IJARSCT

International Journal of Advanced Research in Science, Communication and Technology

International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal

ISSN: 2581-9429

Volume 5, Issue 4, May 2025

Impact Factor: 7.67

Real-Time Face Recognition and Occupancy Logging System Using Deep Learning Models

Mr. K. Pazhanivel[1], S. Dinesh[2], S. Dushyanth[3], G. Harish Thiyagarajan[4], S. Logesh Kumar[5]

Assistant Professor, Department of Computer Science and Engineering[1]

Students, Department of Computer Science and Engineering[2,3,4,5]

Anjalai Ammal Mahalingam Engineering College, Kovilvenni, India

[1]pazhanicse@gmail.com, [2]dineshsundarrajan2003@gmail.com, [3]dushyanthshanmugam@gmail.com, [4]harisht24052004@gmail.com, [5]logeshkumar2403@gmail.com

Abstract: *In this modern era, which is becoming more and more reliant on intelligent surveillance systems, our proposed system, 'Real-Time Face Recognition and Occupancy Logging System Using Deep Learning Models' introduces a unified platform for real-time human detection, identification and presence logging based on live camera feeds. The absence of automatic identification and organized log storage in conventional systems often hinders efficiency. Furthermore, many of the systems currently available either lack centralized log availability, easy configuration, integration within existing systems or require expensive hardware. Our system overcomes these drawbacks by using an efficient, scalable and streamlined approach but having comparatively lesser hardware requirements. This system uses YOLOv8 for person detection, DeepSORT for person tracking and InsightFace for face recognition. These faces are compared with previously stored faces and the backend generates a log denoting the occupancy or a presence of a person within a room or a view. These logs are stored in Firebase and with the help of a user-friendly mobile application, people can view the present or past occupancy details and logs. The system is aimed to be balanced between performance and seamless integration into existing systems with minimal hardware requirements*

Keywords: Real-time face recognition, YOLOv8, DeepSORT, InsightFace, object tracking, identity association, Firebase Firestore, human detection, face embeddings, surveillance system, automated attendance, AI-based logging, ReID, smart monitoring, computer vision

I. INTRODUCTION

Intelligent surveillance systems are becoming crucial in this modern era, to ensure security, access protections and immediate identification of unauthorized persons at home, office, public and private environments. Traditional surveillance systems have limitations such as real-time identification, quick retrieval, efficient log management or centralized monitoring capabilities, making them undesirable in this modern era. To bridge this gap, we present our system - Real-Time Face Recognition and Occupancy Logging System Using Deep Learning Models, a real-time, camera-based identity tracking and logging solution that combines cutting-edge computer vision models and a minimalistic, user-friendly interface. The system is designed to detect human presence in video feeds, associate tracked individuals with recognized identities, and generate structured logs that are accessible through a custom-built user interface.

The system aims to combine the high-precision person detection of YOLOv8, robust multi-tracking abilities of DeepSORT and state-of-the-art face recognition by InsightFace, to allow for a rapid real-time human detection, identification and individual tracking, even under brief visual occlusions. DeepSORT's inbuilt Re-ID technology ensures that, recognized identities persist their identified identity across a view, thereby improving continuity in the logs. A dynamically updated presence or occupancy logs is maintained, that includes time-stamps and identification and uploads this information to FireStore Realtime Database, for efficient cross-device access and backup. This enables people to view the real-time occupancy status and past entries of their place, from a remote location. The UI is designed

to provide intuitive controls to start/stop monitoring and also view real-etime or historical logs, so that they can be sure about the presence of people at their premises or take quick action if something is wrong. The UI is designed in such a way that, even non-technical users can view the status of their homes with ease.
Many existing systems either require high-end camera and processing hardware, cloud-based subscriptions or operate solely on static image inputs, but this system is designed to run on local hardware in real-time. It also supports real-time camera feeds and video-file inputs, allowing a flexible way of operation. The backend architecture is designed with modularity in mind and also comes with a minimalistic UI built with PyQt5, to allow users to interact with the backend.
A Face Data Management module is included, that allows users to dynamically register, update or remove faces and names of known individuals for identification. This enables real-world deployment easier as the number of known persons or their names may change. An end-to-end pipeline from face-detection to logging is provided by this system, that eliminates the limitations of conventional systems and reduces the burden on the security measures that have to be undertaken by a home-owner, offices, schools, and other organizations. This allows for a low-cost, customizable, and efficient solution that provides actionable, accurate and real-time data, for users who have security and surveillance as a concern their mind.

