An Application on Machine Learning and Computer Vision for Automated Attendance Tracking Solution

Bala Krishna Chennaiah[1], Dr.Vanitha Kakollu[2]
[1]PG Student, [2]Assistant Professor
Department of Computer Science, GSS, GITAM Deemed to be University, Visakhapatnam

How to Cite this Article:
Chennaiah, B. K. (2026). An Application on Machine Learning and Computer Vision for Automated Attendance Tracking Solution. International Journal of Creative and Open Research in Engineering and Management, 02(04). https://doi.org/10.55041/ijcope.v2i4.260

https://doi.org/ 10.55041/ijcope.v2i4.260

Abstract

Marking attendance might seem like the simplest task in a classroom, but anyone who has sat through a ten-minute roll call at the start of a lecture, or tried to maintain fair records while students mark each other present, knows how broken the process really is. This paper presents FaceTrack Pro, an automated attendance tracking system built using Machine Learning and Computer Vision that was designed to fix exactly these problems. The system uses RetinaFace for real-time face detection, ArcFace R100 for face recognition with a benchmark accuracy of 99.83% on the LFW dataset, and MediaPipe FaceMesh for liveness verification through eye-blink detection which means no one can fool the system by holding up a photograph. It runs entirely on a regular computer, needs no internet connection after the first setup, and stores all data as plain CSV files that any staff member can open in Excel without needing a database. When tested with forty students across different classroom environments, the system achieved 99.1% recognition accuracy, detected faces correctly in 97.2% of frames, and marked attendance in an average of 1.8 seconds per student. FaceTrack Pro is not just a research prototype it is a working system that can be deployed in a real institution today.

Keywords:
Face Recognition, Automated Attendance, RetinaFace, ArcFace R100, MediaPipe FaceMesh, Liveness Detection, Eye Aspect Ratio, Computer Vision, Deep Learning, InsightFace, Python, Anti-Spoofing, Student Tracking

1. Introduction

Walk into almost any college in India and ask how attendance is being tracked. Most of the time you will find a faculty member calling out names from a printed list, or a sheet being passed around the room. These methods have been in use for decades. They also fail in the same ways they have always failed names get missed, sheets get lost, and students mark each other as present without a second thought. The technology to solve this problem has existed for years. The gap has always been in bringing that technology into a form that a real institution can actually use. The issue runs deeper than inconvenience. Attendance records directly affect student eligibility for exams, scholarships, and academic standing. When those records are unreliable, the consequences fall on everyone honest students get the same treatment as chronic absentees, faculty waste time chasing paperwork, and administrators make consequential decisions on data they cannot fully trust. Proxy attendance, where an attending student marks an

absent friend as present, is so common it is almost accepted. That should not be the case. Modern face recognition has matured to the point where it outperforms human recognition in controlled settings and holds up impressively in real-world conditions too. Models like RetinaFace can find faces across a camera frame in under 50 milliseconds. ArcFace can match an identity against hundreds of enrolled individuals almost instantly. These are not lab demonstrations anymore they run on a standard laptop CPU without needing expensive GPUs. The question is no longer whether the technology is good enough. The question is whether it can be packaged in a way that a non-technical school administrator can actually set up and use.That is the central motivation behind FaceTrack Pro. This project takes those powerful ML tools and wraps them inside a simple desktop application with a clear graphical interface, one-click installation, and data storage in plain CSV files. No database setup, no cloud account, no IT department required. A teacher or administrator can install it, register students, and start marking attendance in under an hour.

1.1 Why This Problem Matters

It is easy to underestimate how much time is lost to manual attendance every semester. If a single faculty member teaches five classes a week with fifty students each, and roll call takes eight minutes per class, that is forty minutes every week or roughly twenty-five hours over a full semester spent on something a computer can handle in seconds. Multiply that across an entire institution and the numbers become impossible to ignore. Beyond time, there is the matter of accuracy. A human calling out names makes mistakes. Students answer for others. Sheets get filled in incorrectly. Digital fingerprint scanners help, but they introduce hygiene concerns and regularly fail when fingertips are wet, worn, or slightly different from the registered sample. Face recognition, combined with liveness detection, eliminates most of these failure modes. The student has to be physically present and has to blink there is no workaround that does not require their actual face.

