iJRASET
International Journal For Research in Applied Science and Engineering Technology

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Volume: 14 **Issue:** V **Month of publication:** May 2026

DOI: https://doi.org/10.22214/ijraset.2026.82094

www.ijraset.com

Call: 08813907089 | E-mail ID: ijraset@gmail.com

International Journal for Research in Applied Science & Engineering Technology (IJRASET)
ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.538
Volume 14 Issue V May 2026- Available at www.ijraset.com

Crowdnexis AI: An Integrated AI Platform for Real-Time Crowd Risk Assessment and Missing Person Detection in Mass Gatherings

Krishna Chandrakant Malode[1], Ayush Shashank Kulkarni[2], Dnyanesh Sameer Patil[3], Keyur Prashant Kolhe[4], Kunal Ramesh Ahire[5]
Department of Information Technology MET's Institute of Engineering, MET Bhujbal Knowledge City Savitribai Phule Pune University (SPPU), Nashik, Maharashtra, India

Abstract: ***Event security teams face a problem that does not get enough attention: two distinct failure modes, crowd density exceeding safe limits and a missing person somewhere in surveillance footage, share the same video data but almost always get handled by separate tools. We built Crowdnexis AI to close that gap. The system runs a single pipeline: a React and TypeScript frontend feeds video into a Node.js and Express backend, which spawns Python subprocess scripts for all AI inference. YOLOv8 handles person detection frame by frame. The counts from that detection feed into a four-tier risk classifier: SAFE, MODERATE, RISKY, CRITICAL. In parallel, face candidates pulled from the same sampled frames get compared against enrolled missing-person embeddings using 512-dimensional FaceNet vectors and cosine similarity at a 0.60 threshold. FAISS acceleration is available for larger person databases. Every result, alert, and match outcome writes to PostgreSQL. Operators interact through a dashboard, a filterable alerts panel, and a chatbot that accepts queries in English, Hindi, and Marathi. We define an evaluation framework covering crowd-count error, risk classification accuracy, face-match precision and recall, and full-path alert latency. This paper covers the architecture, the key design decisions, the implementation, and that evaluation plan.***

Keywords—crowd density estimation, YOLOv8, face recognition, FaceNet, missing person detection, public safety AI, FAISS, multilingual chatbot, real-time surveillance.

I. INTRODUCTION

Religious events, concerts, sports fixtures, political rallies: these all produce crowds that shift from manageable to dangerous faster than a monitoring operator watching twelve simultaneous camera feeds can track. Crowd density is one thing to measure. Locating a specific face among thousands of people in that same footage is a separate problem. Existing technical work treats them as unrelated. We disagree, and Crowdnexis AI is the result.

The bottleneck in manual CCTV monitoring is not equipment but attention, which arises because a single operator cannot accurately count people across dense frames, track density changes, and simultaneously scan for specific faces. A single operator cannot count people accurately across a dense frame, compute density trends, and also scan for a face, which makes real-time crowd surveillance unreliable under high-density conditions. The two tasks draw on the same video. Separating them into two unconnected systems doubles the operator workload without any gain in accuracy. We built around the observation that both tasks can share one pipeline, thereby reducing redundancy and improving system efficiency by leveraging the same video input for multiple analyses**.**The technical contribution here is a working, deployed system: YOLOv8 person detection connected to a FaceNet embedding pipeline, results and alerts written to a relational database, everything accessible through a usable operator interface. We also built a multilingual chatbot for operators who work in Hindi or Marathi, because an English-only interface is a real friction point in the deployment contexts we designed for. This paper covers the architecture, the choices we made and why, and a proposed evaluation framework built around metrics that matter in the field.

II. RELATED WORK

A. Crowd Counting and Density Estimation

Earlier crowd counting systems worked on hand-crafted features fed into regression models, which limited their performance in dense and occluded scenes**.**Sparse scenes were fine. Dense, occluded frames exposed their limits. The shift to density map

regression changed this: Lempitsky and Zisserman [14] showed that a linear mapping from local image features to a density map, where the integral of the map gives the crowd count, was learnable and practical. Zhang et al. [5] built on that with the Multi-Column CNN (MCNN), processing images at multiple scales to handle perspective distortion. CSRNet [4], which used a VGG-16 backbone with dilated convolutions, set a benchmark baseline at MAE 68.2 on ShanghaiTech Part A.

