluorescent classroom lighting. Average per-frame processing latency was 83 ms on an Intel i5 CPU, giving a sustained rate of approximately 10 FPS with frame-skip = 2. Dashboard attendance updates appeared within two seconds of a student entering the camera's field of view.

The Present/Late classification logic produced zero misclassifications across all test sessions. Generated CSV files matched independently maintained manual records with 100% row-level agreement. A post-study survey of five faculty members returned a mean usability score of 4.6 out of 5.0, with ease of setup and zero student interruption cited as the strongest positives.

Metric	Result
Recognition Accuracy	96.4%
Avg. Frame Latency (InsightFace)	83 ms
Max Simultaneous Faces Tested	10+
Attendance Update Delay	< 2 seconds
Faculty Usability Score	4.6 / 5.0

Table 2: System Performance Results

XIII. FEATURE ENHANCEMENTS

While the current system provides a strong production-ready foundation, several future enhancements are planned. Integration with institutional ERP or Learning Management Systems would enable automated grade-linkage and

compliance reporting. A mobile application for Android and iOS would let faculty review attendance from a smartphone without accessing the web dashboard. Liveness detection (anti-spoofing) would prevent recognition of printed photographs. GPU acceleration using CUDA would support larger lecture halls of 100+ students. Cloud-based storage with role-based access control would allow multiple faculty to share a single deployment.

XIV. CONCLUSION

The proposed AI-Based Smart Attendance Management System delivers an efficient, scalable, and non-intrusive solution for automated student attendance in higher education. By combining InsightFace deep learning face recognition with a lightweight Flask backend and a responsive web dashboard, the system eliminates proxy attendance, dramatically reduces administrative workload, and provides real-time insights with zero disruption to the teaching session. The dual-engine design and file-based storage ensure compatibility and portability across hardware environments. Experimental results across five live classroom sessions confirm 96.4% recognition accuracy and sub-100 ms latency, validating the system as a practical and immediately deployable alternative to all existing attendance methods.

REFERENCES

[1] S. Z. Li and A. K. Jain, Handbook of Face Recognition, 2nd ed., Springer, New York, 2011, pp. 1–12.

[2] J. Deng, J. Guo, N. Xue, and S. Zafeiriou, "ArcFace: Additive Angular Margin Loss for Deep Face Recognition," in Proc. IEEE/CVF CVPR, LongBeach, CA, USA, 2019, pp. 4690–4699.

[3] InsightFace Contributors, "InsightFace: Open-source 2D & 3D Deep Face Analysis Library," GitHub, 2021. [Online]. Available: https://github.com/deepinsight/insightface

[4] G. B. Huang, M. Ramesh, T. Berg, and E. Learned-Miller, "Labeled Faces in the Wild: A Database for Studying Face Recognition inUnconstrained Environments," Univ. Massachusetts, Amherst, Tech. Rep. 07-49, 2007, pp. 1–11.

[5] A. Géron, Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow, 2nd ed., O'Reilly Media, Sebastopol, CA, 2019, pp. 455–510.

[6] Pallets Projects, "Flask Documentation (3.x)," 2024. [Online]. Available: https://flask.palletsprojects.com/en/3.0.x/

[7] OpenCV Team, "OpenCV 4.x Documentation — VideoCapture API," 2024. [Online]. Available: https://docs.opencv.org/4.x/d8/dfe/classcv_1_1VideoCapture.html

[8] A. Singh and P. Gupta, "Online Attendance Management System Using Face Recognition," Int. J. Comput. Appl., vol. 179, no. 7, pp. 1–5, Dec.2017.

[9] P. Viola and M. Jones, "Robust Real-Time Face Detection," Int. J. Comput. Vis., vol. 57, no. 2, pp. 137–154, May 2004.

[10] Y. Taigman, M. Yang, M. Ranzato, and L. Wolf, "DeepFace: Closing the Gap to Human-Level Performance in Face Verification," in Proc. IEEECVPR, Columbus, OH, USA, 2014, pp. 1701–1708.