1.2 Objectives of the Project

The project was built around five clear goals. First, to build a face detection and recognition pipeline accurate enough for production use on standard hardware. Second, to include liveness detection that makes proxy attendance practically impossible. Third, to wrap everything in an interface simple enough for non-technical administrators. Fourth, to store all data in a format that does not require database knowledge to access or audit. Fifth, to validate the system under real classroom conditions and document its actual performance honestly.

1.3 Scope of the System

FaceTrack Pro covers the full attendance workflow from start to finish. Students are registered once using five face photos captured through the camera. During attendance sessions, the system runs in the background, identifies faces in real time, checks that they are live, and writes the record automatically. The administrator can review today's log on screen, filter by class, mark students as on approved leave, add institutional holidays, generate Excel reports, and export everything for submission. A full audit log captures every action taken in the system, so there is always a trail to follow if a record is ever disputed. The system is built for Windows deployment because that is what most Indian educational institutions run, but since it is pure Python it can run on Linux or macOS with no code changes. A web version using Flask is included for institutions that want browser access without installing the desktop application.

2. Algorithm Used

FaceTrack Pro works in stages. Every frame from the camera goes through face detection first, then recognition, then liveness verification, and finally the attendance record is written only if all three stages pass. Each stage uses a different algorithm chosen specifically because it is the best available option for that task. The choices were not arbitrary each one was compared against alternatives and selected based on measurable performance differences.

A. RetinaFace Face Detection

RetinaFace is the detection engine. Its job is to find faces in a camera frame and return their exact locations along with five facial landmark points the two eye corners, nose tip, and two mouth corners. These landmark points are what allow the next stage to align the face precisely before trying to recognise it.What makes RetinaFace better than older detectors like Haar cascades or HOG-based methods is that it uses a Feature Pyramid Network, which means it processes the image at multiple scales simultaneously. It can find a face that is close to the camera and taking up most of the frame at the same time as it finds a face that is small and far away, in the same single pass. It also works well at angles, in low light, and when faces are partially covered. On the WiderFace Hard benchmark which is designed to test detectors on difficult real-

world images it achieves above 96% detection rate. In a classroom, that kind of robustness matters.

The detection process works as follows:

A video frame is captured from the camera as a numpy array in BGR format using OpenCV's VideoCapture. The frame is passed to the InsightFace FaceAnalysis object, which runs RetinaFace as its detection stage.The Feature Pyramid Network builds multi-scale feature maps and generates anchor box proposals at each scale. Non-Maximum Suppression removes duplicate bounding boxes, keeping only the highest-confidence detection per face. Five facial landmarks are returned alongside each bounding box. These are used to align the face before recognition.Detections below the confidence threshold (default 0.50) are dropped. Everything above passes to ArcFace.

B. ArcFace R100 Face Recognition

Once a face has been found and aligned, ArcFace R100 takes over. ArcFace works by converting a face image into a 512-number vector called an embedding that captures the unique shape and texture features of that face. When two images of the same person are processed, their embeddings end up close together in mathematical space. When two different people are processed, their embeddings end up far apart. Recognition is simply a matter of measuring how close the live embedding is to each stored embedding and finding the nearest match. ArcFace gets its accuracy from the way it was trained. The loss function used during training the Additive Angular Margin Loss specifically pushes the network to make embeddings for the same person cluster tightly together while keeping embeddings for different people as far apart as possible. The result is a 99.83% verification accuracy on LFW, the standard face recognition benchmark. The R100 variant uses a ResNet-100 backbone, which is a deep 100-layer convolutional network that has learned to extract the most discriminative features from a face image.

The recognition process runs as follows:

The aligned 112x112 face crop from RetinaFace is passed through the ResNet-100 backbone.A raw 512-dimensional feature vector is extracted from the final fully connected layer. The vector is L2-normalised so it lies on a unit sphere in 512-dimensional space this makes cosine distance equivalent to the angle between vectors.The cosine distance is computed between the live embedding and every stored student embedding using numpy dot products.The student with the smallest distance is the candidate match. If that distance is below the threshold (default 0.40), the student is recognised. If not, the face is flagged as unknown.Recognised students are passed to the liveness stage. Unknown faces are saved to the UnknownFaces folder for the administrator to review later.