From around 2018, channel-level scale attention became the dominant technique. Networks weighted features according to local density rather than treating the whole frame uniformly. Guo and Li [1] added a Coordinate Attention Module to standard scale-aware feature fusion in the CAI-CAN architecture, encoding spatial position along horizontal and vertical axes separately. That got MAE down to 61.5 on ShanghaiTech Part A, a 9.8% improvement over CSRNet. Deng et al. [3] observed that benchmark gains on established datasets are slowing, and that combining crowd estimation with tasks like person re-identification is a more productive direction. TransCrowd [7] reframed the problem as sequence-to-count using attention, with competitive results under weak supervision. Cross-dataset generalisation remains a hard problem across all of these methods, which limits their real-world deployment in uncontrolled environments.

B. Face Recognition in Surveillance Environments

Face recognition moved into deep learning through DeepFace in 2014, then VGGFace. FaceNet [6] changed the architecture significantly: metric learning via triplet loss produced 128-dimensional embeddings where Euclidean distance directly encodes face similarity. ArcFace extended this with angular margin loss and pushed accuracy higher on controlled benchmarks. These embedding-based approaches work well for surveillance because you enroll faces once and then query the embedding space at match time, with no need to retrain.

Real surveillance is harder than controlled benchmarks. Lee, Lee, and Chiang [2] ran a four-month study in a classroom using ceiling-mounted CCTV cameras, training CNN models on face images as small as 20 by 20 pixels. Recognition accuracy stayed above 96% at that resolution, with degradation of roughly 0.25 percentage points per month over the study period. Abid, Mustafa, and Ahmad [12] combined RetinaFace and FaceNet on campus video and reached 99.86% accuracy. Khan et al. [13] demonstrated that lightweight FaceNet-based deployments work on constrained hardware. Super-resolution preprocessing using multi-stage FSRCNN [11] is another option for degraded input, though it adds latency.

C. Integration of Crowd Analytics and Person Detection

Few systems try to run crowd density analysis and face-based missing-person search inside the same operational loop. YOLOv3 [8] showed that real-time person detection at 30-plus frames per second is achievable in surveillance contexts, which makes frame-level face extraction practical. FAISS [15] provides approximate nearest-neighbour search over large embedding databases at millisecond scale. Crowdnexis AI puts these pieces together inside a structured, database-backed, operator-facing system. Jobs, alerts, persons, and match outcomes all persist in one place.

III. SYSTEM DESIGN AND METHODOLOGY

A. Architecture Overview

Crowdnexis AI uses a three-tier setup. The React and TypeScript frontend handles operator interaction. The Node.js and Express backend manages API routing and processing coordination. Python subprocess scripts run all AI inference. We kept Python separate from the application server deliberately: the Python ecosystem for computer vision (PyTorch, Ultralytics, facenet-pytorch) is considerably more capable than its JavaScript equivalents for these tasks, and running inference as a subprocess avoids tight coupling without requiring a dedicated inference microservice. PostgreSQL stores all persistent state. Redis supports queue and socket infrastructure. Table I shows the full component layout.

TABLE I CROWDNEXIS AI COMPONENT ARCHITECTURE

Layer	Component	Technology
Presentation	Operator Dashboard, Alerts, Chat, Missing Persons UI	React 18 + TypeScript + Vite
API / Orchestration	Auth, Video, Person, Stats, Chat Routes	Node.js + Express 4

AI / CV Engine	Crowd Detection, Face Enrollment, Video Face-Match	Python (YOLOv8, FaceNet-PyTorch)
Persistence	Video jobs, alerts, detections, persons, match outcomes	PostgreSQL 15
Infra	Queue management, socket telemetry	Redis 7 + Socket.IO + BullMQ

Fig. 1 shows the five-layer architecture of the deployed system.

