Automated Attendance System Using Face Recognition

Name: N. Divyagnan Reddy
Roll No: 2201130

Supervisor: Dr. Upasana Talukdar

Department of Computer Science and Engineering
Indian Institute of Information Technology Guwahati

This project is submitted for the degree of Bachelor of Technology

IIIT Guwahati April 2025

Acknowledgment

I would like to express my heartfelt gratitude to Dr. Upasana Talukdar, my project supervisor, for her continuous guidance, encouragement, and technical support throughout the course of this project. Her insightful feedback and mentorship were crucial in shaping this work.

I would also like to sincerely thank my teammate, Vadisetti Pranay Satvik Reddy, for his contributions, support, and well-coordinated efforts. His work on the backend and integration modules played an instrumental role in bringing this dream project to life.

A special thanks goes out to the developers and contributors of the open-source libraries used in this project, especially the creators of the **InsightFace** framework and pretrained models, whose work enabled significant advancements in the face recognition module.

This project would not have been possible without the collaboration, guidance, and tools provided by the broader research and open-source communities.

Contents

Abstract

This project involves the development of the core **facial recognition system** and student-registration platform for an automated attendance tracking solution. My responsibilities include designing and implementing the **deep learning**-based face recognition pipeline, which includes face detection and identity verification using state-of-the-art models. The pipeline is optimized to handle variations in lighting, angle, and facial features. In addition, I developed the student registration frontend, where students can upload their face data for enrollment. A lightweight pre-verification model ensures that submitted images meet quality requirements before registration. These components form the backbone of the system's recognition capabilities, enabling accurate and user-friendly attendance monitoring.

1 Introduction

1.1 Project Overview

This project is part of a larger facial recognition-based attendance system aimed at automating student attendance tracking using **deep learning**. My contribution centers around developing the face recognition models, the complete recognition pipeline, and the student registration front-end. These components serve as the backbone for accurate and efficient identification of students from video feeds captured during class hours.

1.2 Purpose

The primary objective of my work is to ensure accurate face recognition through a reliable **deep learning** pipeline and to provide a seamless way for students to register their facial data. This data forms the basis for training and fine-tuning the face recognition model, enabling the system to identify individuals in classroom environments with minimal false positives or negatives.

1.3 Models Used

- **ArcFace (ResNet100 Backbone):** A powerful face recognition model fine-tuned on the collected student dataset using an **ArcFace** head for optimized embedding learning.
- **RetinaFace (from InsightFace):** Employed for accurate and efficient face detection, capable of handling faces with varying poses and lighting.
- **face-api.js:** Used in the front-end to provide lightweight, real-time feedback during the registration process, helping users capture clear and front-facing images.

1.4 Data Collection Process

To collect student facial data, I deployed the registration front-end on **Vercel** and exposed the backend API through a **Cloudflare Tunnel**, making it accessible for students to register remotely. Around 30 students participated by submitting facial images through the platform. A lightweight verification model ensured that captures met quality standards before accepting them. The collected data was then used to fine-tune the **ArcFace** model, improving its accuracy in a classroom setting. Final testing was conducted on recorded classroom videos to validate real-world performance.

2 Software Requirements Specification (SRS)

The following subsections outline the functional and non-functional requirements of the system, providing a clear specification of its capabilities and constraints.

2.1 Functional Requirements

Table 1: Functional Requirements

ID	Description	Status
F1	The system shall provide a student registration interface to upload annotated face images.	✓
F2	The system shall verify the quality (lighting, occlusion, angles) of uploaded images in real-time.	✓
F3	The system shall perform data augmentation on submitted student photos.	✓
F4	The system shall train a fine-tuned **ArcFace** model (**ResNet100** + **ArcFace** loss) on the dataset.	✓
F5	The model shall produce 512-dimensional embeddings for each student face.	✓
F6	The system shall export a classification-based face recognition model to a backend-accessible API.	✓
F7	The pipeline shall process student face frames and return predicted identity & confidence.	✓
F8	The system shall generate a log of verified identities to be consumed by the backend attendance system.	✓