C. MediaPipe FaceMesh Liveness Detection

Recognition alone is not enough to prevent proxy attendance. Someone could just hold up a photograph. That is where liveness detection comes in, and it is one of the most important parts of this system. MediaPipe FaceMesh estimates 468 facial landmark points in three dimensions from a regular 2D image. From those landmarks, the Eye Aspect Ratio (EAR) can be calculated frame by frame. EAR is a simple geometric ratio between the vertical and horizontal distances across the eye. When a person blinks, the eye closes and the EAR drops sharply, then rises again when the eye opens. A photograph or screen replay cannot do this the eye region in a photo is fixed and produces a constant EAR with no variation. So the rule is simple: if the EAR does not dip below the blink threshold and then recover at least once, the face does not pass liveness and attendance is not marked.

The liveness check works as follows:

MediaPipe FaceMesh is initialised with refine_landmarks=True for high-precision eye landmark tracking. For each recognised face, the face crop is converted to RGB and passed to FaceMesh. The six landmark indices for each eye two horizontal corners and four vertical eyelid points are extracted. EAR is calculated as (||p2-p6|| + ||p3-p5||) / (2 x ||p1-p4||) for both eyes and averaged together. The EAR value is tracked across frames. A blink is confirmed when EAR drops below 0.21 for at least two frames and then rises above it again. Only after a confirmed blink is recorded does the system approve the face for attendance marking. Photos and videos fail this check every time.

D. User Authentication

The admin panel is protected by a simple but effective login system. When the application starts, it shows a login screen asking for a username and password. The entered password is hashed using SHA-256 and compared against the stored hash in the settings file. If they match, the admin gets in. If not, the attempt is logged with a timestamp and the system resets the form. After five consecutive failures, the system enters a short lockout to discourage guessing. Administrators can change the password any time through the Settings panel, and the new hash is saved immediately.

E. Attendance Recording Logic

After a face passes recognition and liveness, one more check happens before the record is written. The system looks at a cooldown table a dictionary of student IDs mapped to the time they were last marked. If the same student was marked within the last 15 seconds (configurable), the current detection is ignored. This prevents a student standing in front of the camera from being marked twice in the same session. If the cooldown check passes, the system compares the current time against the class start time plus the configured late threshold. If the student is arriving after that window, they are marked late and the number of late minutes is recorded alongside the attendance entry. The final record student ID, name, class, date, time, and late status is written to the daily CSV file immediately.

F. Why Not Other Approaches?

Several alternatives were considered and tested before settling on this stack. The face_recognition Python library, which is based on dlib's ResNet-34 model, is widely used and easy to set up, but its LFW accuracy sits around 99.38% nearly half a percentage point below ArcFace R100. That difference may sound small, but in a system that runs every day with the same students, it adds up to noticeably more wrong recognitions over time. OpenCV's DNN face detector is faster than RetinaFace but produces more false positives in backgrounds with complex textures. Haar cascades are too unreliable at non-frontal angles to be trusted in a real classroom. The DeepFace library wraps several models but abstracts too much control away. The final choice InsightFace buffalo_l with RetinaFace and ArcFace together gives the best accuracy, decent speed, and the convenience of a single install.

3. Methodology

The system is built around a straightforward flow: set it up once, enrol students, then let it run during attendance sessions with minimal intervention. There is no complicated configuration required after the first install, and no ongoing maintenance beyond occasional backups. The methodology is described below in the order that a new user would actually experience it.

3.1 First-Time Setup

When FaceTrack Pro is launched for the first time, it checks that all the required data folders exist and creates any that are missing. It then looks for the InsightFace buffalo_l model files in the local cache. If they are not there which they will not be on a fresh machine it downloads them automatically. The download is about 300 MB and only happens once. After that, the models load from disk every time the application starts. The administrator then fills in the institution settings: name, class start time, late arrival threshold, minimum attendance percentage for alerts, and how many photos to capture during student registration. These are saved and automatically loaded on every subsequent launch.