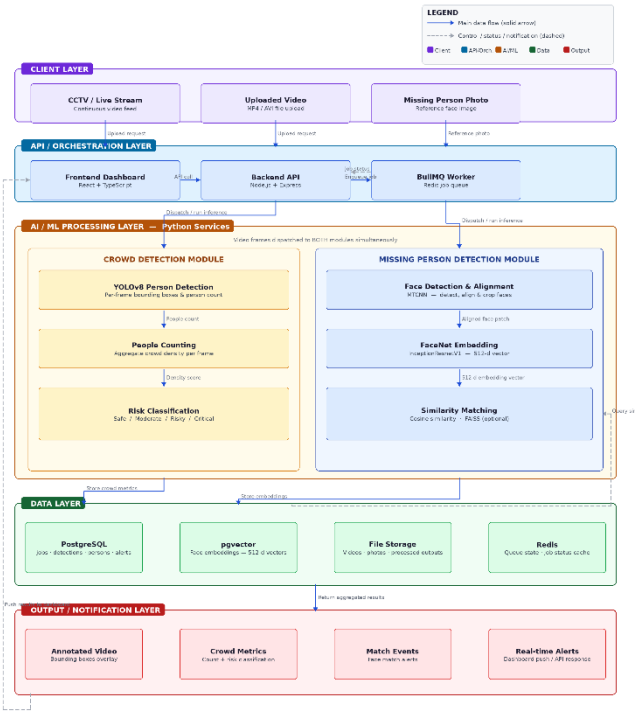


Fig. 1. Layered system architecture ofCrowdnexisAI showing client, orchestration, AI processing, data, and output layers.

B. Data Flow

The runtime flow has three coordinated paths. Authentication goes first: the frontend sends credentials to the Express backend, which validates them with bcrypt against stored hashes and returns a JWT. That token attaches to all subsequent API calls.

Missing person registration is a two-step process. The operator submits a profile and photo through a multipart form. The backend writes the metadata to PostgreSQL, then spawns a Python enrollment script. That script runs MTCNN to detect and align the face, then passes the aligned face through InceptionResnetV1 to produce a 512-dimensional embedding. The embedding is L2-normalised before it goes into the missing_persons table.

Video processing is the core of the system. The operator uploads a video file (up to 1 GB). The backend writes a job record to video_jobs, saves the file to local storage, and asynchronously spawns the Python crowd detection script. That script samples frames at configured intervals to keep compute manageable. On each sampled frame it runs YOLOv8 person detection on class 0 (person), updating running maximum count, average per frame, and frame-level progress. At the same time, it extracts face candidates from those frames, computes embeddings, and compares them against enrolled missing-person embeddings via cosine similarity (inner product on normalised vectors). Matches above 0.60 trigger detection records, flip person status to FOUND, and write alert log entries. All results write back to PostgreSQL. The frontend polls a results endpoint for progress, with socket events available for supplemental real-time updates where the wiring is active.

C. Risk Classification

Crowd risk is assigned from the per-video maximum person count, using four threshold bands drawn from operational practice in mass gathering safety:

- SAFE: fewer than 10 persons detected
- MODERATE: 10 to 24 persons
- RISKY: 25 to 49 persons
- CRITICAL: 50 or more persons

These thresholds are heuristic and do not originate from annotated ground truth data, but instead reflect operational conventions used by event safety teams, allowing practical deployment while remaining configurable for different venues**.**Venue-specific calibration is supported through configurable parameters, which allows adaptation to different crowd environments such as stadiums, festivals, and narrow corridors.

Fig. 2 shows the complete processing sequence from upload to result, including the per-frame detection and matching loop