2.2 Non-Functional Requirements

Table 2: Non-Functional Requirements

ID	Description	Status
N1	The system shall detect spoof attempts (e.g., phones, paper faces) using a **YOLOv8n-face** model.	×
N2	The system shall process face recognition from video streams within 15 seconds of real-time feed.	×
N3	The system shall support recognition from up to 10 simultaneous camera feeds.	×
N4	The fine-tuned model shall achieve ¿95% classification accuracy on the student dataset.	✓
N5	The lightweight image verifier model shall run under 50ms per image.	✓
N6	The registration process shall run asynchronously, not blocking server-side performance.	✓
N7	The trained model shall be tailored to one class (up to 30 students), not scalable across campus.	✓

3 Modules

3.1 Student Registration Module

The student registration module is a critical entry point to the **face recognition pipeline**. It allows students to enroll themselves into the system by submitting high-quality, diverse facial images directly through a web-based front-end. This module was built using **React** and deployed via **Vercel** to ensure ease of access across devices. It utilizes **face-api.js**—a lightweight, browser-compatible library—for real-time face detection and orientation estimation in the front-end.

Registration Procedure
The system guides students through a structured, interactive face capture process:

- **Front-Facing Capture:** The student is asked to look directly into the camera. Once the face is detected with a confidence score of $\geq 60\%$, the system captures 10 high-quality images.
- **Left Profile Capture:** After the front-facing step, the student is asked to turn their head to the left. The **face-api.js** model estimates the orientation, and only when the detected yaw angle falls within a predefined left-facing threshold, 10 side-profile images are captured.
- **Right Profile Capture:** The final stage asks the student to turn their head to the right, where another 10 images are taken under the same orientation-based condition.

In total, each student contributes 30 facial images (10 per orientation), offering a well-rounded dataset for fine-tuning the recognition model.

Image Quality Controls
To ensure usable data:

- Images are captured only if lighting conditions are adequate, judged by pixel brightness thresholds.
- Face detection confidence must exceed 60%, balancing the need to detect sideways profiles while filtering out poor detections.
- The system also ensures that only one face is present in the frame to prevent accidental multi-person registration.

Screenshots of the registration process (including different head orientations and UI prompts) will be included in this report to visually validate the pipeline.

Data Submission
After successfully completing the image capture:

- Students fill in personal details (e.g., name, ID, contact).
- All metadata and image data are then sent to the backend (exposed via **Cloudflare Tunnel**) for storage, verification, and eventual use in model fine-tuning.

Security and Verification Challenges
A key challenge is the possibility of impersonation—a student registering another person's face. To address this:

- A verified boolean flag is attached to each registration entry in the backend.
- This variable is manually toggled by an admin, who reviews the submissions visually before marking them as valid.
- This also helps ensure that only one individual appears in the frame throughout the process.

As of now, the system has successfully registered 27 students, providing a dataset of approximately 810 facial images. While the registration flow captures images from multiple head orientations, the lighting conditions were mostly consistent due to environmental constraints. To address this, further data augmentation—including simulated lighting variations, occlusions, and minor facial changes—is required to improve the model's robustness and generalization to real-world classroom scenarios.

3.2 Data Preprocessing and Augmentation

To prepare the collected images for training, a structured preprocessing pipeline was implemented to ensure consistency and compatibility with the **deep learning** model architecture.

Preprocessing Steps
Each image underwent the following preprocessing steps:

