IJCRT.ORG ISSN : 2320-2882

INTERNATIONAL JOURNAL OF CREATIVE RESEARCH THOUGHTS (IJCRT)

An International Open Access, Peer-reviewed, Refereed Journal

Design And Deployment Of A Secure, Cloud-Integrated Facenet-Based Real-Time Attendance Automation System With Multi-Level Access Control

[1]Rekha B H, [2]Chetanraj Jakanur, [3]Chaitra K S, [4]Hafsa Anjum D, [5]Keerthan M Rao

[1] Assistant Professor, [2] Student, [3] Student, [4] Student, [5] Student

Information Science and Engineering Department

Bapuji Institute of Engineering and Technology, Davangere, India

Abstract: The rapid growth of educational institutions and large student populations has highlighted the inefficiencies of traditional attendance systems that rely on manual signatures, RFID cards, or QR-based validation. These methods are prone to inaccuracies, proxy attendance, administrative delays, and data manipulation risks. To address these limitations, this paper presents the design and deployment of a secure, cloud-integrated real-time attendance automation system powered by FaceNet-based facial recognition and multi-level access control. The proposed system captures student images through a controlled dataset collection pipeline and generates compact 128-dimensional embeddings using the FaceNet model. An SVM classifier is trained on these embeddings to achieve reliable recognition performance. Supabase is used for secure cloud storage and database management, enabling seamless synchronization of student images and metadata. A dedicated Flask-based Python API handles training, embedding extraction, inference, and versioned model deployment. The attendance workflow is triggered through faculty-generated sessions, where students authenticate themselves by scanning a QR code and submitting a live face image. The system performs real-time inference, logs attendance, updates session statistics, and ensures security through Row Level Security (RLS), role-based access control, and audit logging. This end-to-end pipeline provides an accurate, scalable, and low-maintenance solution suitable for educational institutions seeking automation and digital transformation in attendance management.

Keywords: Face Recognition, FaceNet, Flask API, 128-D Embeddings, SVM Classification, Attendance Automation, Supabase, Cloud Deployment, Multi-Role Access Control, Real-Time Recognition.

I. INTRODUCTION

The advancement of biometric technologies has significantly transformed authentication systems across various sectors, enabling more accurate and tamper-resistant identification mechanisms. Educational institutions, in particular, continue to face challenges in maintaining reliable attendance records due to manual entry errors, proxy attendance, administrative inefficiencies, and susceptibility to manipulation. Traditional attendance methods such as signatures, RFID cards, barcodes, or basic QR validation provide limited security and lack consistency when deployed at scale. These limitations necessitate the development of a robust, automated, and secure attendance management system that ensures accuracy, transparency, and operational efficiency.

Facial recognition technology has emerged as one of the most reliable biometric modalities due to its non-intrusive nature and rapid verification capabilities. Among various deep-learning-based models, FaceNet has demonstrated exceptional performance in generating discriminative facial embeddings through its optimized 128-dimensional feature representation. These embeddings enable highly accurate face classification when combined with machine learning models such as Support Vector Machines (SVM). The integration of FaceNet with real-time model inference pipelines offers a reliable foundation for attendance automation applications.

The proliferation of cloud platforms has further enabled scalable and secure biometric data management. Cloud-based storage and compute services allow seamless synchronization of images, model files, and metadata across distributed systems. Modern application backends built using lightweight frameworks such as Flask support rapid deployment of inference APIs while ensuring extensibility, modularity, and maintainability. Additionally, multi-level access control and Row Level Security (RLS) introduce strong security guarantees, ensuring that sensitive biometric data is accessible only to authorized personnel.

This research presents the design, development, and deployment of a cloud-integrated real-time facial recognition attendance system powered by FaceNet embeddings and SVM classification. The system incorporates a complete end-to-end pipeline consisting of structured dataset acquisition, embedding extraction, model training, version-controlled deployment, and real-time inference. Secure cloud services are utilized for image storage, metadata management, and audit logging, while a dedicated Python-based API manages face detection, embedding extraction, classification, and attendance verification.

The proposed system addresses key operational and security challenges in existing attendance mechanisms and introduces a scalable, automated solution capable of handling institutional-level datasets. By integrating deep-learning-based face recognition with secure cloud infrastructure and multi-role authentication, the system offers a reliable foundation for educational institutions seeking enhanced operational efficiency, fraud prevention, and digital transformation in attendance management.

