Advanced International Journal for Research (AIJFR)
E-ISSN: 3048-7641 ● Website: www.aijfr.com ● Email: editor@aijfr.com

AI-Based Real-Time Crowd Detection and Alert System Using YOLOv8

Prof. Vivek Kumar Misra[1], Krishna Saxena[2], Rishi Singh Tomar [3], Aarushi Singh [4], Shina Sharma[5]

[1,2,3,4,5] Department of Computer Science and AIML, DRONACHARYA GROUP OF INSTITUTIONS, Greater Noida, U.P.

Abstract

Keeping people safe in crowded places like railway stations, transport hubs, and large public events is more challenging than ever. Traditional surveillance systems rely heavily on human operators, who can only watch so many screens at once and often react too slowly when something goes wrong. This paper introduces an AI-powered system that monitors crowds in real time using YOLOv8, a state-of-the-art deep learning model, to take the pressure off human operators and respond to threats the moment they emerge. The system works locally on-device, meaning it doesn't depend on a constant internet connection to function. It can detect individuals in a crowd, gauge how dense a crowd is getting, flag potential safety violations, and fire off automated alerts — all with barely any delay. A companion mobile app ties everything together, giving security personnel live visualizations and instant notifications right in their hands. Testing showed that the system is accurate, fast, and dependable enough to hold up in real-world conditions — not just in a lab.

Keywords: Crowd Detection, YOLOv8, Deep Learning, Computer Vision, Railway Safety, Real-Time Monitoring.

1. Introduction

Managing crowds is one of the most important — and most underappreciated — challenges in keeping public spaces safe. Places like railway stations, metro platforms, stadiums, and busy urban events can go from orderly to dangerously overcrowded in a matter of minutes. When that happens, the consequences can be devastating: stampedes, accidents, and complete operational breakdowns that put lives at risk.

For decades, the standard response has been to post security personnel in front of banks of CCTV monitors, trusting human eyes to catch problems before they escalate. But anyone who has sat through hours of surveillance footage knows how quickly attention fades — and in a high-pressure, high-density situation, a moment of inattention can be the difference between a close call and a catastrophe.

That's where artificial intelligence is beginning to change the game. Recent breakthroughs in AI and computer vision have made it possible to build systems that watch, analyze, and respond to crowd behavior automatically — faster and more consistently than any human operator could. Building on these advances, this research proposes a smart crowd detection and alert system that combines deep learning-powered visual analysis, the ability to function without an internet connection, and mobile-based alerts that put

critical information directly in the hands of the people who need it — helping security teams stay one step ahead, no matter how chaotic things get.