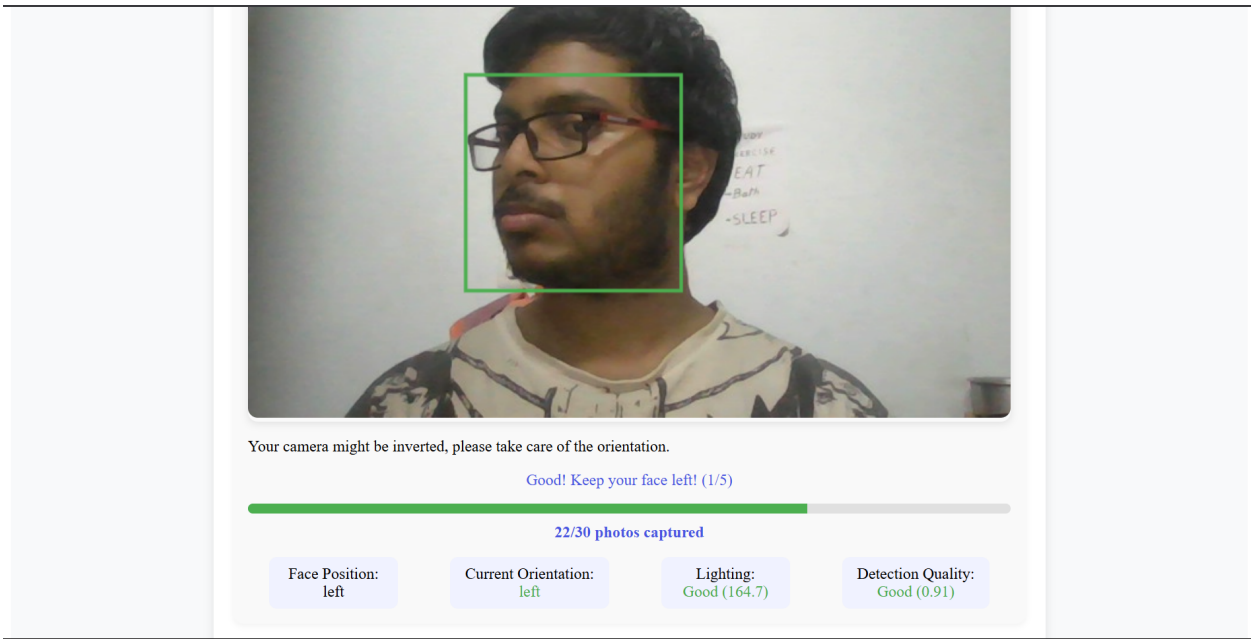

Figure 1: Student Registration Interface

- **Face Detection & Alignment:** Faces were detected using **RetinaFace** from the **InsightFace** library. The detector outputs five facial landmarks, which were used to align the face by centering the nose and maintaining upright orientation.
- **Bounding Box Relaxation:** Instead of tightly cropping the face, a relaxed bounding box was applied to include some contextual area around the face, improving recognition of side-profile or partially occluded images.
- **Resizing:** The aligned face regions were resized to 112×112 pixels, matching the input format expected by the **ResNet100 ArcFace** model from **InsightFace**.

This ensured that the data fed into the model closely matched the conditions under which the pretrained model was originally trained, improving fine-tuning results.

Augmentation Strategy

To increase data diversity and enhance model generalization, heavy augmentation was applied using the **Albumentations** Python library. Augmentations simulated real-world conditions such as changes in brightness, noise, blur, occlusion, and perspective.

- **Training Set:** Each original image was augmented 5 times, producing a richly varied dataset. In addition, mixups and other combinatorial transformations were applied across the dataset to further increase variability.
- **Validation Set:** Each image was augmented 3 times with moderate transformations to test the model's adaptability while maintaining reasonable consistency.
- **Test Set:** Each image was augmented once to simulate real-world inference with minimal artificial variation.

This augmentation pipeline expanded the original dataset of ~ 810 images into over 8000 augmented images across training, validation, and test splits:

- Training Set: ~ 6000 images
- Validation Set: ~ 1500 images

- Test Set: ~ 700 images

This significantly enriched dataset allowed the model to learn more robust features and better generalize to real-world classroom scenarios, despite the limited number of initially collected student samples.

3.3 Face Recognition Model

The core of the face recognition system is built upon the **ResNet100** backbone trained with **ArcFace** loss, chosen for its proven performance in high-accuracy face verification tasks. The model is derived from the **InsightFace** framework, which consistently ranks among the top in face recognition benchmarks like **LFW**, **MegaFace**, and **IJB-C**.

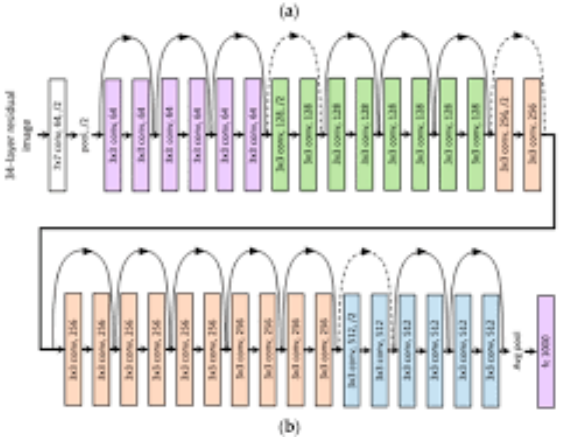

Figure 2: ResNet100 Architecture with ArcFace Head

Why ResNet100 + ArcFace?