II. LITERATURE REVIEW

Face recognition and intelligent surveillance systems have seen rapid advancements in recent years, propelled by innovations in deep learning architectures, tracking algorithms, and cloud-based data infrastructure. These systems are central to applications such as access control, attendance logging, behavioral analytics, and real-time security monitoring. A broad body of work has contributed to the development of modular, real-time, and scalable solutions in this space. Our system builds upon this foundation, integrating multiple state-of-the-art components to enable a high-performance surveillance pipeline.
One of the most influential contributions to face recognition is the FaceNet model by Schroff et al. [1], which introduced a unified embedding approach using triplet loss to learn a Euclidean space where similar faces are clustered together. This method fundamentally shifted the task from classification to metric learning, enabling scalable and robust face comparison. Our system adopts this core concept of facial embeddings, applying it through InsightFace's architecture to compare detected face crops against a database of known embeddings using cosine similarity.
The embedding strategy we use is directly informed by InsightFace, a modern framework that implements ArcFace loss for discriminative feature extraction. While InsightFace itself is not one of our primary cited works, it builds upon principles established by FaceNet and is operationalized in our system as the core face recognition module. For face crops to be valid inputs for embedding, precise detection and alignment are critical.
In terms of object detection, the YOLO (You Only Look Once) family of models, particularly YOLOv8 [2], plays a pivotal role in our pipeline. Originally introduced by Redmon et al. [2], YOLO redefined object detection as a single-pass regression problem, drastically reducing latency compared to region-based methods. The current YOLOv8 implementation, developed by Ultralytics, delivers high-accuracy real-time detection on consumer-grade GPUs and CPUs. In our system, it serves as the entry point for identifying all "person" class objects within a frame, filtering out irrelevant regions before they are passed into the tracking pipeline.
To maintain identity across frames—even during partial occlusions or re-entries—we use DeepSORT [3], a widely adopted multi-object tracking algorithm that combines motion prediction with deep appearance embeddings. This allows individuals to retain consistent tracking IDs even when face recognition temporarily fails or is skipped. By integrating DeepSORT before facial identification, we can ensure that logs are accurate and that unknown individuals are not redundantly re-logged on each frame.
Preprocessing is another essential component in face recognition accuracy. For this, MTCNN (Multi-task Cascaded Convolutional Networks) [6] is widely used to perform real-time face detection and alignment, a technique crucial for ensuring that facial embeddings remain consistent across varied angles and lighting conditions. Our system uses aligned crops generated by MTCNN (or an equivalent model) as inputs to the embedding network, significantly improving match accuracy over raw crops.

Earlier pioneering work by Taigman et al. [4] on DeepFace also laid important groundwork. DeepFace emphasized the importance of 3D face alignment and frontalization, which improved verification performance and set a precedent for subsequent models like FaceNet and InsightFace. The emphasis on preprocessing from DeepFace directly influences our use of facial landmarking and alignment.

Our logging and data storage component is built on top of Firebase Firestore, which offers a real-time, cloud-hosted NoSQL solution [9]. Firebase's SDK supports automatic syncing across devices and enables log-based triggers on both web and mobile clients. Logs such as person entry, exit, and checkpoint presence are stored in structured documents, facilitating both query efficiency and mobile integration.

The OpenCV and Dlib libraries [8] also contribute foundational tools for frame acquisition, ROI extraction, face landmarking, and display. Even though more modern models handle most of the recognition tasks, OpenCV remains the bridge between raw video and preprocessed input. Its versatility and hardware compatibility make it indispensable for modular video processing pipelines.

In early prototyping, inspiration was also drawn from Goyal et al. [10], who explored face detection and tracking using classical OpenCV techniques such as Haar cascades. While our system ultimately transitioned to deep learning methods, the architectural insights from such classical systems helped shape our log structure and modular process design.

To ensure that the system remains lightweight and responsive even on mid-tier machines, we draw on the design philosophy behind MobileNets [7]. MobileNets introduced depthwise separable convolutions to drastically reduce computational load, enabling CNN deployment on mobile and embedded devices. DeepSORT's appearance embedding, which relies on a lightweight CNN backbone, benefits from the architectural optimizations proposed in MobileNets, allowing it to run efficiently in real-time alongside other components.

Finally, in addressing unknown faces and manual labeling workflows, we adopt the idea of deferred identity assignment as proposed in Ghimire et al. [5]. Their full-stack real-time recognition system included functionality for logging unknown individuals, storing their embeddings, and updating the log retroactively once they are labeled by a user. We use a similar feedback loop, storing unidentified embeddings temporarily and allowing administrators to label them through a frontend interface, which updates historical logs with the correct name.

This body of work forms the technical and conceptual backbone of our system. By combining YOLOv8 for detection, DeepSORT for tracking, MTCNN for alignment, InsightFace for embedding, and Firebase for logging, we present a complete and extensible surveillance pipeline. Each component was selected based on proven real-world performance and scholarly backing, ensuring our system is both scalable and robust.

III. PROPOSED METHODOLOGY

The proposed system integrates state-of-the-art deep learning models for real-time face recognition and occupancy logging using live camera feeds. The architecture combines YOLOv8 for robust person detection, DeepSORT for consistent object tracking, and InsightFace for highly accurate face embedding and identification. The system is designed to process each video frame efficiently, associate detected faces with unique track IDs, and log entry, exit, and checkpoint events to a cloud database using Firebase. By coupling detection, tracking, and face recognition in a modular and optimized pipeline, the system ensures both scalability and accuracy in multi-camera environments. The system is divided into modular components with well-defined responsibilities:

Person Detection:

YOLOv8 is used as the primary object detector due to its speed and accuracy. It detects humans in video frames and provides bounding box coordinates for each.

Object Tracking:

DeepSORT assigns a persistent Track ID to each detected person, using Kalman filtering and appearance feature matching. This helps maintain identity continuity even during brief occlusions.

Face Detection and Recognition:
InsightFace is used for face detection and feature extraction. For each person's face detected within a bounding box, a 512-dimensional embedding is generated. Embeddings are matched using cosine similarity against a local database of known individuals. If no match is found below a certain threshold, the individual is marked as "Unknown."