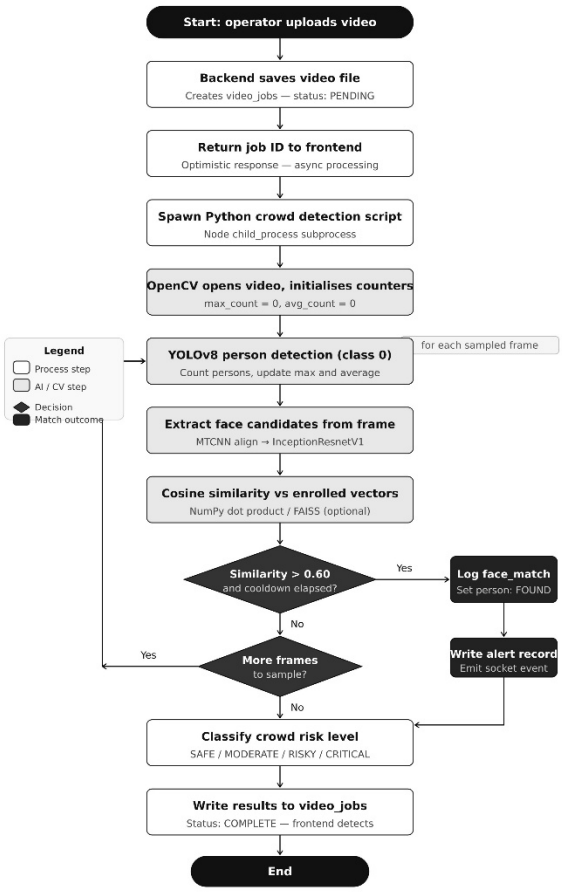


Fig. 2. Video crowd analysis and face matching pipeline showing the per-frame YOLOv8 detection loop, cosine similarity decision branch, and risk classification step.

D. Face Matching Logic

At enrollment, MTCNN detects and aligns the face in the submitted photo. InceptionResnetV1 produces a 512-dimensional embedding from the aligned face. That embedding is L2-normalised before storage, which means all later comparisons reduce to a dot product. During video processing, face candidates from sampled frames go through the same alignment and embedding pipeline. The inner product between a normalised frame embedding and each enrolled embedding gives the cosine similarity. Matches above 0.60 are recorded, as this threshold balances false positives and false negatives in typical FaceNet-based surveillance systems.A cooldown window prevents the same person from triggering repeated alerts across consecutive frames, which reduces alert duplication and improves operational clarity for monitoring teams.For larger person databases, an optional FAISS index provides approximate nearest-neighbour acceleration. Smaller deployments fall back to numpy.

IV. IMPLEMENTATION

A. Technology Stack

The frontend uses React 18 and TypeScript, bundled with Vite. react-router-dom handles routing. @tanstack/react-query manages server state and polling. Forms use react-hook-form with Zod validation schemas. Recharts renders the detection timeline. socket.io-client carries real-time events. The backend is Node.js with Express 4. Middleware includes helmet for security headers, express-rate-limit for abuse protection, multer for file uploads, and jsonwebtoken with bcrypt for authentication. PostgreSQL access uses the pg library. Redis uses ioredis. Winston handles structured logging.

All AI computation runs in Python. YOLOv8 inference goes through the Ultralytics library. Face detection and alignment use MTCNN from facenet-pytorch. Embedding extraction uses InceptionResnetV1 from the same package. Python writes to PostgreSQL directly via psycopg2. OpenCV reads video files and extracts frames. NumPy handles the vector arithmetic for similarity computation.

B. Key Engineering Decisions

The backend returns a 200 with the job ID immediately on video upload, before processing starts. Processing runs in the background via subprocess. This prevents HTTP timeout on large file uploads and keeps the frontend responsive, which is essential for maintaining usability during long-running video processing tasks**.**The frontend polls a status endpoint, using localStorage to hold the job ID across page refreshes during long-running jobs.

Face matching runs a cooldown window to stop alert flooding when the same person appears in several consecutive sampled frames. When a high-confidence match is confirmed, the matched person status updates to FOUND in the database automatically, giving a clear auditable state transition. The multilingual chatbot uses a keyword-scoring intent engine. Dictionary entries map Hindi and Marathi terms to English equivalents. The scored intent routes to a live SQL query for crowd status, person status, or alert retrieval.

Docker Compose handles local PostgreSQL and Redis setup. A SQL migration script creates the schema at startup. PowerShell and batch scripts cover Windows-based development environments.

V. RESULTS AND EVALUATION

A. Proposed Evaluation Framework

We designed the system to be evaluated on four dimensions. Full quantitative results against annotated ground truth are future work. The metrics are defined now so evaluation can proceed once test datasets are assembled.