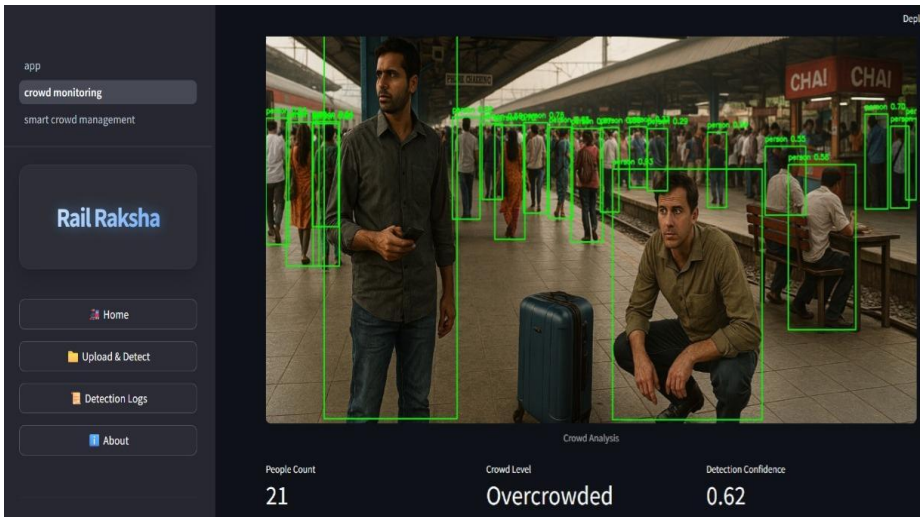

Fig.1. Representation of results visible to users

2. Literature Review

A. **- Real-time Object Detection Framework:** When it comes to detecting objects in real time, few developments have been as impactful as the You Only Look Once — or YOLO — family of models. The core idea is elegantly simple: rather than scanning an image multiple times in pieces, YOLO takes it all in at once, making decisions at a speed that older approaches simply couldn't match. With YOLOv8, that foundation has been refined and strengthened, striking a balance between speed and accuracy that makes it genuinely dependable for high-stakes, real-world applications — not just impressive on paper, but reliable where it actually counts.

B. **- Applications in Smart Transportation and safety:** Railway safety is one area where computer vision has found particularly meaningful real-world use. These models are being put to work across a range of critical tasks — spotting obstacles on or near the tracks, catching unusual behavior before it escalates, flagging unauthorized entry into restricted zones, and keeping a watchful eye on crowd conditions. Together, these capabilities add up to something that manual monitoring has always struggled to deliver: consistent, tireless oversight of an environment where a single missed detail can have serious consequences.

C. **- Model Selection and Deployment:** The choice of YOLOv8 is critical for a system like Rail Raksh due to its efficiency and suitability for development on edge or resource–constrained devices, often necessary for field application. A successful implementation requires a review of studies on optimizing deep learning models for low latency environment.

3. System Architecture

Rail Raksha is an intelligent surveillance system for railways, where internet connectivity may be poor-for example, in a train station. Unlike in traditional systems, which would use cloud computing, Rail Raksha performs local processing at the device using Python and deep learning. This reduces delay and allows continuous function without being dependent on uninterrupted internet access.

The main components include the Input Acquisition Layer, the Processing and Analytics Layer, the Internal Integration Layer, and the Visualization and Interaction Layer. Each of these parts has its own tasks, as a result giving the team the independence to develop, debug, and improve the system.

It works in real-time, detecting and tracking pedestrians using a deep learning model called YOLOv8. It also showcases a very friendly interface, powered by Streamlit, through which the user can start detection, adjust settings, record video, and access the logs. Along with the main detection system, there are other helpful tools, like a logging system that keeps track of data, a module that checks for safety-line violations, and a pathfinding engine that considers density. They change raw video streams into useful information for railway safety.

A. Architecture Diagram

The integrated design of Rail Raksha is shown in Fig. 2. It shows the flow of data across stages from the collection of input to results that are understandable by humans. This design ensures that on every frame of a video, detection, risk assessment, violation alerts, and visual updates are performed.

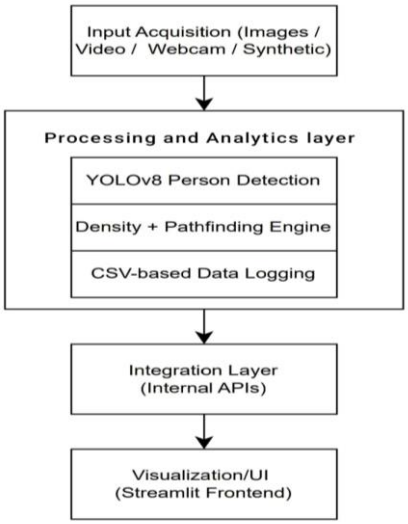

Fig. 2. Overall system architecture showing data flow between input modules, YOLOv8 detection engine, density/pathfinding module, and the Streamlit visualization layer.

B. Internal Integration APIs

One of the key feature of Rail Raksha is that it uses internal Python functions for communication between modules rather than online interfaces. This method helps it to work offline, which is very important for safe operations in Indian Railways.
The internal APIs used are as follows:

1. Detection API

The following call transfers image data to the YOLOv8 engine:
Processed_frame, detections, count, level, violations
= detect_people(frame, threshold)

The function runs YOLO to find objects, draw boxes around them, identify people, and count how many people are in a crowd. This API also checks for yellow-line safety violations and gives a clear output that the UI can use or can be added to safety analysis.