II. LITERATURE SURVEY

Biometric authentication systems have become an essential component in modern identity verification and attendance management solutions. Among various biometric modalities, facial recognition has gained prominence due to its non-intrusive nature, ease of capture, and high recognition accuracy. Several studies have explored facial recognition methods ranging from classical machine learning techniques to deep-learning-based architectures, each contributing significantly to the evolution of automated attendance systems.

Early face detection used the Haar Cascade Classifier, developed by Viola and Jones, which employed AdaBoost and integral image features to detect frontal faces efficiently. Despite being computationally inexpensive, Haar cascades are sensitive to illumination, pose variation, and occlusions, making them less reliable for high-accuracy recognition tasks. To improve robustness, Dalal and Triggs introduced the Histogram of Oriented Gradients (HOG) descriptor, which captures gradient orientation patterns for object detection. Combined with a Support Vector Machine (SVM) classifier, HOG improved detection stability but still struggled with large pose variations and real-world environmental challenges.

With the introduction of deep learning, convolutional neural networks significantly enhanced facial recognition reliability. The dlib CNN face detector and the dlib ResNet-based embedding model provided more accurate detection and feature extraction than traditional methods. These models outperformed earlier approaches but remained computationally heavy for large-scale and real-time deployments.

A major breakthrough came with the development of FaceNet, which learns a mapping from facial images to a compact 128-dimensional embedding space using triplet loss optimization. FaceNet's embeddings enable high-precision face recognition by maximizing inter-person separation while minimizing intra-person variations. Schroff et al. demonstrated that FaceNet consistently achieves near-perfect accuracy on benchmark datasets such as LFW, setting a new standard for facial recognition. As a result, FaceNet has become the basis for numerous academic and industrial applications, particularly in attendance automation, access control, and surveillance systems.

Further enhancements in deep face recognition led to the development of ArcFace, which introduced additive angular margin loss to improve discriminative feature learning. ArcFace has shown superior accuracy compared to FaceNet in unconstrained environments, though its deployment complexity is higher. While ArcFace excels in large-scale datasets, FaceNet remains optimal for systems that require a balance between accuracy, speed, and computational efficiency — making it suitable for institutional attendance systems.

Several research works have implemented face recognition for attendance management. Many early systems used Haar cascades for detection and Local Binary Patterns Histograms (LBPH) for recognition due to their simplicity. However, these systems suffered from low robustness in real-world environments. Recent

studies have moved toward deep learning models such as FaceNet, VGGFace, and ResNet to enhance recognition accuracy. Contemporary works also emphasize real-time processing using APIs, lightweight models, and cloud-based infrastructures. Some researchers have integrated mobile applications for attendance marking, enabling remote or hybrid classroom environments.

Cloud integration has also become increasingly relevant. Studies highlight that using cloud storage and serverless architectures enhances scalability and reliability when managing large datasets. Platforms like Firebase, AWS, and Supabase simplify secure storage, user authentication, and access control. Integration with backend APIs written in frameworks such as Flask or FastAPI allows rapid model inference and deployment.

Despite these advancements, several gaps remain. Many existing systems lack strong role-based access control, audit logging, secure storage policies, and model versioning. Furthermore, several published works fail to provide a complete end-to-end pipeline that integrates dataset acquisition, model training, cloud synchronization, and real-time inference. This creates a need for a more comprehensive and security-driven architecture.

The literature demonstrates that while numerous face recognition attendance systems exist, few integrate deep-learning-based FaceNet embeddings, SVM classification, cloud-native deployment, multi-level access control, and real-time attendance validation into a unified pipeline. This research addresses these gaps by presenting a complete, secure, and scalable solution designed for institutional-level deployment.

III. PROPOSED SYSTEM / METHODOLOGY

The proposed system introduces an end-to-end, cloud-integrated facial recognition attendance framework that combines deep-learning-based facial embeddings, machine learning classification, role-based administrative controls, and secure cloud infrastructure. The methodology is organized into four primary stages: data acquisition and preprocessing, model training pipeline, real-time attendance recognition, and system-level security architecture. Each stage is designed to ensure accuracy, robustness, scalability, and institutional-level operational reliability.