For crowd density estimation, the primary metric is Mean Absolute Error (MAE) between YOLOv8 person counts and manual ground-truth counts across a representative set of test video clips. Root Mean Square Error (RMSE) captures how badly the system misbehaves on hard frames. Both metrics should be computed separately within each density band, so you know where the system holds and where it breaks down.

Risk classification accuracy is a four-class problem: SAFE, MODERATE, RISKY, CRITICAL. Evaluate with precision, recall, and F1 per class, plus a confusion matrix. In a safety-critical context, recall on the CRITICAL class is the metric that matters most operationally.

For face matching, compute ROC and precision-recall curves over a test set of enrolled persons and video clips. Measure False Acceptance Rate (FAR) and False Rejection Rate (FRR) explicitly at the 0.60 operating threshold. Also report top-1 and top-5 retrieval accuracy.

End-to-end latency, from video upload to first alert generation, is the operational metric that determines whether the system is useful in the field. Profile it in components: upload time, YOLO inference time, face embedding and matching time, and database write time.

B. Baseline Expectations

Based on published YOLOv8 benchmark results and FaceNet-style embedding results in low-resolution surveillance settings, we expect crowd count MAE in the range of 5 to 15 persons for low-to-medium density scenes (under 50 persons). Larger error is expected for dense, occluded frames. Face-match accuracy for enrolled persons with clean reference photos is expected to exceed 90% at the 0.60 threshold, with degradation for low-quality video or significant appearance change. These expectations come from the literature reviewed in Section II. They are not from the system itself.

VI. DISCUSSION

A. Strengths

The most useful aspect of the design is that crowd risk and missing-person detection are not two modules that happen to share infrastructure. They are wired together: when a face match fires, the record includes the crowd risk level at that moment. That joint record is more useful to an operator than either signal alone.

The subprocess model for Python inference is not architecturally elegant. It was the right call for this project scope. It kept the development surface small without needing the team to maintain a separate inference service. Because all state persists in the database, a component failure mid-pipeline does not lose results. The polling-first frontend means the UI stays functional even when socket events are unavailable.

The multilingual chatbot addresses a real friction point. Indian mass gathering security teams often work in Hindi or Marathi. An English-only interface is not a minor inconvenience in that context. Rule-based intent resolution is less flexible than a language model, but it is deterministic, which matters for a safety-critical command interface.

B. Limitations

Running subprocess calls means concurrent video jobs compete for the same CPU and GPU resources, which limits scalability under high-load scenarios. There is no distributed scheduling. Throughput drops under load. The system also has no mechanism to detect when the similarity threshold drifts out of calibration as the enrolled person database grows or as video quality changes.

The risk thresholds are not calibrated against annotated ground truth, which may result in variation in classification accuracy across different environments.A sports stadium and a narrow pilgrimage corridor have different density tolerances. The current implementation uses aggregate frame-level count, not spatial density distribution. A crowd compressed into one corner of the frame counts the same as the same number spread evenly. That is a real limitation in practice. Face recognition accuracy will degrade for low-quality reference photos, significant appearance changes between enrollment and detection time, and heavily occluded frames. No fairness assessment across demographic groups has been conducted.

Some infrastructure (BullMQ queue paths, parts of the socket event wiring) is present in the codebase but not fully connected to the active runtime flow. This creates a gap between what the architecture is designed for and what is currently live, which adds maintenance overhead.

VII. CONCLUSION

Crowdnexis AI shows that crowd risk assessment and missing-person detection can run in a single operator platform without exotic infrastructure. The components have solid empirical backing: YOLOv8 for person detection, FaceNet-style embeddings for identity matching, FAISS for scalable similarity search. The system packages them into a working workflow with persistent audit trails, a dashboard, and a multilingual chatbot. The architecture reflects real engineering trade-offs, not theoretical ideals, and the limitations are listed honestly. The evaluation framework defined here gives a concrete path to measuring how well the system performs once annotated test data is available.