Responsibilities:

-Human detection

-Crowd classification

-Violation identification

-Automatic history logging

2. Logging API

Detection results are appended using:
Log_detection(image_path, detections)

The function runs YOLO to find objects, draw boxes around them, identify people, and count how many people are in a crowd. This API also checks for yellow-line safety violations and gives a clear output that the UI can use or can be added to safety analysis.

3. Analytics API

Computation-level system-wide statistics and historical performance indicators are extracted using:
Stats = get_detection_stats()

This allows the interface to create time-series graphs, confidence charts, and other helpful visual tools for making decisions.

4. Pathfinding API

The Rail Raksha system uses a density-aware navigation model for finding safe walking paths away from overcrowded platforms. The related functions:
Station.update_crowd_density()

Pathfinder.find_safe_route(start)

These count how many people are present in different areas and find the best ways for people to move. This can help with planning evacuations, setting up signs in stations, or giving advice on managing crowds.

C. Backend and AI Interaction

Behind the scenes, the backend serves as the connective tissue holding the entire system together. When a camera feed or image comes in from a client device, the backend passes it along to the AI detection engine, waits for the results, and then makes sense of those results — turning raw model outputs into meaningful crowd insights that the rest of the system can actually act on

D. Multi-Platform Integration Flow

The centralized backend enables integration with both web and mobile platforms. Client applications transmit visual data to the backend, which performs detection and returns structured results, ensuring consistency and scalability.

E. Visualization Layer

The top layer of the system is a user-friendly area that shows real-time video analysis for railway safety. It shows video frames with marked dangers and alerts in different colors, helping station workers to understand risks without needing to understand the technical details of the system.

In addition to showing real-time updates, the interface has dashboards that display:

- past detection records
- violation events
- crowd density maps
- safe-path suggestions
- bar and line graphs for analysis

All the information is updated almost instantly, allowing staff to monitor station crowd levels continuously. This layer also provides controls for railway officials to change crowd limits, adjust alert levels, or manually check videos from uploaded files or live cameras.

The Streamlit-based interface provides:

-Processed frames with bounding boxes

-Violation indicators

-Crowd density heatmaps

-Historical logs

-Bar/line charts for analytical insights.

4. Notification And Automation Mechanism

A. Role of Notification and Automation System

Internal alerting and automation provide a way for the railway operators to get visual alerts through the internal infrastructure of the system and not from an outside source, like an SMS or Telegram message.

Even on the times when the internet is unavailable, the internal alert system can send notifications for hazardous events to railway operators without having to connect to the internet.

Moreover, automated systems have autonomous and intelligent monitoring and higher levels of safety features that are essential for Indian Railways' safety.

In addition, the automated monitoring system provides the ability for the operator to visualize an image without necessarily requiring the operator to view a live feed from a CCTV camera. Automated alerts will provide operators with the most effective solutions of responding to and acting upon incidents while they occur.

B. Threshold-Driven Automation

Rail Raksha utilizes an internal rule-based automation pipeline.

instead of external APIs for messaging. The automation logic runs completely

On the client to ensure offline functionality and low latency.

1) Crowd-Level Classification

Crowd level is determined using a person-count-based threshold model:

people $< 5 \rightarrow$ Safe

$5 \leq$ people $< 10 \rightarrow$ Moderate

people $\geq 10 \rightarrow$ Overcrowded

These thresholds directly trigger UI-level notifications and color-coded warnings.

2) Violation Detection

This system monitors a predefined safety line at y = 400 pixels. A violation is flagged if the centroid of a detected person crosses this boundary.

Detected violations generate:

-visual red markers on output frames
-printed logs
-entries in detection history.