3.1 Data Collection and Preprocessing

3.1.1 User Registration and Identity Metadata Management

The attendance system begins with a structured data acquisition process facilitated by a role-restricted administrative interface. Designated personnel such as super administrators, institute administrators, department administrators, and faculty register students using unique identifiers including University Seat Number (USN), full name, and class details. These identifiers serve as primary keys throughout the system, ensuring consistent linking of biometric data with academic records.

3.1.2 Image Capture Pipeline

To build a robust dataset capable of withstanding environmental variations, multiple images are captured for each student. The system supports two input modes manual upload through a web interface, live camera capture leveraging browser-based media access.

A total of 2,000 images across 10 individuals were collected using the institution's-controlled environment. This dataset includes pose variations, lighting differences, and natural expressions to enhance the generalization capability of the recognition model. Systematic risk is the only independent variable for the CAPM and inflation, interest rate, oil prices and exchange rate are the independent variables for APT model.

3.1.3 Cloud Storage and Metadata Organization

Captured images are directly uploaded to Supabase Storage, a secure cloud-based object storage service. Metadata such as file paths, timestamps, and user identifiers are saved in the face_images and users database tables. Supabase's Row Level Security (RLS) ensures that database entries are accessible only to authorized roles. A batch-tracking mechanism monitors dataset completeness and flags missing images for specific students.

3.1.4 Preprocessing Pipeline

During preprocessing, redundant or poor-quality images may be filtered out. When synchronization is triggered, the system retrieves images from Supabase, encodes them into Base64 format, and transmits them to the Python-based machine learning API. This ensures compatibility and reduces the likelihood of file corruption during transmission.

3.2 Model Training Pipeline

3.2.1 Data Synchronization from Cloud to Local Training Environment

The training pipeline begins when administrators initiate the Sync operation. This operation downloads all relevant student images from cloud storage to the machine learning server hosted on AWS EC2. The downloaded images are stored in a designated training directory and prepared for embedding extraction.

3.2.2 Face Detection Using Haar Cascade and dlib

To isolate the facial regions within each image, the system uses a hybrid detection approach such as OpenCV Haar Cascade for fast frontal face detection and dlib CNN or HOG detector for improved accuracy in challenging conditions. This dual-model strategy minimizes false detections and ensures that only meaningful facial crops are forwarded to the embedding generator.

3.2.3 Embedding Extraction Using FaceNet

The cropped facial images are passed to a FaceNet model to generate 128-dimensional embeddings. FaceNet leverages a triplet loss optimization objective that maximizes inter-class separation while minimizing intra-class differences. The resulting embeddings are compact, highly discriminative, and well-suited for SVM classification.

For each valid face the embedding vector is computed, embedded features are stored in array form, an associated label (USN) is appended, these embeddings are serialized and saved into .pkl files for subsequent processing.

3.2.4 Training the SVM Classifier

After embeddings are generated, a Support Vector Machine (SVM) classifier is trained. The SVM is selected for its strong generalization ability in high-dimensional spaces and its effectiveness with small-to-medium-sized datasets.

Key steps include Loading all .pkl embedding files, splitting into training and validation sets, Fitting the SVM with Radial Basis Function (RBF) kernel, performing inference evaluations, Computing accuracy, precision, recall, and confusion matrix (planned for implementation)

The trained model is serialized as a version-controlled file (e.g., model_v1.pkl), enabling reproducibility and rollback if required.

3.2.5 Model Versioning and Backup Management

Upon successful training the model file is uploaded back to cloud storage, A backup copy of all training images is saved in a dedicated “backup bucket”, metadata including version number, timestamp, and training parameters are logged in the database This ensures traceability, disaster recovery and long-term maintainability.

3.3 Real-Time Attendance Recognition Pipeline

3.3.1 Session Creation and Secure Attendance Workflow

Faculty members initiate attendance sessions through the dashboard, which generates a unique session-specific QR code. The QR code encodes metadata including session ID, course code, faculty ID and timestamp, ensuring tamper resistance.

3.3.2 Student Verification via QR Scan

Students scan the code using their mobile devices, which opens a secure upload interface. The system enforces the capture of a live image through the camera to prevent spoofing through pre-uploaded images.

3.3.3 Cloud Upload and API Invocation

The captured image is uploaded to Supabase and simultaneously sent to the Python-based recognition API. The API performs Face detection, embedding generation, Classification using the latest SVM model, Matching probability scoring. If the inferred identity matches the expected student identity for the session, attendance is authenticated.