VIII. FUTURE WORK

The processing path needs to move from direct subprocess calls to queue-driven workers, to enable distributed execution and improve scalability across multiple machines. That addresses the concurrent-job bottleneck without touching the API surface. It is also the prerequisite for anything resembling horizontal scaling.

Risk threshold calibration needs to be grounded in annotated, venue-specific data. Collecting ground-truth crowd counts and expert risk labels for a representative set of video segments would let you set thresholds by optimising classification F1 instead of by convention. Spatial density analysis, tracking density per frame region rather than total frame count, would produce far more actionable alerts than the current aggregate approach.

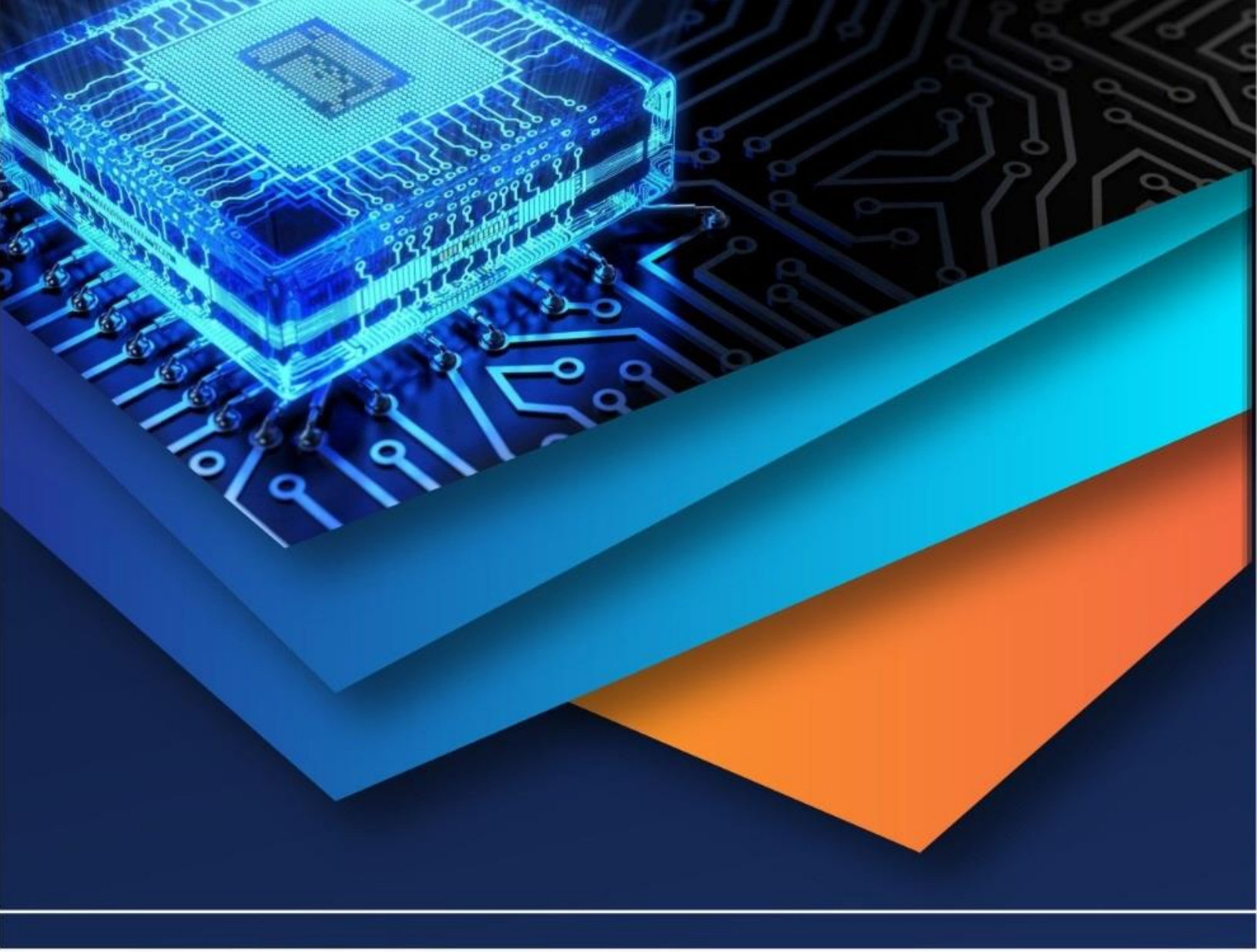
For face matching, temporal smoothing across consecutive frames would reduce false positives without touching the threshold. Treating a sequence of same-person detections as a single track rather than independent match events would make alerts more reliable. A calibration dataset covering different lighting conditions, camera angles, and demographic groups is needed before the system gets deployed in production.