- **High Capacity and Depth: ResNet100** provides a deep residual architecture capable of capturing fine-grained facial details across identities, expressions, and orientations. It strikes a strong balance between accuracy and computational feasibility when run on a **GPU**.
- **ArcFace Loss for Enhanced Separability:** Unlike traditional softmax or triplet loss, **ArcFace** loss (Additive Angular Margin Loss) enforces a more discriminative embedding space. It does this by:
  - Normalizing both feature vectors and class weights to lie on a hypersphere.
  - Introducing an angular margin between classes to increase inter-class variance and reduce intra-class variance.

  The **ArcFace** loss is formulated as:

$$L = -\frac{1}{N}\sum_{i=1}^{N}\log\frac{e^{s(\cos(\theta_{y_i}+m))}}{e^{s(\cos(\theta_{y_i}+m))}+\sum_{j=1,j\neq y_i}^{n}e^{s\cos\theta_j}} \tag{1}$$

  This encourages the model to not just correctly classify faces, but to maximize angular separation between different identities—an essential quality in surveillance and attendance systems.

3.3.1 Experiments

Experimentation and Model Prototyping
In this section, we present two key experimental approaches explored during the early development stages of the recognition pipeline. Both methods utilize face embeddings generated by a pre-trained **ArcFace** model, which outputs 512-dimensional vectors representing each face image. These embeddings form the foundation for downstream classification and verification tasks.

Dataset Overview

The experiments were conducted on a curated dataset of 810 original images of 27 unique students, which were augmented to approximately 7000 images using techniques such as:

- Rotation and mirroring
- Brightness and contrast changes
- Simulated occlusion (partial blurring or masking)
- Noise injection and JPEG compression artifacts

The augmentation aimed to simulate real-world conditions (e.g., camera glare, motion blur, or hairstyle changes) and improve generalization.

1. K-Nearest Neighbors (KNN) Voting-based Recognition

Motivation

The first experiment was designed to explore a simple but effective face verification method. Instead of training a traditional classifier, we attempted a nearest neighbor search in embedding space. Given a query embedding (from the test face), we compute the **cosine similarity** with all stored embeddings from the dataset and retrieve the top K most similar embeddings. The final identity is assigned via majority voting among these K neighbors.

Implementation

- *Embedding Generator:* **ArcFace** (**IResNet100**) pre-trained model.
- *Distance Metric:* **Cosine similarity**.
- *Search Strategy:* Brute-force O(N) comparison with each embedding.
- *Voting Strategy:* K = 5, majority class selection.
- *Data:* Full 7000 augmented image set for reference embeddings.

Results and Analysis

- *Accuracy:* 1.0 (100%) on the test dataset.
- *Time Complexity:* O(N) per prediction. As N grows (e.g., more students or daily images), performance degrades.
- *Latency:* High — not suitable for real-time tracking or in-classroom usage without vector index optimization (e.g., **FAISS**).
- *Strengths:* Extremely accurate when embeddings are discriminative; works well even in noisy conditions.
- *Limitations:* Scalability is a bottleneck. Every query image needs comparison with thousands of embeddings, which is computationally expensive and memory-heavy.

2. Random Forest Classifier on Face Embeddings

Motivation

The second experiment evaluated the feasibility of using classical machine learning classifiers on face embeddings, specifically the **Random Forest** classifier. Unlike **KNN**, **Random Forest** learns decision rules over the embedding space and provides near-instant predictions after training.

Implementation

- *Model:* **Random Forest** with 100 decision trees.

- *Features:* 512-d embeddings per face.
- *Labels:* Student roll numbers (mapped to 27 unique classes).
- *Training Set:* ∼7000 images.
- *Testing:* 30% stratified split from the same set (to simulate a student who is already registered).