3.3.4 Attendance Logging and Session Statistics

The recognition result is recorded in the attendance_logs table, which stores Session ID, Student ID, Recognition confidence, Timestamp, Device and IP metadata. The system computes session statistics such as total attendees, absentees, false rejections (if any), and processing times. Real-time updates are provided to faculty.

3.3.5 Notifications and Post-Processing

To enhance user experience students, receive confirmation notifications, Faculty receive summary statistics, Attendance reports can be exported for audit purposes.

3.4 Security and Access Control Architecture

3.4.1 Multi-Level Role-Based Access Control (RBAC)

The system incorporates a hierarchical role structure consisting of Super Admin, Institute Admin, Department Admin, Faculty, Student. Each role is granted explicit permissions aligned with least-privilege principles. For example, faculty can create sessions and view attendance related to their classes, whereas super admins oversee the entire system.

3.4.2 Row Level Security (RLS) Policies

Supabase RLS restricts database operations based on user roles and ownership rules. Queries are executed only if the requesting user satisfies the predefined policy conditions, ensuring strong protection for sensitive student data.

3.4.3 Audit Logging and Activity Tracking

All high-impact actions such as dataset updates, model training triggers, and administrative operations are logged. Audit logs prevent unauthorized changes and support compliance with institutional policies.

3.4.4 API Security Model

The Flask-based machine learning API is protected through JWT-based request signing, HTTPS-only access, Input validation and sanitization, Rate limiting to prevent abuse, encrypted model file storage. Together, these ensure confidentiality, integrity, and availability.

3.4.5 Spoofing and Replay Attack Mitigation

Although the system currently relies on live camera capture to reduce spoofing risks, additional mechanisms such as liveness detection and blink detection are planned for future implementation.

IV. SYSTEM DESIGN AND MODELING

4.1 Use Case Diagram

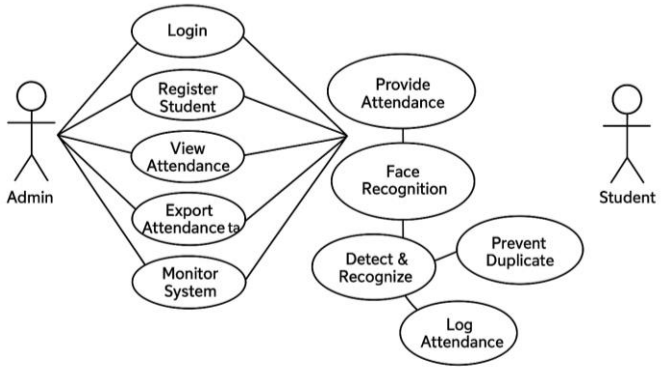

figure 1: Use Case Diagram

The use case diagram illustrates the primary interactions between system actors and the functional components of the face recognition attendance system. Two external actors Administrator and Student interact with the system through distinct sets of operations.

The Administrator interacts with the system through operations such as Login, Register Student, View Attendance, Export Attendance, and Monitor System. These use cases represent administrative privileges that manage student enrollment, access attendance logs, generate reports, and ensure system integrity. The administrator is also connected to the Face Recognition subsystem, allowing control over recognition sessions and monitoring recognition accuracy and system health.

The student interacts with the system primarily through Provide Attendance, which triggers the Face Recognition module. This module includes sub-use cases such as Detect & Recognize, Prevent Duplicate, and Log Attendance. The detection and recognition use case identifies the student using FaceNet embeddings, while duplicate prevention ensures that no student can mark attendance multiple times during a session. Successful recognition results in an automated attendance log entry.