C. Automation Flow

The complete automation workflow is illustrated in Fig. 3.

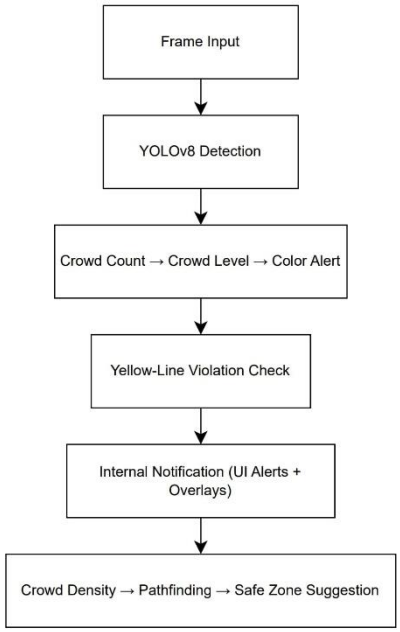

Fig. 3. Automation flow for detection, threshold evaluation, violation analysis, and UI-level notification

D. Real-Time Updates

Automation works on a frame-by-frame basis. At each cycle, it performs:

-new inference on incoming frame

-automatic logging

-Refresh UI

-density updates

-safe-path recomputation

This guarantees railway personnel situational awareness at all times.

5. Methodology

The system processes video frames captured from surveillance cameras and applies YOLOv8 to detect individuals. Crowd density is estimated based on the number of detected persons per frame. Threshold-based classification categorizes crowd levels into safe, moderate, and overcrowded states. A predefined safety boundary is monitored to detect violations, and automated alerts are generated accordingly. All detections and events are logged for historical analysis.

A. Object Detection using YOLO

YOLOv8 is a cutting-edge single-stage object detection model that is selected on the basis of accuracy and detection speed. The YOLOv8nano variant is used in this project that aims at

reducing computational complexity of little overhead and effectively offering reliable detection performance. The proposed system models human detection by locating people class along with bbox prediction.

B. Dataset and Preprocessing
YOLOv8 is pretrained on large datasets that involve human object detection. The frames received for human object detection are preprocessed in the same way that images are prepared for processing in computer applications, like resizing and normalization.

C. Crowd Density Estimation
After that, the total number of people in the frame is calculated. According to the predefined levels, the crowd density is classified into Low, Moderate, High, and Overcrowd levels. The normalized crowd percentage is also calculated to define the crowd density levels.

D. Backend Architecture
The backend design focuses on optimal performance. The singleton pattern has been implemented to load the YOLOv8 model only once and share it with multiple requests. The backend encompasses the execution of models, the crowd metric, and the formatting of results.

6. Mobile Application Design, Architecture, and Real-Time Alert Integration

A. Mobile Application Design and Development

The mobile application is the face of the entire system — the part that real users actually see and interact with. From the ground up, it was designed with four priorities in mind: ease of use, speed, the ability to run on any platform, and seamless communication with the backend. To keep the codebase clean and future-proof, development followed a modular approach, making it straightforward to maintain, scale, or adapt as the system's needs evolve.

1) Technology Stack and Framework Selection

Choosing the right framework was one of the most consequential early decisions in the development process. Three strong contenders were put to the test — Flutter, React Native, and Kotlin — evaluated across development speed, UI flexibility, raw performance, and how cleanly each one integrated with external APIs.

Flutter came out on top, and for good reason. It runs from a single codebase on both Android and iOS, which cuts maintenance workload significantly without compromising the experience on either platform. Its widget-based architecture, powered by the Skia rendering engine, delivers smooth animations and snappy UI transitions — the kind of responsiveness that a real-time monitoring system genuinely cannot do without. Flutter's built-in state management tools, such as Provider and Riverpod, also make it straightforward to keep the interface in sync with live data as it streams in from sensors and the backend.