Results and Analysis

- *Training Accuracy:* Nearly perfect (∼1.0).
- *Testing Accuracy:* ∼0.30 (30%).
- *Training Time:* Extremely fast (¡ 10 seconds) on **CPU**, even with 7000 samples and 512 features.
- *Inference Time:* Low (sub-millisecond per image).
- *Strengths:* Simplicity, interpretability, and fast training cycles.
- *Limitations:*
  - Poor generalization in high-dimensional space without handcrafted feature selection.
  - Sensitive to noise and misalignment in face images.
  - Trees overfit to specific embedding patterns that might not generalize across augmentations.

Why Performance is Poor

While **ArcFace** embeddings are high-quality, they are not linearly separable enough for a decision-tree-based method to model class boundaries effectively. The randomness in tree splitting combined with high feature dimensionality and intra-class variance from augmentations results in reduced generalization capacity. Additionally, **Random Forest** lacks the capability to learn nuanced inter-class margins, which is crucial in face recognition.

Conclusion of Experiments

These two experiments were crucial in evaluating the feasibility and trade-offs of different recognition strategies. While the **KNN** approach demonstrated perfect accuracy, it proved impractical for real-time attendance tracking due to its computational cost. The **Random Forest** model, on the other hand, was extremely efficient but unable to provide acceptable accuracy levels.

This justified the shift towards a fine-tuned **ArcFace** classifier model, using the same 512-d embeddings but backed by a deep neural network trained with additive margin loss, offering both high accuracy and efficient inference — a balance that neither of the above baselines could achieve.

Comparative Summary of Models

Table 3: Comparative Summary of Models

Metric / Model	**KNN + Voting**	**Random Forest**	**Fine-Tuned ArcFace**
Accuracy	1.0	∼0.30	1.0
Training Time	None	¡10 seconds	∼25 minutes (**GPU**)
Inference Time	Slow (O(N))	Very Fast (ms)	Fast (1-2 ms/image)
Complexity	High at runtime	Low	Moderate
Scalability	Poor	Good	Excellent
Augmentation Used	✓	✓	✓
Memory Requirement	High	Low	Medium
Overfitting Risk	None	High	Low (Augmented)
Real-Time Suitability	No (needs **FAISS**)	No (low acc)	Yes

3.3.2 Current Implementation

Model Architecture & Fine-Tuning
The training process began by:

- Initializing an empty **ResNet100** block using the **InsightFace** PyTorch implementation.
- Loading pretrained weights from arcface_resnet100torch.pth, providing a solid feature extraction foundation.
- Adding an **ArcFace** classification head, which maps the 512-dimensional embedding vector to the number of student identities (30 classes in this case).
- Fine-tuning the entire network (both the backbone and the classification head) using the augmented dataset and **ArcFace** loss.

Fine-tuning allowed the pretrained model to specialize in recognizing the specific faces in the dataset, adapting to variations introduced by augmentation while maintaining the discriminative power learned from large-scale datasets.

Results
Training and validation results:

```
Epoch 40/40 Results:
2025-04-08 16:03:47,942 - INFO - Train Loss: 0.1513 | Acc: 0.9986
2025-04-08 16:03:47,942 - INFO - Val Loss: 0.0226 | Acc: 1.0000
2025-04-08 16:03:47,942 - INFO - Val Precision: 1.0000 | Val Recall: 1.0000 | Val F1: 1.0000
```

Test results:

```
Final Test Results:
2025-04-08 16:03:52     Loss: 0.0353 | Acc: 1.0000
2025-04-08 16:03:52,843 - INFO - Precision: 1.0000 | Recall: 1.0000 | F1: 1.0000
```

3.4 Recognition Pipeline

The recognition pipeline functions as a **producer-consumer system**, where video streams are processed frame-by-frame to detect, track, and recognize faces in real time.

Producer: Face Detection & Preprocessing
Each incoming frame is processed using **RetinaFace**, a highly accurate single-stage face detector optimized for real-time applications. It detects all visible faces in a frame and produces:

- The bounding box of each detected face.
- A cropped and preprocessed image of the face (resized, aligned, normalized).
- Optionally, face embeddings using a lightweight model for fast updates.

These outputs are passed on to the **FaceTracker**, a custom tracker that handles identity preservation across multiple frames using a combination of **IoU** (Intersection-over-Union) and **cosine similarity** of facial features.

FaceTracker: Identity Preservation Across Frames
The custom **FaceTracker** maintains a set of active "tracks"—each corresponding to a unique person moving across the frame. Each track stores:

- Bounding box coordinates,
- Last frame seen,
- Facial embedding features,
- Skip count for missed detections.

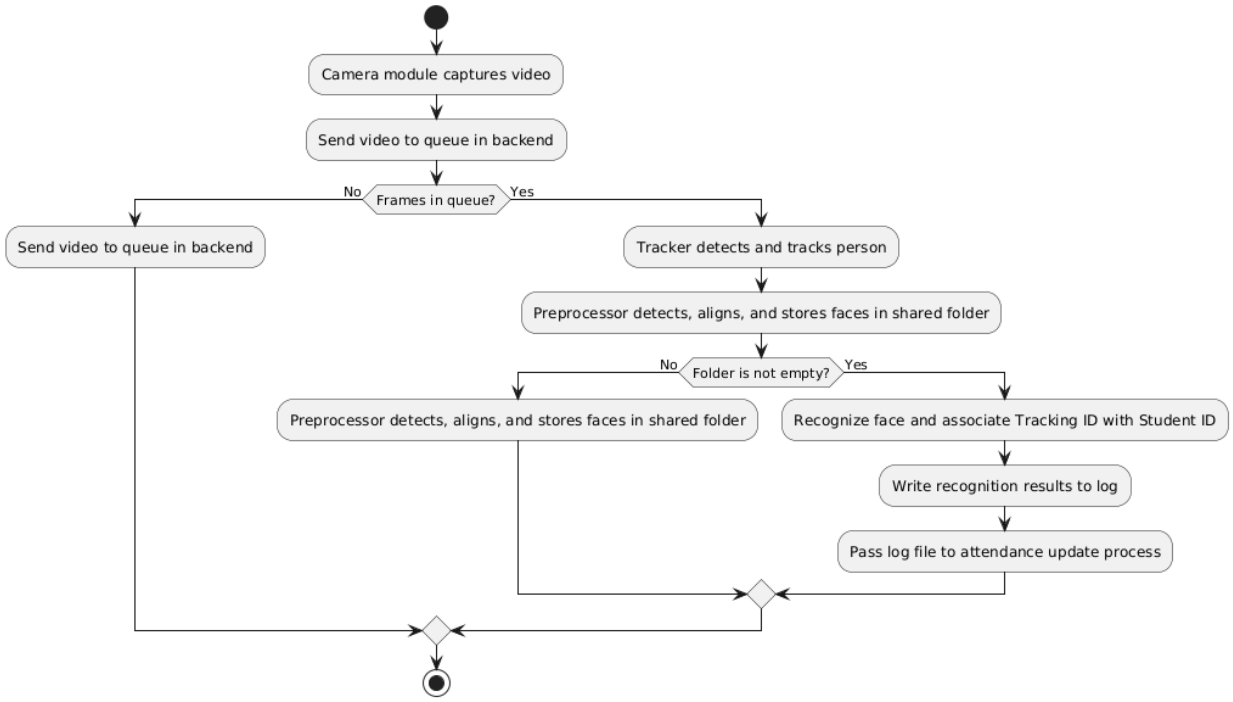

Figure 3: Face Recognition Pipeline Overview

The tracker operates in multiple stages:

- **IoU-Based Matching:** For each new face, compare its bounding box with existing tracks using **IoU**. If above a threshold, the face is matched to the corresponding track.
- **Embedding-Based Matching:** If **IoU** fails, match using **cosine similarity** between face embeddings.
- **Track Management:** Unmatched tracks are eventually removed after a number of skipped frames. New tracks are created for unmatched faces.

This hybrid tracking approach is lightweight yet robust, handling occlusions and fast movements with graceful recovery.

Consumer: Face Recognition & ID Assignment

Once tracks are stabilized by the tracker, the consumer side kicks in, handling recognition and attendance marking.

- Each preprocessed face image is passed through the fine-tuned **ArcFace** model to generate embeddings.
- The embeddings are classified using the **ArcFace** head to get the most likely student identity.
- Since predictions can occasionally be incorrect (due to blur, angle, lighting, etc.), the system maintains a buffer of predictions per track. For each track, it records a history of student IDs predicted across frames.
- **Majority Voting:** Once a student leaves the frame or is consistently tracked for several frames, the most frequently predicted ID (majority vote) is assigned to the track. This corrects one-off mispredictions and increases overall robustness.

In-Room Entry Logic

Students are counted only when they enter a room. The assumption is that classroom entry is always from the right or left side of the camera's field of view. This spatial logic helps reduce false positives from passersby or background faces. Upon confirmed entry:

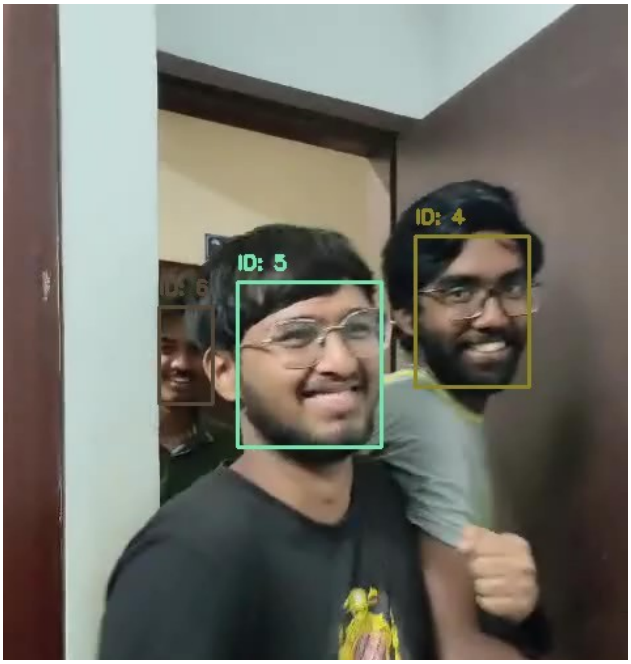

Figure 4: Face Tracking in Action

- The track's ID (after majority voting) is mapped to a roll number.
- This roll number is added to the attendance log.

Attendance Logging
Once a student's identity is finalized via tracking and majority voting:

- A log entry is generated including:
  - Timestamp,
  - Student roll number,
  - Room/class ID,
  - Video source ID,
  - Confidence (optional).
- This entry is sent to the backend server for real-time attendance recording.

Example Attendance Log
Below is an example attendance log generated by the system for a session involving 6 students, showcasing the output format and content:

```
Track ID,Roll number,Highest Conf Prediction,Highest Confidence,timestamp,came_in
track_0,2201157,2201157,1.000000,2025-04-07 23:34:23.987,True
track_1,2201134,2201134,1.000000,2025-04-07 23:34:24.037,True
track_2,Unknown,Unknown,0.732957,2025-04-07 23:34:27.037,False
track_3,2201130,2201130,1.000000,2025-04-07 23:34:29.037,True
track_4,2201221,2201221,1.000000,2025-04-07 23:34:30.537,True
track_5,2201213,2201213,1.000000,2025-04-07 23:34:31.537,True
track_6,2201051,2201051,1.000000,2025-04-07 23:34:32.537,False
```

This log demonstrates the system's ability to accurately identify students with high confidence (1.0) when they enter the room ('came$_in = True$'), $while also flagging unrecognized individuals (e.g., 'track_2') with lower confidence and n$

4 Drawbacks and Limitations

Unmet or Partially Met Requirements

Table 4: Unmet or Partially Met Requirements

Type	Requirement	Status
Non-Functional	Spoof Detection	×
Non-Functional	15s Real-Time Recognition	×
Non-Functional	10 Live Feed Processing	×

Design & Scalability Constraints

- **Model Rigidity:** The **ArcFace** model is fine-tuned per class, making it unsuitable for dynamically adding new students. Retraining is required to accommodate new identities.
- **Limited Scope – One Model per Class:** Given batch size (∼30 students/class), it's feasible to maintain one model per class. Training a universal model for the entire campus may reduce performance and increase inference time drastically.
- **Live Recognition Accuracy:** While test accuracy is 100%, real-world video recognition achieves ∼80% accuracy due to:
  - Motion blur, occlusions
  - Lighting variation
  - Frame selection inconsistency

5 Future Work and Extensions

1. Spoof Detection for Enhanced Security
To ensure the integrity and reliability of the face recognition system, future versions should integrate a dedicated **spoof detection** module. The current system lacks explicit mechanisms to prevent presentation attacks such as printed photos or video replays. **Spoof detection** would add a crucial layer of defense against such vulnerabilities.