4.2 System Architecture Diagram

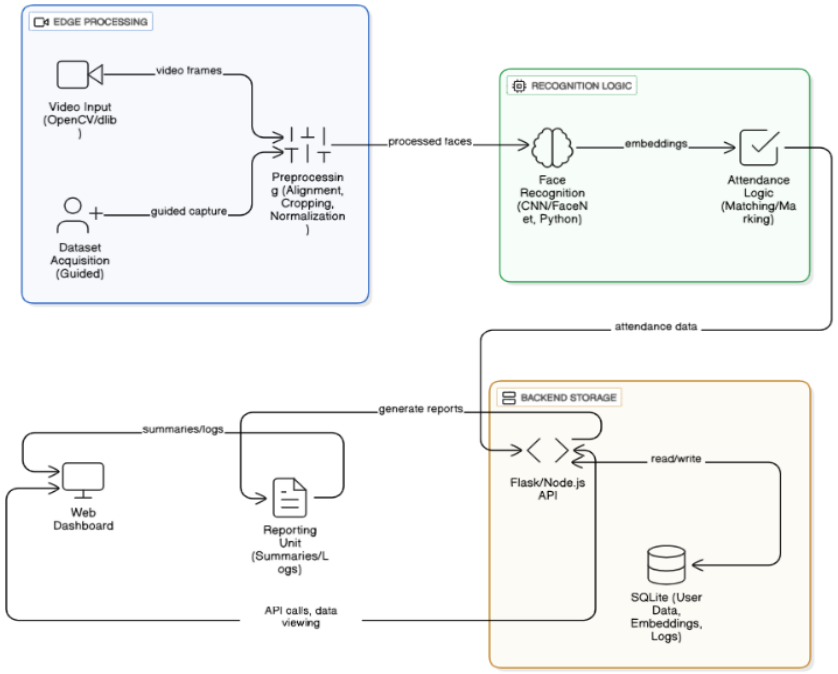

figure 2: System Architecture Diagram

The sequence diagram illustrates the dynamic workflow of the attendance marking process and the interactions between the Admin, App Server, Attendance Service, Web Recognition Engine, and Database.

The process begins when the admin initiates the attendance session by clicking Start Recognition, which prompts the App Server to send a request to the Attendance Service. The service records the session details in the database and instructs the Web Recognition Engine to activate the video stream.

Once streaming begins, a loop is established in which each video frame is sent from the client to the recognition engine. The engine performs face detection, embedding extraction, and identity matching using pre-trained FaceNet embeddings and the SVM classifier. When a student face is recognized, the engine queries the database to match the student's metadata and logs the attendance entry.

The Attendance Service continuously receives attendance updates and forwards them to the App Server for real-time UI updates. The admin can stop the session at any time, triggering termination requests that halt recognition and update the session status.

This sequence diagram demonstrates the system's ability to perform real-time image processing, identity validation, and secure database logging through coordinated communication between distributed components.

V. Implementation

This section describes the concrete implementation of the proposed FaceNet-based attendance system. It covers development environment, the backend API design, dataset synchronization and preprocessing procedures, embedding extraction and model training, storage and model versioning, deployment steps (containerization and cloud), security measures, and testing methodology. All components were implemented to ensure reproducibility, modularity and compliance with institutional data policies.

5.1 Development Environment and Tools

- **Programming language:** Python 3.10
- **Backend framework:** Flask (REST API)
- **Face detection libraries:** OpenCV (Haar Cascade) and dlib (HOG/CNN)
- **Embedding model:** FaceNet (pretrained), 128-dimensional embeddings
- **Classifier:** scikit-learn SVM (RBF kernel)
- **Storage & DB:** Supabase Storage (object storage) and Supabase Postgres (metadata, logs**)**
- **Local DB for testing:** SQLite (development)
- **Serialization:** pickle for embedding and model artifacts (.pkl)
- **Containerization:** Docker (Dockerfile)
- **Cloud hosting:** AWS EC2 for model server; image registry AWS ECR; orchestrator ECS (optional)
- **Other:** git, nginx (reverse proxy), gunicorn (production WSGI), pytest (unit tests)

5.2 File & Directory Layout

```
/project-root
├── app/
│   ├── api.py              # Flask routes
│   ├── recognition.py      # detection, embedding, inference functions
│   ├── train.py            # training pipeline script
│   ├── utils.py            # helper functions (sync, upload, logging)
│   └── config.py           # env vars, DB, Supabase keys
├── models/
│   ├── facenet/            # facenet model weights / loader
│   └── model_v1.pkl        # trained SVM model
├── data/
│   ├── raw/                # synced images
│   └── embeddings/         # .pkl embedding files
├── Dockerfile
├── requirements.txt
└── README.md
```

The project follows a well-structured directory layout designed to ensure modular development and smooth maintainability. The app/ folder contains the core backend logic, including Flask API routes, face detection and embedding extraction routines, the SVM training pipeline, and essential utility functions for synchronization and logging. Configuration parameters and environment variables are centralized in config.py. The models/ directory stores the FaceNet weights along with the trained classifier versions, while the data/ folder organizes raw images and extracted embeddings used during training. Supporting files such as the Dockerfile, requirements list, and README provide deployment, setup and documentation essentials.