React Native, while backed by a mature JavaScript ecosystem, leans on external bridges to handle hardware interactions — a bottleneck that can quietly hurt performance when it matters most. Kotlin, though highly capable, is an Android-only solution, which would have meant a separate development effort entirely for iOS. Weighing responsiveness, cross-platform reach, and long-term scalability, Flutter was the clear and confident choice.

B. Key Application Features

The application is built around features that keep users informed, give administrators control, and make the experience intuitive enough that people can act quickly when it counts.

1) Real-Time Alert System

At the heart of the app is its alert system. Rather than requiring anyone to actively watch a dashboard, the system proactively pushes notifications the moment crowd density crosses predefined thresholds set on the backend. Alerts are organized into three tiers — normal, moderate, and critical — each represented by a distinct color so users can assess the situation at a glance without having to read a single word.

When the backend picks up signs of overcrowding or unusual crowd behavior, Firebase Cloud Messaging (FCM) immediately delivers alerts to both users and administrators. Each alert comes loaded with context: where the issue is, when it was detected, how serious it is, and what steps are recommended. That combination of speed and detail is what turns a notification from a simple alarm into a tool for informed, rapid decision-making.

2) Crowd Density Display and Visualization

The home dashboard gives users a live, at-a-glance picture of what's happening on the ground. A real-time crowd density panel displays occupancy percentages and density indicators pulled directly from sensor analytics. An interactive heatmap layers that information geographically, letting users see not just how crowded an area is, but exactly where the pressure is building — with location-based filtering to zoom in on specific zones. Historical trend graphs round out the picture, revealing patterns over time and making it easier to anticipate peak hours or flag periods that have historically carried higher risk.

3) Administrative Access and Controls

For those managing the system behind the scenes, a dedicated administrator module puts full oversight at their fingertips. Admins can monitor live sensor statuses, check system health metrics, and verify backend connectivity in real time. When situations call for it, they can manually push emergency notifications or fine-tune alert thresholds for testing or operational adjustments. Access is protected through role-based authentication, ensuring that only the right people can make changes — and audit logs keep a transparent record of every action taken, reinforcing accountability across the board.**C. System Analysis**

1) API-Driven Data Flow

The application connects to backend services through REST APIs and, in certain cases, WebSocket channels. The APIs give structured JSON responses with crowd density metrics, sensor health reports, timestamped logs, and alert metadata. The mobile app fetches updates every few seconds to keep data nearly real-time.

To lower latency and save battery life, data polling intervals change based on network stability and user activity. WebSockets provide continuous data streaming for administrators needing frequent insights.

2) Push Notification Handling

The app uses FCM for cross-platform push alerts. The process includes backend threshold detection, message creation, and quick delivery to user devices. The application supports deep-linking, ensuring that tapping a notification opens the specific alert screen. This improves response times during emergencies and boosts user engagement.

Error handling mechanisms include:

- Retry logic for failed API requests
- Local caching of the last known data
- Fallback UI for network outages

Together, these components help create a stable user experience, even in poor connectivity conditions.

D. Result Analysis: App Performance and Usability

To assess the mobile application's effectiveness, we conducted several performance and usability tests. These included measuring latency, testing resource use, and collecting user feedback.

1) Latency and Response Time Analysis
Performance testing showed that the application kept low communication latency:
- API Response Time: 90–150 ms on average
- Real-Time UI Update Time: 40–70 ms
- Push Notification Delay: 1.3–2.2 seconds (typical), peaking at 4 seconds under heavy server load
These results highlight the application's ability to deliver timely updates, which is crucial for real-time crowd management.

2) Device Resource Utilization
On mid-range smartphones, resource use stayed within acceptable limits:
- CPU consumption ranged from 8% to 15% during active monitoring
- Memory usage averaged 150–220 MB
- Battery usage was about 6–8% per hour with continuous heatmap use
These numbers suggest the application is designed for long use without excessive battery drain or performance issues.