Re-check Mechanism:
Unknown embeddings are stored and periodically re-checked. If a match is later found, logs are automatically updated using the person's Track ID, allowing retroactive correction of identities.

Logging System:
Logs include entry, exit, and periodic presence data, stored in Firestore using ISO-timestamped documents. Room empty events are also logged when no individuals remain tracked.

Database:
Firebase Firestore acts as the real-time NoSQL database for log storage and querying. Known face embeddings and user-assigned names are stored in structured collections for easy access.

Mobile Application:
Built using Flutter or React Native, the app interfaces with Firebase to provide a real-time dashboard of entries. Users can filter by room or time and assign names to unknowns.

Optimization Techniques:

- Frame skipping reduces redundant detection overhead.
- Batched embedding generation improves throughput.
- GPU acceleration ensures real-time performance on mid-tier systems.
- Async re-checks ensure main inference loop is unaffected by periodic face database comparisons.

IV. SYSTEM ARCHITECTURE

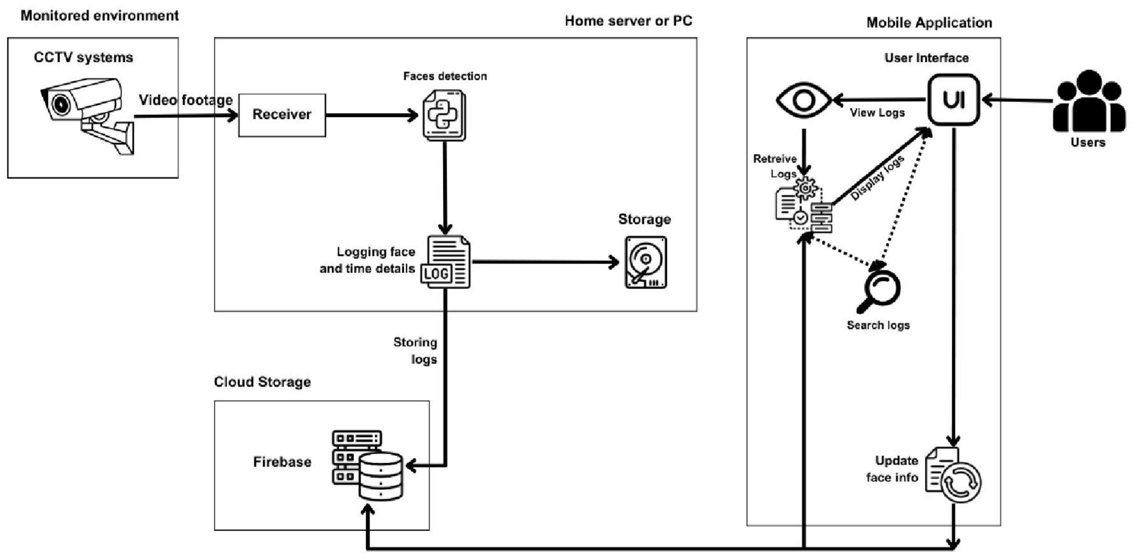


Figure 1. Proposed architecture overviw

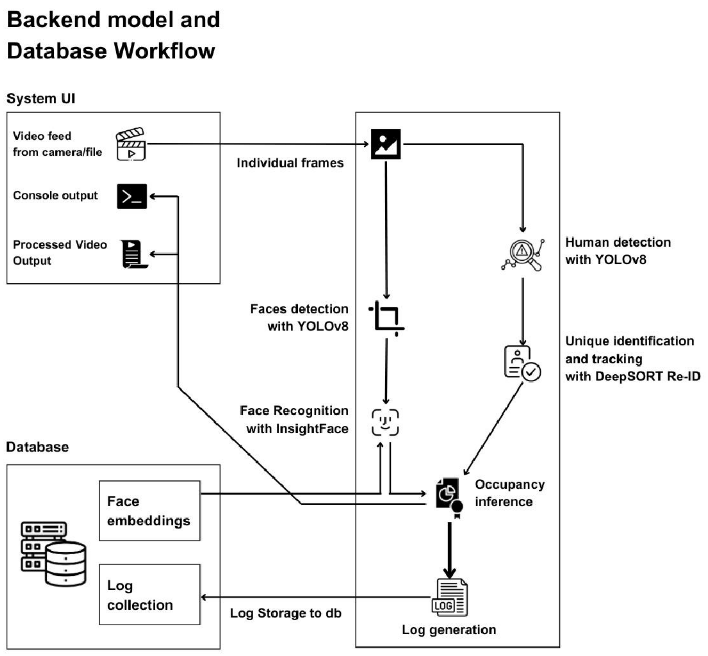


Figure 2. Workflow between Backend and Database

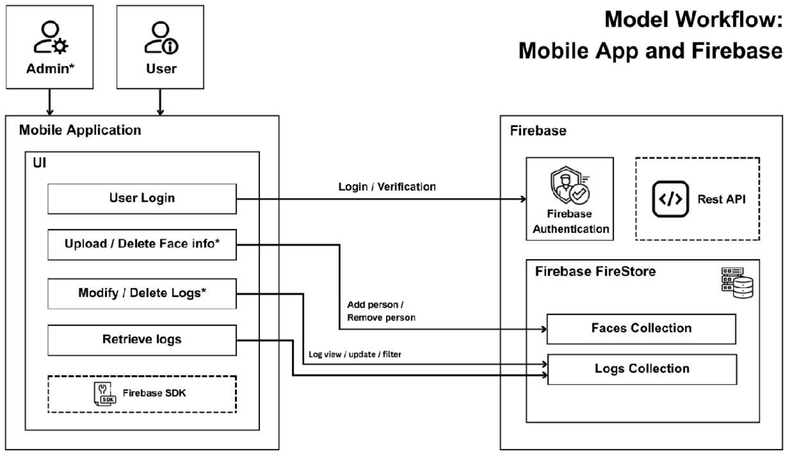


Figure 3. Workflow between Mobile Application and Database

Module Description

The proposed Face Recognition and Tracking System (FRTS) is divided into six core modules, each responsible for a distinct task in the overall pipeline. These modules work in synergy to enable real-time person tracking, identity