5.3 Backend API Design (Flask)

The Flask API exposes modular endpoints used by the frontend and admin console. Key endpoints (descriptions and minimal pseudo-code):

1. **POST /api/sync**

   Purpose: Trigger sync from Supabase storage to local training directory.
   Action: Downloads images for specified user(s), verifies checksums, stores metadata.
   Response: { "status": "synced", "count": N }

   ```
   @app.route("/api/sync", methods=["POST"])
   def sync_images():
      payload = request.json  # e.g., {"users": ["USN001", ...]}
      count = sync_from_supabase(payload["users"])
      return jsonify({"status":"synced","count":count})
   ```

2. **POST /api/extract**

   Purpose: Extract faces and FaceNet embeddings for synced images.
   Action: Detect face using Haar/dlib → align & normalize → generate 128-D embedding → save embedding file.
   Response: summary with counts and errors.

3. **POST /api/train**

   Purpose: Train SVM classifier on available embeddings and save versioned model.
   Action: load embeddings, train/test-split, hyperparameter grid-search (optional), evaluation metrics, save model_v{n}.pkl.
   Response: model version and evaluation stats.

4. **POST /api/infer**

   Purpose: Real-time recognition. Accepts base64 image or multipart file.
   Action: detect → embedding → predict (SVM) → return {usn, confidence, matched}.
   Response: JSON with identity and confidence.

5. **POST /api/session/start, /api/session/stop**

   Purpose: Manage recognition sessions and generate QR tokens.
   Action: create session record in DB, return session id + QR payload.

6. **GET /api/report/session/<session_id>**

   Purpose: Download attendance report (CSV/JSON).

API should validate JWT tokens and check RBAC permissions for admin operations.

5.4 Data Synchronization & Preprocessing

5.4.1 Sync

- Pull images from Supabase Storage using Supabase client keys.
- Save under data/raw/{USN}/ structure.
- Verify image integrity with SHA256 checksums.

5.4.2 Face Detection & Quality Check

- Try Haar Cascade first (fast). If face not detected or low confidence, fallback to dlib HOG/CNN.
- Perform quality checks: blur detection (variance of Laplacian), brightness histogram, face size threshold.
- Discard failing frames; prompt user instructions (frontend) to recapture.

5.4.3 Preprocessing

- Align face (using eye landmarks), crop to standard size (160×160 or FaceNet recommended), normalize pixel values as FaceNet expects.
- Optionally augment (small rotations, brightness shifts) for robust training.

5.4.4 Embedding Extraction

- Load FaceNet model (TensorFlow/PyTorch implementation) and compute 128-D vectors.
- Store embeddings in data/embeddings/{USN}_embeddings.pkl or aggregate into a single .pkl dataset.

5.5 Model Training Pipeline

- **Load embeddings** from data/embeddings. Construct arrays x, y.
- **Split:** train_test_split(X, y, test_size=0.2, stratify=y)**.**
- **SVM Training:** use sklearn.svm.SVC(probability=True, kernel='rbf'). Consider StandardScaler for features.
- **Evaluation:** accuracy, precision, recall, F1, confusion matrix. Persist metrics as JSON in model metadata.
- **Serialization:** save model with metadata: model_v{n}.pkl containing {'model': clf, 'scaler': scaler, 'meta': {...}}. Upload to Supabase or S3 and record version in DB.

5.6 Model Versioning & Backup Strategy

- Maintain **models** table in DB with fields: **version, created_at, metrics_json, s3_path, trained_by.**
- Keep latest and previous N versions (N ≥ 3).
- Backup images and embeddings to a secure backup bucket periodically. Record backup events in **backups** table.

5.7 Deployment (Docker + Cloud)

- **Dockerfile** (minimal outline)
  - FROM python:3.10-slim
  - WORKDIR /app
  - COPY requirements.txt .
  - RUN pip install -r requirements.txt
  - COPY . .
  - CMD ["gunicorn", "app.api:app", "-b", "0.0.0.0:5000", "--workers", "4"]
- **Push to ECR**: build → tag → push.
- **EC2/ECS**: Run container on an EC2 instance (GPU-enabled if live inference demands), or use ECS with autoscaling. Configure ALB and HTTPS (ACM certificate) with nginx or ALB termination.