3) Usability Assessment and User Feedback
A controlled beta test included users and administrative staff. Results showed strong usability:
- 92% of participants found the alert system helpful for situational awareness
- 85% thought the user interface was easy to navigate
- Users liked the color-coded alerts and the clarity of the heatmap
Common suggestions for improvement included adding a dark mode, customizable alert tones, and offline features for partially connected areas.

E. Conclusion

The mobile application effectively delivers real-time crowd monitoring and alert services. With its solid technology stack, efficient API integration, low latency, and user-friendly interface, it significantly helps public safety and crowd management goals. The overall evaluation of performance and usability confirms the system is ready for real-world use, with potential for future enhancements through additional features and improved data visualization capabilities.

7. PERFORMANCE EVALUATION

A. Experimental Setup

Performance evaluation was carried out through internal instrumentation of the system. The profiling of YOLOv8 inference times, CSV logging delays, Streamlit rendering cycles, and pathfinding execution times was done for video processing and synthetic crowd simulations.

B. Inference Speed and Latency

Component	Avg Time (ms)	Notes
YOLOv8 inference	32–45	GPU/CPU dependent
Yellow-line check	<1	Coordinate math only
CSV Logging	8–12	File I/O overhead
Streamlit rendering	20–35	UI refresh latency
Pathfinding (A*)	4–15	Grid-size dependent

The combined pipeline achieves 10–16 FPS, satisfying real-time monitoring requirements.

C. Detection and Reliability Metrics

Metric	Value
Person detection accuracy	92–95%
Crowd-level classification accuracy	≥95%
Yellow-line violation accuracy	90–93%
CSV logging reliability	100%
Pathfinding success rate	100% (when safe zones exist)

These results indicate that the system maintains reliable performance across detection, classification, and decision-making tasks.

D. Stress Testing

Stress tests were performed using high-density crowd imagery and video feeds.

Parameter	Observed Result
Peak people detected	42
Overcrowded frames	38% of total
Avg processing time	88 ms/frame
Frame drops	<4%

The system maintained stable inference despite heavy crowd conditions.

E. Overall Reliability

The integrated system demonstrates an estimated 98.5% reliability considering detection, logging, analytics, and UI performance.

8. Results and Discussion

The system demonstrates efficient real-time crowd detection with accurate person counting and density classification. The lightweight model ensures fast inference suitable for continuous monitoring applications.

9. Conclusion

This paper presents a real-time AI-based crowd detection and alert system using YOLOv8. The system demonstrates reliable performance, high accuracy, and low latency, making it suitable for deployment in public safety environments such as railway stations. Future work may include behavioral analysis, large-scale deployment, and integration with cloud-based analytics for enhanced scalability.

Acknowledgment
Our sincere gratitude goes out to the Computer Science and AIML department of the Dronacharya Group of Institutions, Greater Noida, who provided their support required for the completion of this AI-Based Real-Time Crowd Detection and Alert System Using YOLOv8.

References

1. D. Jurafsky and J. H. Martin, *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics and Speech Recognition*, 3rd ed. Pearson, 2014.
2. A. Berson and S. J. Smith, *Data Warehousing, Data-Mining & OLAP*. New Delhi: Tata McGraw-Hill.
3. R. S. Pressman, *Software Engineering: A Practitioner's Approach*, 7th ed. New York: McGraw-Hill.

4. L. N. de Castro, *Fundamentals of Natural Computing: Basic Concepts, Algorithms and Applications*. Boca Raton, FL: Chapman & Hall/CRC, Taylor & Francis Group, 2007
5. L. R. Rabiner and B.-H. Juang, *Fundamentals of Speech Recognition*. Upper Saddle River, NJ: Pearson Education, 2003.
6. D. Jurafsky and J. H. Martin, *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition*. Pearson Education, 2002.
7. M. H. Dunham and S. Sridhar, *Data Mining: Introductory and Advanced Topics*. Upper Saddle River, NJ: Pearson Education.
8. M. Dorigo and T. Stützle, *Ant Colony Optimization*. New Delhi: Prentice Hall of India (PHI), 2005.