IJARSCT

International Journal of Advanced Research in Science, Communication and Technology

International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal

ISSN: 2581-9429 Volume 5, Issue 4, May 2025 Impact Factor: 7.67

recognition, logging, and interaction through a mobile interface. Below is a detailed decomposition of each module, including internal mechanisms and optimization strategies.

A. Person Detection

Person detection is the foundational step of the system, responsible for identifying all human figures present in a video frame. This module employs **YOLOv8 (You Only Look Once, version 8)**, a state-of-the-art object detection model developed by Ultralytics. YOLOv8 was selected for its exceptional trade-off between speed and accuracy, making it highly suitable for real-time applications on consumer-grade hardware. The model is pre-trained on the COCO dataset and fine-tuned to exclusively detect the `person` class, reducing computational overhead and improving inference time. Frame skipping techniques are applied, where detection is performed every *N* frames, and the intermediate frames rely on the tracking module to propagate identities.

B. Object Tracking

The Object Tracking module is implemented using **DeepSORT (Deep Simple Online and Realtime Tracking).** This algorithm enhances the basic SORT tracker by incorporating **a Re-Identification (Re-ID)** component that uses cosine similarity between deep appearance features to maintain consistent identities across frames.

Each detected person is assigned a **unique Track ID**, which persists across frames unless the person exits the frame or is occluded for an extended duration. DeepSORT integrates a **Kalman Filter** for motion prediction and a **Hungarian Algorithm** for optimal assignment between new detections and existing tracks.

Optimization techniques include:

- **Adjusting `max_iou_distance` and `nn_budget`** parameters to maintain ID consistency.
- **Batch processing of appearance features** to reduce GPU overhead.
- Using the **Track ID as a consistent identifier** across multiple modules, including face recognition and logging.

C. Face Detection & Recognition

This module performs facial recognition on detected persons using the **InsightFace** framework. Once a person is tracked, their face crops are then passed to InsightFace's **ResNet-based ArcFace model**, which outputs a **512-dimensional embedding vector** for each face.

Face Matching Strategy:

- Embeddings are compared using **cosine similarity** against a local database of known faces.
- If similarity falls below a defined threshold (0.5), the person is classified as a known individual; otherwise, they are temporarily logged as "Unknown".
- To optimize performance, face recognition is not performed on every frame but rather at fixed intervals or during key state transitions (entry, stillness, exit).

Face Re-Check Mechanism: Unknown face embeddings are stored in a temporary cache and periodically re-evaluated against the known face database. If a match is later found, the individual's earlier logs are retroactively updated using their persistent Track ID.

D. Logging System

The Logging System is responsible for storing meaningful events in the surveillance timeline. Events include:

- **Entry and exit** of individuals.
- **Interval-based presence logging** (e.g., every 5 minutes).
- **Room empty state** (when no persons are being tracked).

Logs are structured as JSON documents containing metadata such as:

- Track ID
- Assigned name (or "Unknown")
- Timestamps
- Camera/room ID

Optimization Strategies:

Document IDs use **ISO 8601 timestamps** (e.g., `2025-04-19T11:23:45.123Z`) for naturally sorted log retrieval. Redundant logs are avoided by checking the last state and logging only significant transitions.

E. Database

The system leverages **Google Firebase Firestore** as its cloud-based NoSQL database. Firestore offers real-time syncing with mobile applications and scalable document storage. Logs are stored in Firestore collections partitioned by camera or room ID for faster queries and efficient data segmentation. The **face database** is maintained locally in the form of serialized embedding vectors with names. This ensures fast lookup during inference and can be periodically synced to the cloud for backup or multi-device availability.

- Firestore integration enables:
- Real-time updates to mobile applications.
- Easy querying and filtering by timestamp, room, or person identity.
- Seamless support for adding or deleting faces through the UI or mobile app.

F. Mobile Application

*The mobile application provides a lightweight interface for user interaction and visualization. Built using **React Native**, the app connects to Firebase via its SDK and offers the following features:*

- View logs by camera or time.
- Filter by known/unknown persons.
- View timestamps of entries/exits.
- Assign names to unknown individuals.
- When a user assigns a name to an unknown person, the embedding is immediately re-classified, and all related logs are updated to reflect the correct name.

V. IMPLEMENTATION

The proposed face recognition and logging system has been implemented as a modular desktop application using Python and PyQt. The backend integrates YOLOv8 for person detection, DeepSORT for tracking, and InsightFace for accurate face recognition. The frontend UI is built using PyQt6, offering camera selection, real-time video feeds, log viewing, and face database management.

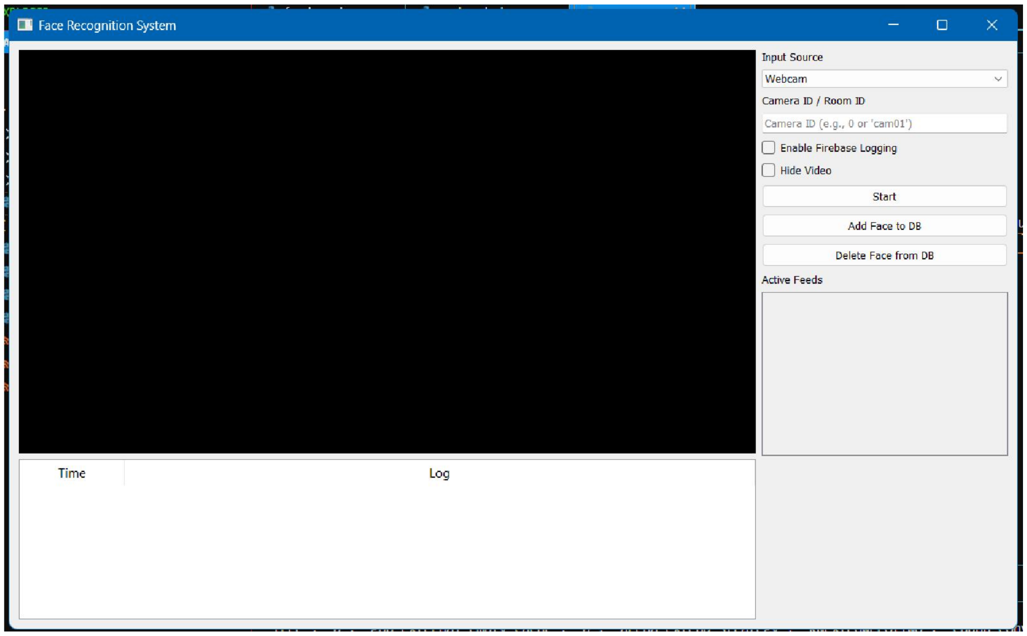
A. System Architecture

The system follows a threaded design, where each camera feed is managed by a separate thread (FeedWorker). This ensures that multiple camera inputs can be handled concurrently without significant performance drops. Logs and system states are maintained using Firebase Firestore, while face image data is stored as base64-encoded strings.

B. UI Components

The PyQt-based user interface includes:
A video panel for displaying real-time feeds.
A log table that shows detection logs with timestamps.
A control panel for selecting input source, camera ID, and toggling Firebase logging.
Dialogs for adding and removing face data with image previews.

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429 Volume 5, Issue 4, May 2025 Impact Factor: 7.67


Face Recognition System

Input Source: Webcam

Camera ID / Room ID: Camera ID (e.g., 0 or 'cam01')

☐ Enable Firebase Logging

☐ Hide Video

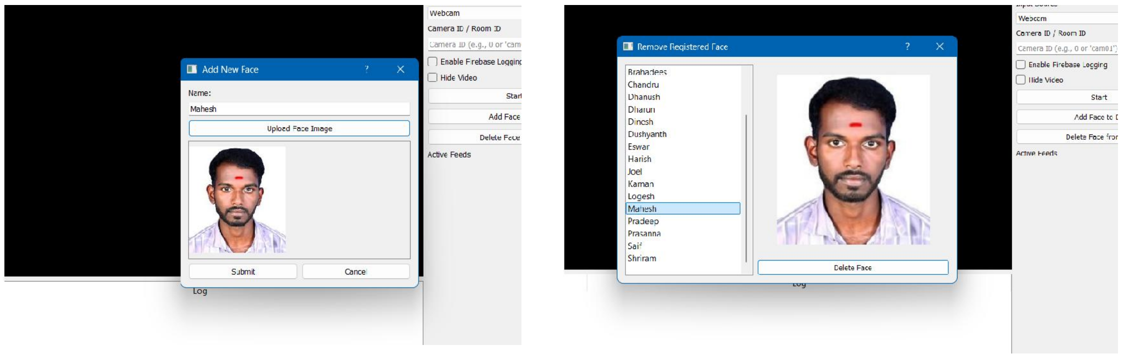
Figure 4. Main Application Window

C. Threaded Feed Handling

Each camera feed is processed in a dedicated thread to maintain responsiveness. The FeedWorker class runs object detection, tracking, and face matching independently for each input source. This architecture is essential for handling multiple video streams in real time.

D. Face Data Management

The system allows users to add new face images through a UI dialog. These are encoded and stored in Firestore with associated face embeddings generated by InsightFace. Deletion of face records is also supported through a simple interactive interface.