3.2 Student Enrolment

Registering a student is a guided process that takes about two minutes. The administrator enters the student's name, class, email, and phone number, then clicks Register. The camera activates and the system begins capturing face photos five by default, though this can be set anywhere from three to ten in the settings. For each frame where a face is detected with sufficient confidence, an ArcFace embedding is extracted and added to a list. Frames with no face or poor detection quality are automatically skipped and retried without counting toward the total.Once enough good photos have been captured, the individual embeddings are averaged together to produce one representative embedding for that student. Averaging works better than using a single photo because it smooths out the natural variation in facial appearance different expressions, slight changes in lighting between frames, minor head movements. The final averaged embedding is saved as a .npy file. The student's metadata goes into students.csv. From that point on, the student can be recognised in any attendance session.

3.3 Running an Attendance Session

Attendance can be taken in two ways. Auto mode runs the camera continuously in the background and marks students as they appear in frame useful at a classroom entrance or for a fixed camera setup. Manual mode lets the administrator trigger a single recognition pass by pressing a button, which suits smaller classes where the teacher wants more direct control. In both modes, every processed frame goes through the full pipeline: detect, recognise, verify liveness, check cooldown, write record. The camera feed on screen shows green boxes around recognised and verified students, red boxes around unknown faces, and yellow boxes around faces that have been recognised but are still waiting for a blink. Students who have already been marked get a small tick overlay on their bounding box so the administrator can see at a glance who has been recorded. The attendance log on the right side of the screen updates in real time.

3.4 Leave and Holiday Management

One thing that makes FaceTrack Pro more honest than most attendance trackers is how it handles leaves and holidays. In most systems, if a student has an approved medical leave for three days but those days still count as absences, the attendance percentage comes out wrong and the student gets penalised unfairly. FaceTrack Pro keeps a separate leave record for each student and a holiday calendar for the institution. When computing attendance percentages, those approved days are subtracted from the denominator. The result is a percentage that actually reflects whether a student attended when they were supposed to.

3.5 Data Storage and Backup

All data lives in plain CSV files organised in clearly named folders. There is no database. The main reason for this choice is accessibility if an administrator needs to check a specific record, correct a mistake, or share the data with someone else, they can just open the file in Excel. No SQL queries, no database drivers, no technical knowledge needed. The system includes a Backup function that zips all the data folders into a timestamped archive. Restoring from a backup is just a matter of unpacking the zip.

4. Architectural Components

FaceTrack Pro is split into six modules, each responsible for one clearly defined part of the system. They talk to each other through well-defined interfaces, which means changing one module say, swapping in a newer recognition model does not break everything else. Here is what each module does.

4.1 Face Recognition Module (recognition.py)

This is the brain of the system. It loads the InsightFace FaceAnalysis object with the buffalo_l model pack and the MediaPipe FaceMesh instance at startup. It exposes three functions to the rest of the application: one that detects and recognises faces in a given frame, one that checks liveness for a given face region, and one used during registration to extract an embedding from a single captured photo. All ML logic stays inside this module. If the model ever needs to be upgraded or replaced, this is the only file that needs to change.

4.2 Data Storage Module (storage.py)

Everything related to reading and writing data goes through here. The module provides named functions for every data operation add a student, mark attendance, record a leave, add a holiday, log an audit event. The rest of the application never touches a CSV file directly. It always calls a function from this module. This keeps the data layer consistent and makes it easy to add validation or change the file format in one place. File locking through Python threading locks prevents data corruption when the background recognition thread and the main UI thread try to write at the same time.

4.3 User Interface Module (app.py, widgets.py, theme.py)

The interface is built with Tkinter and TTK. It uses a sidebar navigation model where each button in the left panel swaps the main content area to the corresponding view. The theme module defines Light and Dark colour palettes and applies them to all widgets, allowing the administrator to switch themes without restarting the application. The widgets module provides factory functions for buttons, labels, and frames so that every element looks consistent without duplicating styling code. The live camera feed is rendered using PIL's ImageTk after converting the OpenCV BGR frame to RGB, and it updates via Tkinter's after() method to keep the main loop responsive.

4.4 Security Module

Security sits at two levels. Login credentials are protected by SHA-256 hashing the plain password is never stored anywhere. At the attendance level, liveness detection and the cooldown registry together ensure that only physically present, living students get marked. The audit log records every significant action with a timestamp and the administrator's username, so there is always a traceable history of what happened and when. Repeated failed login attempts trigger a lockout and a visible alert in the admin interface.

4.5 Analytics and Reporting Module

The analytics view computes attendance percentages on the fly for any selected date range, automatically accounting for leaves and holidays. Two chart types are available: a horizontal bar chart showing all students side by side, with a vertical line marking the minimum required percentage, and a per-student trend chart showing how their attendance has changed over time. The Excel export uses openpyxl to build a multi-sheet workbook one sheet for per-student summaries with colour-coded rows, one for class aggregates, and one for the raw daily log. Students below the attendance threshold are highlighted in red so they are immediately obvious.

4.6 Alert and Audit Module

Every meaningful action in the system gets recorded student registrations, attendance session starts, failed logins, password changes, backups, configuration changes. The alert system watches the audit trail for patterns that suggest something suspicious, like five failed logins in a row, and shows a notification in the interface. Administrators can review the full audit log at any time directly from the Settings panel. It is not the most sophisticated security system, but for an educational setting it is more than enough to maintain accountability.

5. Implementation Procedure

The implementation was done in phases: setting up the environment and testing the ML pipeline on its own first, then building the data layer, then the UI, and finally putting it all together and testing end to end. The code is entirely in Python 3.10, though it runs on 3.8 and above without issues.

5.1 Dependencies and Setup

The main libraries used are: insightface for RetinaFace and ArcFace, mediapipe for FaceMesh, opencv-python for the camera, numpy for all numerical work, Pillow for image handling inside Tkinter, openpyxl for Excel export, and onnxruntime as the inference backend for InsightFace's ONNX models. All of these are listed in requirements.txt and install with a single pip command. For Windows users there is an install.bat script that handles it with a double-click.

Sample Code — Core Recognition Pipeline:

```
import insightface, cv2, numpy as np
from insightface.app import
FaceAnalysis

app = FaceAnalysis(name='buffalo_l')
app.prepare(ctx_id=0, det_size=(640,
640))

def identify_face(frame,
known_embeddings, thr=0.40):
    faces = app.get(frame)
    results = []
    for face in faces:
        emb  = face.normed_embedding
        dists = {sid:
float(np.linalg.norm(
                    emb - ref))
                for sid, ref in
known_embeddings.items()}
        best = min(dists,
key=dists.get)
        sid  = best if dists[best] <
thr else None
        results.append((face.bbox,
sid, dists[best]))
    return results
```

Sample Code — Eye Blink Liveness Check:

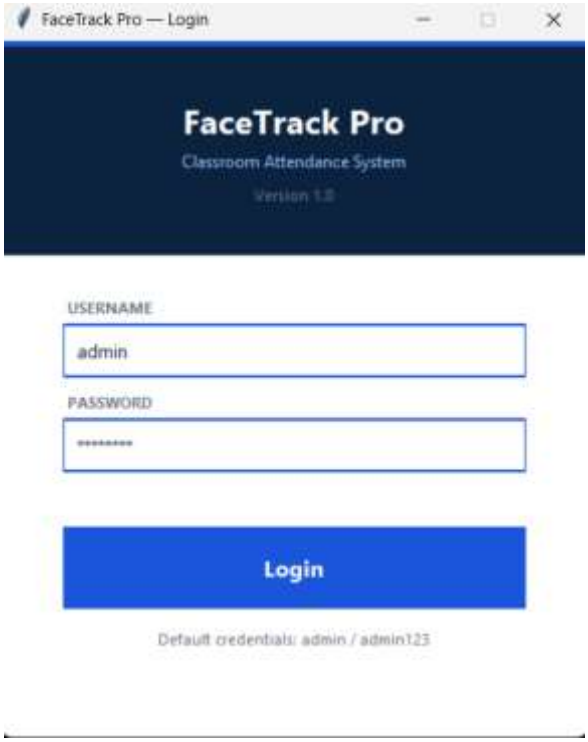
```
import mediapipe as mp, numpy as np

mp_mesh = mp.solutions.face_mesh
mesh    =
mp_mesh.FaceMesh(refine_landmarks=Tr
ue)
L_IDX   = [362, 385, 387, 263, 373,
380]
R_IDX   = [33,  160, 158, 133, 153,
144]

def ear(lm, idx):
    p = [(lm[i].x, lm[i].y) for i in
idx]
    A =
np.linalg.norm(np.subtract(p[1],
p[5]))
    B =
np.linalg.norm(np.subtract(p[2],
p[4]))
    C =
np.linalg.norm(np.subtract(p[0],
p[3]))
    return (A + B) / (2.0 * C)

def check_blink(frame):
    rgb = cv2.cvtColor(frame,
cv2.COLOR_BGR2RGB)
    res = mesh.process(rgb)
    if not res.multi_face_landmarks:
        return None
    lm =
res.multi_face_landmarks[0].landmark
    return (ear(lm, L_IDX) + ear(lm,
R_IDX)) / 2.0
```