Potential directions include:

- Depth estimation-based methods to differentiate 3D live faces from 2D spoof artifacts.
- Temporal-based approaches like blink detection, lip movement, and micro-expression monitoring.
- Lightweight **CNN** models trained on spoof datasets like **CASIA-SURF**, **CelebA-Spoof**, or **MSU-MFSD**, ensuring real-time performance on edge devices.
- **YOLO**-based spoof detection integrated into the current **YOLO** detection pipeline for efficient joint processing.

2. Incremental Learning for Continuous Improvement
The face recognition model can be extended to support **incremental learning** or continual learning to adapt to:

- Gradual appearance changes in students (e.g., facial hair, hairstyles, accessories).
- New student registrations without full retraining of the base model.
- Drift in data distribution due to changes in camera angles, lighting, or classroom setups.

This can be achieved using:

- Replay-based methods like **Elastic Weight Consolidation (EWC)** or **Memory Aware Synapses (MAS)** to prevent forgetting past faces.
- Prototype-based classifiers that adapt to new identities with minimal data.
- Unsupervised incremental updates triggered during verification failures or low-confidence predictions.

These additions would make the system more resilient and adaptive in real-world educational environments, reducing manual intervention over time.

6 Conclusion

This project marks a significant step toward a fully automated, camera-based attendance system designed with reliability, scalability, and real-world constraints in mind. Starting from the student registration module, a lightweight **face-api.js** model ensured that data collection was feasible even on front-end platforms, capturing multi-angle facial images only under proper lighting and confident detection. With over 30 students registered via a frontend deployed on **Vercel** and backend tunneled through **Cloudflare**, a diverse preliminary dataset was collected.

To enhance generalizability, the raw images were preprocessed using **InsightFace**'s **RetinaFace** for alignment and resized to 112x112 resolution before undergoing extensive augmentation using the **Albumentations** library. This led to the creation of a robust dataset with nearly 7000 images across training, validation, and test sets (5766 + 800 + 408).

For the core recognition model, **InsightFace**'s **IResNet100** was selected for its strong performance on face recognition tasks. It was finetuned with an **ArcFace** head using additive angular margin loss, which improves inter-class separability and intra-class compactness, a key factor in improving real-world recognition accuracy.

Finally, a real-time recognition pipeline was engineered, involving a **producer-consumer structure**. **RetinaFace** served as the producer, detecting and preprocessing faces, while a custom-built **FaceTracker** maintained consistent identities using **IoU** and feature similarity. The **ArcFace**-based recognizer then consumed these faces, and majority voting across frame predictions ensured stability against occasional model errors. The system was sensitive to directionality—only recognizing entries when students passed from the side into view—mirroring real-world behavior. Recognized identities were logged and forwarded to the backend for attendance marking.

Overall, this comprehensive pipeline—from registration to recognition—lays down a scalable and reliable framework for institutional attendance systems, balancing computational efficiency with real-world robustness.

7 References

- InsightFace: 2D and 3D Face Analysis Project
  GitHub Repository: `https://github.com/deepinsight/insightface`
- Additive Margin Softmax for Face Verification
  Feng Wang, Weiyang Liu, Haijun Liu, Jian Cheng
  arXiv preprint arXiv:1801.05599, 2018.
  Paper: `https://arxiv.org/abs/1801.05599`
- ArcFace: Additive Angular Margin Loss for Deep Face Recognition
  Jiankang Deng, Jia Guo, Jing Yang, Niannan Xue, Irene Kotsia, Stefanos Zafeiriou
  arXiv preprint arXiv:1801.07698, 2018.
  Paper: `https://arxiv.org/abs/1801.07698`
- InsightFace PyTorch Implementation
  GitHub Repository: `https://github.com/TreB1eN/InsightFace_Pytorch`

- Albumentations: Fast and Flexible Image Augmentations
  GitHub Repository: `https://github.com/albumentations-team/albumentations`
- face-api.js: JavaScript Face Recognition API
  GitHub Repository: `https://github.com/justadudewhohacks/face-api.js`