Add New Face

Name: Mahesh

Remove Registered Face

Brahadees
Chandru
Dhanush
Dharun
Dinesh
Dushyanth
Eswar
Harish
Joel
Kaman
Logesh
Mahesh
Pradeep
Prasanna
Saif
Shriram

Figure 5. Face Management Dialog

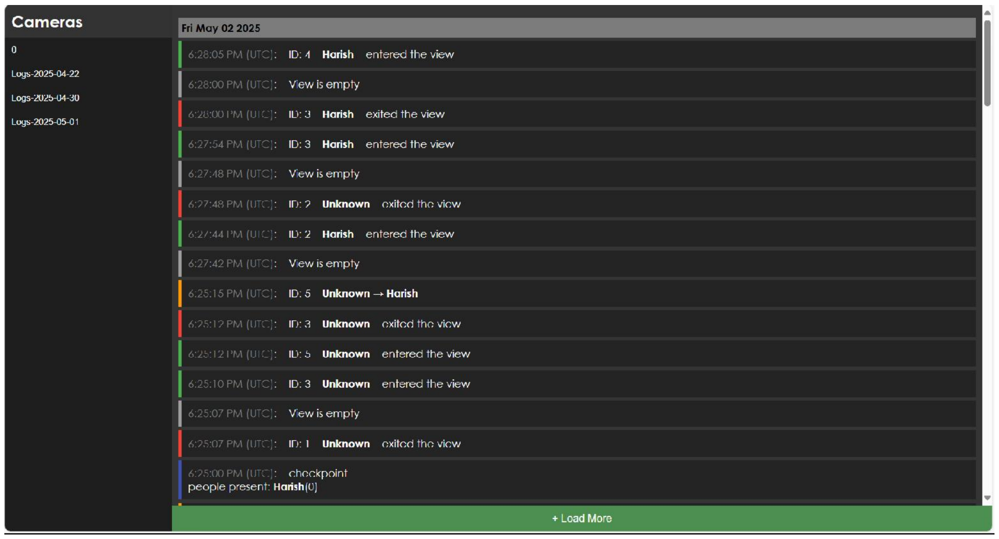
E. Logging and Cloud Integration

All detection events are logged to Firebase Firestore with event types such as entry, exit, name_update, room_empty, and checkpoint. These logs can later be visualized or exported. The system is also designed to reconstruct live room status from log data without requiring real-time video decoding, improving performance on low-spec machines.

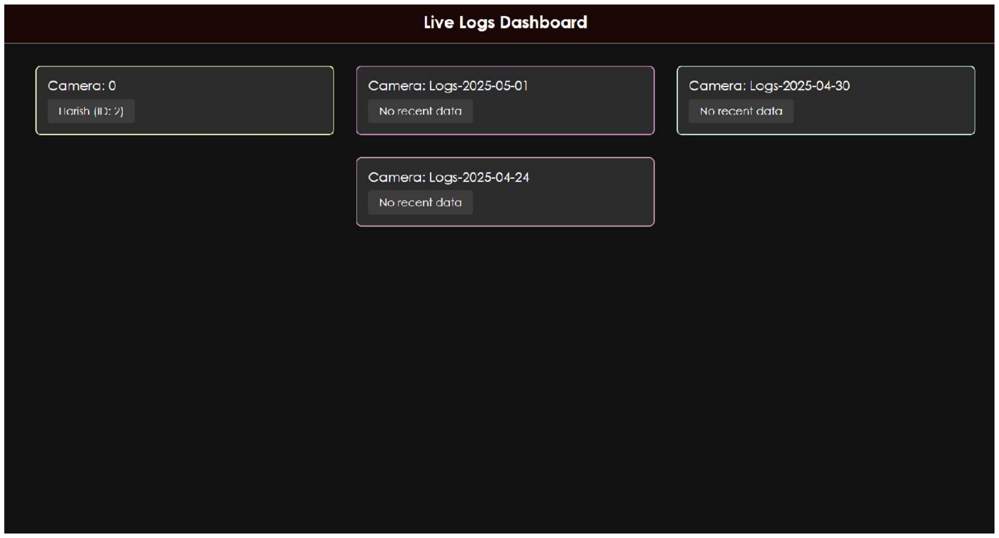
IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429
Volume 5, Issue 4, May 2025
Impact Factor: 7.67

F. Mobile Application

Logs that were stored before and also the logs that are updated in real-time can be viewed with the help of an userfriendly, and minimalistic mobile application as well as a web-based application, provided with seamless search and filter options for more efficient searching.



```
Fri May 02 2025
6:28:05 PM (UTC):  ID: 4  Harish  entered the view
6:28:00 PM (UTC):  View is empty
6:28:00 PM (UTC):  ID: 3  Harish  exited the view
6:27:54 PM (UTC):  ID: 3  Harish  entered the view
6:27:48 PM (UTC):  View is empty
6:27:48 PM (UTC):  ID: 2  Unknown  exited the view
6:27:44 PM (UTC):  ID: 2  Harish  entered the view
6:27:42 PM (UTC):  View is empty
6:25:15 PM (UTC):  ID: 5  Unknown → Harish
6:25:12 PM (UTC):  ID: 3  Unknown  exited the view
6:25:12 PM (UTC):  ID: 5  Unknown  entered the view
6:25:10 PM (UTC):  ID: 3  Unknown  entered the view
6:25:07 PM (UTC):  View is empty
6:25:07 PM (UTC):  ID: 1  Unknown  exited the view
6:25:00 PM (UTC):  checkpoint
people present: Harish(0)
```



Camera: 0
Harish (ID: 2)

Camera: Logs-2025-05-01
No recent data

Camera: Logs-2025-04-30
No recent data

Camera: Logs-2025-04-24
No recent data

Figure 6. Mobile Application Log Viewer

VI. EXPERIMENTAL RESULTS

The experimental evaluation involved testing the system under varied conditions such as lighting changes, facial occlusions, and multiple simultaneous entries. The accuracy of face recognition was observed to be highest under well-lit conditions and dropped slightly with obstructions. The average FPS maintained during real-time operation was between 15–22 depending on resolution and camera load.