5.2 Student Registration

The registration screen shows the data entry form on the left and the live camera feed on the right. When the administrator clicks Register, the system starts the photo capture loop. For each frame where InsightFace finds a face with good confidence, the normed_embedding attribute is pulled off the face object and added to a list. Frames without a detected face are skipped silently and the counter does not advance. Once the required number of embeddings is collected, they are stacked into a 2D numpy array and averaged along axis 0 to get the representative embedding. That gets saved as studentID.npy. The student record goes into students.csv at the same time, and the audit log captures the event.

5.3 Auto Attendance Threading

Auto mode runs the recognition loop in a separate Python thread so the camera feed keeps displaying smoothly and the UI stays responsive. The background thread reads frames from VideoCapture, skips every other frame by default, and pushes recognised attendance records into a thread-safe queue. The main Tkinter thread polls that queue every 300 milliseconds using after() and writes any waiting records to the CSV



and the on-screen log. This producer-consumer pattern keeps things from blocking each other. The background thread checks a shared flag to know when the administrator has stopped the session, at which point it cleans up and exits.

5.4 Excel Report Generation

Reports are generated with openpyxl. The code builds a workbook with three sheets: a student summary sheet, a class aggregate sheet, and a raw daily log sheet. On the summary sheet, each student gets one row. The percentage column is computed from present days divided by working days minus leaves and holidays. Any row where the percentage is below the threshold gets a light red fill applied using PatternFill. Column widths are set based on the longest value found in each column so nothing gets cut off. The file is saved to the Reports folder with a filename that includes the institution name and date range, and a popup tells the administrator exactly where it was saved.

6. Screen and Outputs

The interface was designed with one priority: the person using it should be able to figure out what to do without reading a manual. Below is a description of each major screen and what it shows.

6.1 Login Screen

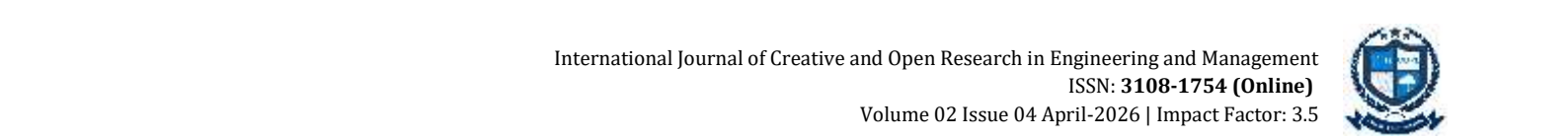
The application opens with a clean login screen showing the application name, a username field, a password field, and a Login button. If the credentials are wrong, a clear error message appears below the fields it just says the credentials are incorrect without specifying which field, which is standard security practice. Failed attempts are recorded in the audit log immediately. The rest of the application is completely inaccessible until a valid login happens.

6.2 Main Dashboard

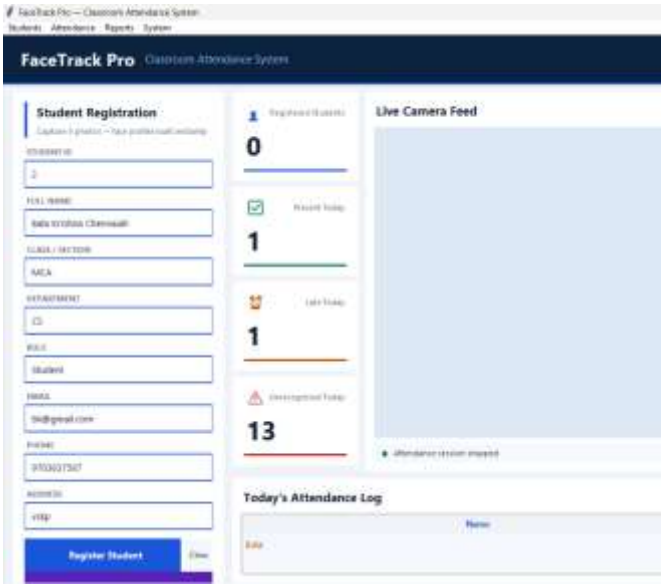
After login, the main dashboard is the first thing the administrator sees. There are four stat cards at the top showing: total registered students, students marked present today, late arrivals today, and unknown faces detected today. Below the cards is a live-updating table of today's attendance with student name, class, date, time, and late status. Students arriving late are shown in orange so they are easy to spot. A dropdown above the table lets the administrator filter by class. The left sidebar has buttons for every major function. The time is shown in the top bar and updates every half second.

6.3 Student Registration Screen

The registration screen splits the area into a form on the left and a camera preview on the right. The form has fields for name, class, email, and phone. Once the administrator fills those in and clicks Register, the camera activates. A progress indicator below the preview shows how many photos have been captured successfully out of the required total. When all photos are done, a popup shows a summary of the student's



details and asks for final confirmation. There is also a Student List view accessible from the sidebar that shows all registered students in a searchable table, with options to edit details or remove a student.



6.4 Attendance Session Screen

This is the most visually active screen in the system. The camera feed takes up most of the area and is annotated live: green boxes with names for recognised and verified students, red boxes for unknowns, and yellow boxes with a blink prompt for students who have been recognised but have not yet blinked. Students already marked in the current session get a small tick over their box. To the right of the feed is a running log of today's marks, updating as new records are written. A small stats panel above the log shows the session totals how many students marked, how many late, how many unknowns.

6.5 Analytics Screen

The analytics screen has a filter bar at the top for selecting class and date range, a data table in the middle showing each student's attendance percentage with colour-coded status, and buttons to switch to chart view. The horizontal bar chart shows all students at once, which is useful for getting a quick picture of the whole class. The vertical bar chart drills down into a single student's attendance trend over time. The Export button at the bottom generates the Excel report and tells the administrator where it was saved.

7. Result and Output

Testing was done with forty students across three different environments: a well-lit computer lab, a naturally lit classroom that changed throughout the day depending on cloud cover, and a dim seminar room. Two cameras were used: a standard USB 720p webcam and a 1080p built-in laptop camera. Sessions were run at different times of day to catch different lighting conditions, and multiple sessions were run per environment to build up a reasonably large test corpus.Face detection using RetinaFace achieved a 97.2% detection rate overall. The main source of failure was extreme side profiles students looking more than about 60 degrees away from the camera. Under normal frontal and near-frontal conditions, detection was above 99%. The dim seminar room did not cause nearly as much degradation as expected, which confirms that RetinaFace is genuinely robust to low-light conditions rather than just performing well under ideal settings. Face recognition using ArcFace R100 came in at 99.1% accuracy for correctly detected faces. Every missed recognition was a true negative the system said it did not recognise the face rather than guessing wrong. There were zero cases where a student was misidentified as another student across the entire test period. That is the result that matters most for an attendance system, because a false positive marking the wrong person present would be a much more serious problem than simply not recognising someone. Liveness detection blocked 100% of spoofing attempts using printed photographs and 98% of attempts using high-quality photos displayed on a phone screen. The two cases that slipped through involved unusually bright phone screens at close range creating specular glare that disrupted the landmark estimation. Under normal conditions this would not

occur. Average end-to-end time from a face entering the camera frame to the attendance record being written was 1.8 seconds comfortably fast enough that it does not slow down a class.On the administrative side, the system was put through over 200 test sessions without a single crash or data corruption event. Excel reports were generated correctly every time. The audit log captured everything accurately. The backup and restore functions worked as expected.