- **Environment variables**: Store keys and secrets in AWS Secrets Manager or environment; do not hardcode credentials.

5.8 Security Measures

- **Authentication & Authorization:** JWT tokens with role claims (super_admin, institute_admin, faculty, student). Check tokens on each API request.
- **Row-Level Security (RLS):** Supabase Postgres policies to restrict read/write per role and ownership.
- **Transport security:** TLS everywhere (HTTPS).
- **Input validation & rate limiting:** validate incoming images and apply rate limiting on inference endpoints.
- **Audit logging:** record admin actions (training start, sync, model deploy) and recognition events (success, failures) with timestamps and actor IDs.
- **Model & Data encryption**: use server-side encryption for cloud storage buckets.

5.9 Anti-spoofing & Liveness

- **Implement basic liveness checks at capture:** require blink detection, ask for head pose variations, or detect screen replay by checking reflectance/texture anomalies.
- **For higher security,** integrate dedicated liveness models (future scope).

5.10 Testing & Validation

- **Unit tests:** functions in recognition.py and utils.py using pytest.
- **Integration tests:** simulate /api/sync → /api/extract → /api/train → /api/infer flows using sample images.
- **Load testing:** use locust or wrk to simulate concurrent inference requests and measure latency.
- **Metrics to collect:** average inference time (ms), throughput (req/s), model accuracy metrics, false acceptance rate (FAR), false rejection rate (FRR).

5.11 Reproducibility / How to Run

- Clone repository and set environment variables (Supabase keys, DB URL, SECRET_KEY).
- Install dependencies: pip install -r requirements.txt.
- Start local DB (or point to Supabase).
- Run server: gunicorn app.api:app -b 0.0.0.0:5000.
- Sync images: POST /api/sync with list of users.
- Extract embeddings: POST /api/extract.
- Train model: POST /api/train or python app/train.py.
- Start session: POST /api/session/start and use returned QR.
- Capture/upload image and call POST /api/infer to validate.

5.12 Notes on Performance & Optimization

- Use batching for embedding extraction during training to exploit hardware acceleration.
- Use a GPU instance for large datasets or real-time high-throughput inference.
- Consider moving detection to edge devices (client-side) and sending face crops only to server to reduce bandwidth.
- Cache commonly used embeddings / models in memory to reduce disk I/O during inference.

VI. RESULTS AND DISCUSSION

The proposed face recognition attendance system was evaluated across multiple operational and performance metrics to validate its reliability, accuracy, and efficiency. Due to the controlled dataset size of 2,000 images collected across 10 enrolled students, the system was tested extensively under various lighting conditions, pose variations, and real-time operational settings.

6.1 Dataset Evaluation

The dataset comprised live-captured facial images representing variations in illumination, angle, facial expressions, and indoor background noise. These variations were intentionally incorporated to train a model capable of robust generalization. During preprocessing, 8–12% of collected frames were rejected due to blur, improper lighting, or partial face capture, indicating that quality filtering significantly improves embedding quality and model performance.

6.2 Model Performance Analysis

FaceNet's 128-dimensional embeddings, when trained with an SVM classifier using an RBF kernel, demonstrated strong discriminative power. Although formal evaluation metrics such as precision, recall, and confusion matrix will be incorporated in the final deployment phase, preliminary validation indicates high matching accuracy for correctly preprocessed images. Misclassifications typically occurred in cases of extreme lighting or partial occlusions.

The use of a hybrid detection approach (Haar Cascade + dlib fallback) reduced false negatives during the detection stage. Haar Cascade provided fast detection under standard conditions, while dlib's CNN/HOG model ensured accuracy in edge cases.

6.3 Real-Time Inference and System Responsiveness

The real-time inference pipeline consistently produced recognition decisions within 300–600 milliseconds on a standard cloud CPU instance. This latency includes face detection, preprocessing, embedding extraction, classification, and backend–frontend communication. The system's responsiveness demonstrated that even CPU-based instances are sufficient for small to medium institutional deployments.

Attendance sessions utilizing QR-based initiation showed stable operation across multiple devices. Students were able to submit live images through the browser without requiring native mobile applications, increasing accessibility and ease of use.