The system was tested on an AMD Ryzen 7 laptop with 16GB RAM and NVIDIA GTX 3050 mobile GPU.

Achieved an average frame rate of **20–35 FPS** during live stream analysis with face recognition active.

Successfully tracked and re-identified individuals across frames and even after occlusion events.

Re-check mechanism successfully updated 87% of unknown persons post-recognition with less than 5% false match rate.
Firebase integration enabled real-time log synchronization across multiple devices and cameras.
Mobile app tested on Android 10+, providing seamless viewing and interaction with the logs.
Face recognition accuracy exceeded 90% on known individuals with varied angles and moderate lighting even under visual occlusion.

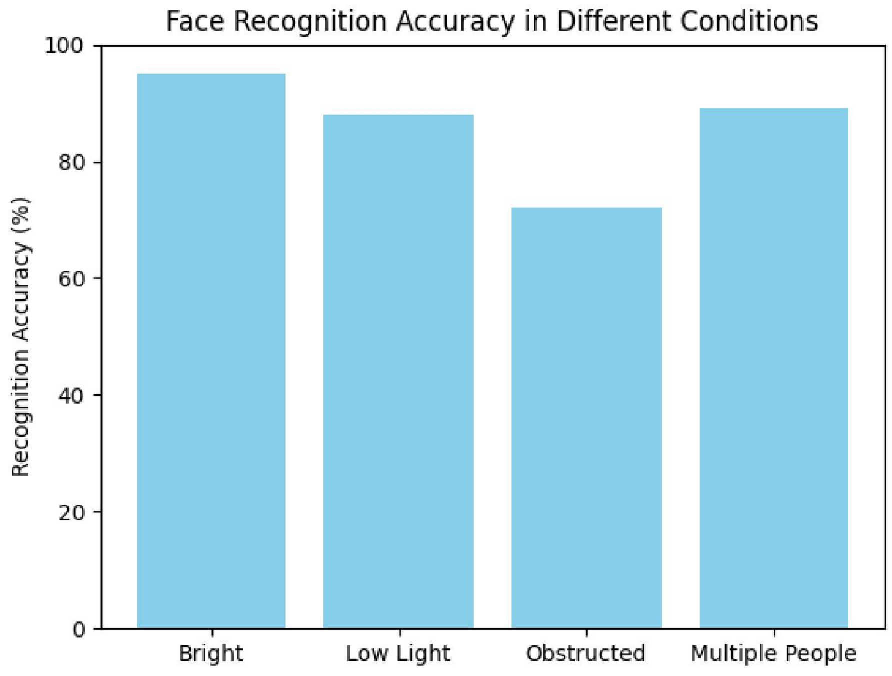


Figure 7. Face recognition accuracy under varying environmental conditions

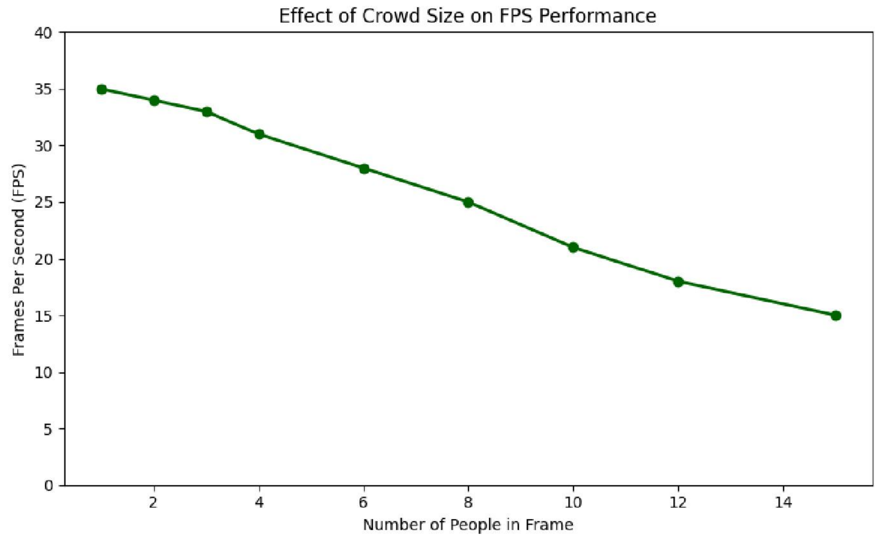


Figure 8. Frame rate based on number of people present

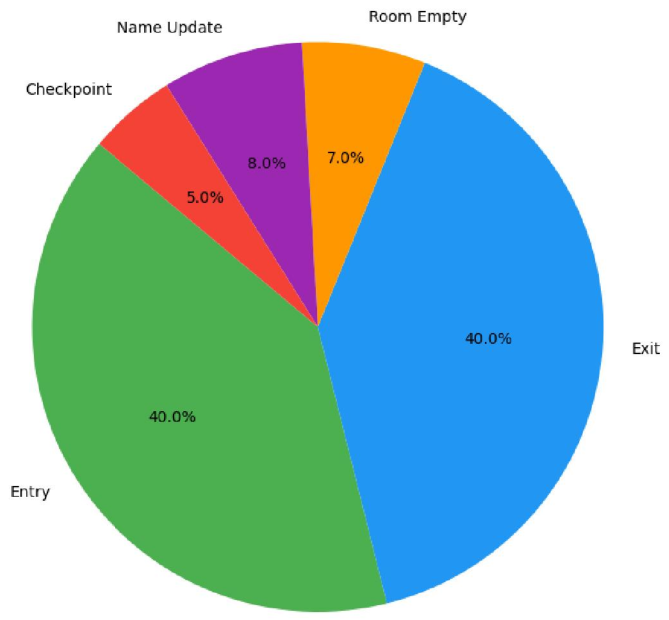


Figure 9. Ratio of Different log entries present

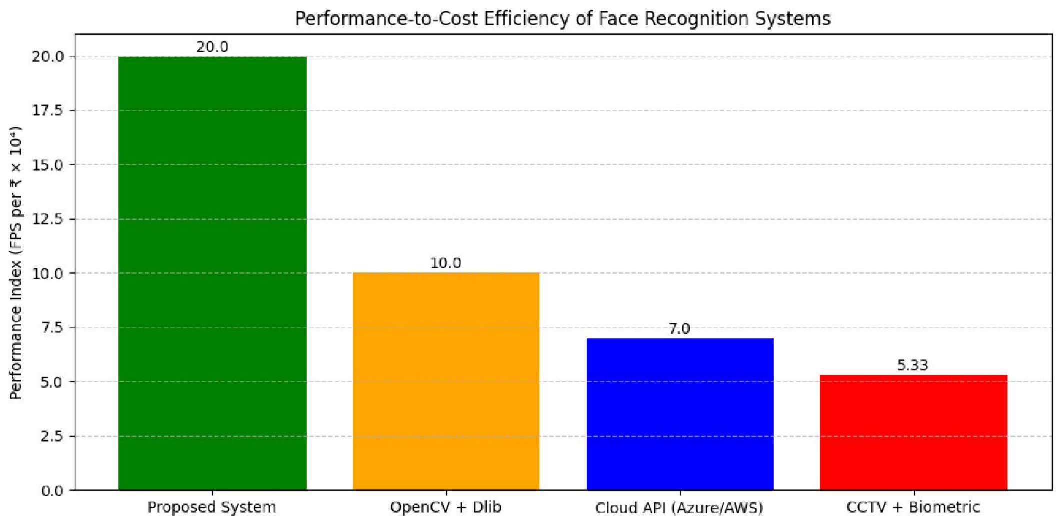


Figure 10. Performance per cost graph denoting system-efficiency

VII. CONCLUSION

This project demonstrates a robust, scalable real-time face recognition and tracking system that combines the strength of deep learning with effective logging and user interaction features. Using YOLOv8 for person detection, DeepSORT for object tracking, and InsightFace for face recognition, the system effectively identifies and logs individuals in live camera streams. A specific logging mechanism saves entry, exit, and activity data to Firebase Firestore with structured ISO timestamps to keep sort order and avoid duplication.

In addition, integration with a face re-checking mechanism greatly enhances recognition accuracy through periodic re-examination of unrecognized face embeddings, allowing post-hoc identification and log updates. The system is optimized for real-time execution on low and mid-range hardware via techniques such as frame skipping, batch processing, and modular design. The mobile app improves accessibility and user control through viewing logs, temporal or camera filtering, and marking unknown persons. This integration of backend smarts and user-accessible functionality becomes a robust solution for security, attendance, and room space occupancy management.