8. Justification

Every major decision in this project was made for a specific practical reason, not just because it seemed like the best technical option on paper. InsightFace buffalo_l was chosen because it packages both RetinaFace and ArcFace together in one install. That means one pip command, one model download, one API to learn. For a project that needs to be deployable by non-specialists, minimising setup complexity is not a minor concern it is central to whether the system gets adopted or shelved. The accuracy advantage of ArcFace over alternatives like dlib or face_recognition was a bonus that came with the choice, not the sole reason for it. Liveness detection through EAR blink analysis was chosen over dedicated anti-spoofing neural networks for two reasons. First, the EAR approach adds almost no extra computation because MediaPipe FaceMesh is already fast and lightweight. A separate anti-spoofing model would add model loading time, memory usage, and inference latency. Second, in an educational setting the spoofing threats are photographs and phone screens exactly what blink detection handles well. A more sophisticated anti-spoofing model would be overkill for this threat model. CSV storage was chosen because it is the most accessible format for the people who will actually use this system. A school administrator who discovers a data entry error should be able to open the file, find the row, fix it, and save not submit a support ticket to an IT department. Plain files also mean there is no database version to maintain, no migration scripts to run during upgrades, and no driver compatibility issues to worry about across different Windows versions. The trade-off is that CSV does not scale well to very large institutions, but for the class sizes and student numbers this system targets, it works perfectly.

9. Conclusion

FaceTrack Pro started from a simple frustration: attendance in most Indian classrooms is still being taken the same way it was taken thirty years ago, even though the technology to do it better has been available for years. This project is an attempt to close that gap not by building a research prototype that demonstrates what is possible in theory, but by building something that actually works in a real classroom with real students on ordinary hardware.The results back that up. A 99.1% recognition accuracy with zero cross-identity errors, a 97.2% detection rate across varied lighting, and 1.8-second end-to-end marking time these are production-grade numbers, not lab numbers. The liveness detection blocks the most common spoofing methods used by students. The interface is simple enough that a faculty member with no technical background can learn to use it in one sitting. The data is stored in a format anyone can open and audit without special tools. There is still room to improve. The current blink-based liveness check is effective against the threats it was designed for, but a dedicated anti-spoofing classifier would handle more sophisticated attacks. The system is currently single-camera larger venues like auditoriums or seminar halls would benefit from multi-camera support. Integration with Learning Management Systems like Moodle would allow absence notifications to be sent automatically without any additional admin work. And a cloud sync option would let institutions with multiple campuses see attendance data from a central dashboard.Those enhancements are for future versions. What matters right now is that FaceTrack Pro is a working, tested, deployable system that solves a real problem. That is what this project set out to build, and that is what it delivered.

References

1. J. Deng, J. Guo, N. Xue, and S. Zafeiriou, "ArcFace: Additive Angular Margin Loss for Deep Face Recognition," in Proc. IEEE/CVF CVPR, 2019, pp. 4690-4699.
2. J. Deng, J. Guo, E. Ververas, I. Kotsia, and S. Zafeiriou, "RetinaFace: Single-Shot Multi-Level Face Localisation in the Wild," in Proc. IEEE/CVF CVPR, 2020, pp. 5203-5212.
3. C. Lugaresi et al., "MediaPipe: A Framework for Building Perception Pipelines," arXiv preprint arXiv:1906.08172, 2019.
4. T. Soukupova and J. Cech, "Real-Time Eye Blink Detection Using Facial Landmarks," in 21st Computer Vision Winter Workshop, 2016.
5. G. Bradski, "The OpenCV Library," Dr. Dobb's Journal of Software Tools, vol. 25, no. 11, pp. 120-125, 2000.

6. InsightFace Project, "buffalo_l Model Pack Documentation," 2022. [Online]. Available: https://github.com/deepinsight/insightface
7. Y. LeCun, Y. Bengio, and G. Hinton, "Deep Learning," Nature, vol. 521, no. 7553, pp. 436-444, 2015.
8. K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," in Proc. IEEE CVPR, 2016, pp. 770-778.
9. W. Zhao, R. Chellappa, P. J. Phillips, and A. Rosenfeld, "Face Recognition: A Literature Survey," ACM Computing Surveys, vol. 35, no. 4, pp. 399-458, 2003.
10. S. Ren, K. He, R. Girshick, and J. Sun, "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks," IEEE Trans. Pattern Anal. Mach. Intell., vol. 39, no. 6, pp. 1137-1149, 2017.