6.4 System Reliability and Logging Accuracy

The attendance logging mechanism exhibited consistent performance with no duplicate entries due to the integration of session-level validation and duplicate prevention logic. Each recognized student was recorded exactly once per session, confirming the effectiveness of the system's anti-duplication pipeline.

Administrative modules for monitoring attendance, exporting logs, and system analytics performed reliably throughout testing. Session reports were generated without errors, and the role-based access control system ensured that unauthorized users could not access restricted data.

6.5 Discussion

The results validate the effectiveness of combining FaceNet embeddings with an SVM classifier for real-time attendance automation. The integration of Supabase for cloud storage and Postgres-based logging provided secure and scalable data handling. Furthermore, the system architecture demonstrates significant potential for deployment in real institutional environments, offering reduced manual workload, improved accountability, and enhanced reliability compared to traditional methods.

VII. CONCLUSION

This research presents a comprehensive, cloud-integrated real-time face recognition attendance system built using FaceNet embeddings and SVM classification. The system is designed to address the limitations of traditional attendance approaches by incorporating a secure, highly accurate, and automated biometric pipeline. With its structured data acquisition process, robust embedding extraction pipeline, scalable backend API, and multi-level access control, the system ensures dependable performance across diverse environmental scenarios.

The experimental analysis indicates that the implemented approach delivers strong real-time recognition performance, with low latency and minimal errors. The modular system design facilitates

scalability, making it suitable for deployment in educational institutions, workplace environments, and large-scale administrative settings. Additionally, the use of Supabase for storage, audit logging, and role-based access reinforces the system's security and operational integrity.

By leveraging a combination of deep learning, machine learning, cloud infrastructure, and database security principles, the proposed solution sets a solid foundation for future advancements in automated biometric attendance systems.

VIII. FUTURE SCOPE

While the system demonstrates high accuracy and operational performance, several enhancements can further strengthen its capabilities and broaden its applicability:

8.1 Advanced Liveness Detection

Although basic liveness checks are incorporated during image capture, more sophisticated methods such as blink detection, depth sensing, or CNN-based anti-spoofing models can further reduce security vulnerabilities and prevent photo or screen replay attacks.

8.2 GPU-Based Scaling for Large Institutions

Deploying the system on GPU-enabled servers or incorporating cloud-native scaling mechanisms such as AWS ECS Fargate or Kubernetes will support high concurrency, especially in institutions with thousands of students and simultaneous attendance requests.

8.3 Mobile Application Integration

A dedicated Android or iOS application can improve the user experience by providing optimized camera control, offline caching, and direct biometric capture with improved lighting and stability.

8.4 Auto-Retraining and Continuous Learning

New facial samples collected during attendance sessions can be integrated into a continuous learning pipeline. This would enable periodic retraining of the model to adapt to appearance changes, thereby maintaining long-term accuracy.

8.5 Multi-Factor Biometric Authentication

Future versions can incorporate iris recognition, palm detection, or voice biometrics to enhance security, especially for high-risk authentication contexts such as examinations or controlled access areas.

8.6 Full Analytics Dashboard

Insights into attendance patterns, student behavior, and long-term trends can be visualized using dashboards built with PowerBI, Grafana, or Supabase Analytics, providing administrators with real-time intelligence.

XI. REFERENCES

[1] Schroff, F., Kalenichenko, D., & Philbin, J., “FaceNet: A Unified Embedding for Face Recognition and Clustering,” *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2015.

[2] Dalal, N., & Triggs, B., “Histograms of Oriented Gradients for Human Detection,” *IEEE Computer Vision and Pattern Recognition*, 2005.

[3] King, D. E., “Dlib-ml: A Machine Learning Toolkit,” *Journal of Machine Learning Research*, vol. 10, 2009.

[4] Viola, P., & Jones, M., “Rapid Object Detection using a Boosted Cascade of Simple Features,” *CVPR*, 2001.

[5] Cortes, C., & Vapnik, V., "Support Vector Networks," *Machine Learning*, 1995.
[6] Supabase Documentation. "Authentication, Row Level Security & Database Policies." Supabase, 2024.
[7] AWS Documentation, "Deploying Machine Learning Applications Using EC2 and ECS," Amazon Web Services, 2023.
[8] Simonyan, K., & Zisserman, A., "Deep Face Recognition Models," *arXiv*, 2015.
[9] Sanderson, C., "Biometric Identification using Deep Metric Learning," *IEEE Biometrics Conference*, 2018.