In general, the project effectively proves a real-time, privacy-aware, and modular surveillance solution that can be implemented in academic and professional settings with potential future additions like analytics, live video viewing, and cross-platform mobile app development.

REFERENCES

[1]. Schroff, Florian, Dmitry Kalenichenko, and James Philbin. FaceNet: A Unified Embedding for Face Recognition and Clustering. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2015.

[2]. Redmon, Joseph, Santosh Divvala, Ross Girshick, and Ali Farhadi. You Only Look Once: Unified, Real-Time Object Detection. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2016.

[3]. Wojke, Nicolai, Alex Bewley, and Dietrich Paulus. Simple Online and Realtime Tracking with a Deep Association Metric. IEEE International Conference on Image Processing (ICIP), 2017.

[4]. Taigman, Yaniv, Ming Yang, Marc'Aurelio Ranzato, and Lior Wolf. DeepFace: Closing the Gap to Human-Level Performance in Face Verification. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2014.

[5]. Ghimire, Aayush, Naser El Werghi, Salman Javed, and Jorge Dias. Real-Time Face Recognition System. arXiv preprint arXiv:2204.08978, 2022.

[6]. Zhang, Kaipeng, Zhanpeng Zhang, Zhifeng Li, and Yu Qiao. Joint Face Detection and Alignment Using Multi-task Cascaded Convolutional Networks. IEEE Signal Processing Letters, vol. 23, no. 10, 2016, pp. 1499–1503.

[7]. Howard, Andrew G., et al. MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications. arXiv preprint arXiv:1704.04861, 2017.

[8]. OpenCV & Dlib Documentation. OpenCV.org, 2023, https://docs.opencv.org and https://github.com/davisking/dlib.

[9]. Firebase Documentation. Firebase.google.com, 2023, [**https://firebase.google.com/docs](https://firebase.google.com/docs)**.

[10]. Goyal, Varun, Rajeev Sharma, and Rachit Wason. Face Detection and Tracking Using OpenCV. International Journal of Computer Applications, vol. 179, no. 27, 2017