On the infrastructure side, the incomplete socket wiring needs to be finished so that progress and alert events stream consistently from the backend to the frontend.

The current polling-heavy approach works but is not the right long-term answer. A lightweight model drift monitoring pipeline, even one based on embedding distribution statistics, would catch degrading recognition performance before it shows up as missed matches in the field.

REFERENCES

[1] L. Guo and J. Li, "Crowd counting method based on deep neural network," in Proc. 2022 Int. Conf. on Machine Learning, Control, and Robotics (MLCR), pp. 78–84, IEEE, 2022.
[2] G. C. Lee, Y. C. Lee, and C. C. Chiang, "Low-resolution face recognition in multi-person indoor environments using convolutional neural networks," in Proc. 2021 Int. Conf. on Computational Science and Computational Intelligence (CSCI), pp. 1629–1633, IEEE, 2021.
[3] L. Deng, Q. Zhou, S. Wang, J. M. Gorriz, and Y. Zhang, "Deep learning in crowd counting: A survey," CAAI Trans. on Intelligence Technology, vol. 2, no. 4, pp. 1043–1077, 2023.
[4] Y. Li, X. Zhang, and D. Chen, "CSRNet: Dilated convolutional neural networks for understanding the highly congested scenes," in Proc. IEEE CVPR, pp. 1091–1100, 2018.
[5] C. Zhang et al., "Cross-scene crowd counting via deep convolutional neural networks," in Proc. IEEE CVPR, pp. 833–841, 2016.
[6] F. Schroff, D. Kalenichenko, and J. Philbin, "FaceNet: A unified embedding for face recognition and clustering," in Proc. IEEE CVPR, pp. 815–823, 2015.
[7] D. Liang et al., "TransCrowd: Weakly-supervised crowd counting with transformers," Science China Information Sciences, vol. 65, no. 6, 2022.
[8] J. Redmon and A. Farhadi, "YOLOv3: An incremental improvement," arXiv:1804.02767, 2018.
[9] K. Zhang, Z. Zhang, Z. Li, and Y. Qiao, "Joint face detection and alignment using multitask cascaded convolutional networks," IEEE Signal Processing Letters, vol. 23, no. 10, pp. 1499–1503, 2016.
[10] P. Dandekar, M. Narwaria, and G. Bhatnagar, "Low-resolution face recognition: Review, challenges and research directions," Computers & Electrical Engineering, vol. 98, 2024.
[11] Tommy, R. Siregar, and E. R. Syahputra, "Low-resolution face image reconstruction using multi-stage FSRCNN," Int. Journal on Informatics Visualization, vol. 9, no. 3, pp. 1022–1032, 2025.
[12] M. M. Abid, R. B. Mustafa, and I. Ahmad, "Computationally intelligent real-time security surveillance using RetinaFace and FaceNet," Multimedia Tools and Applications, vol. 83, 2024.
[13] M. A. Khan, H. Sardar, and S. Zaidi, "A lightweight real-time CCTV surveillance framework for educational institutions using FaceNet," J. Advances in Information Technology, vol. 16, no. 8, 2025.
[14] [V. Lempitsky and A. Zisserman, "Learning to count objects in images," in Advances in NIPS, vol. 23, pp. 1324–1332, 2010.
[15] J. Johnson, M. Douze, and H. Jegou, "Billion-scale similarity search with GPUs," IEEE Trans. on Big Data, vol. 7, no. 3, pp. 535–547, 2021.



10.22214/IJRASET

45.98

IMPACT FACTOR: 7.129

IMPACT FACTOR: 7.429

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY