



# International Journal of Research Publication and Reviews

Journal Homepage: [www.ijrpr.com](http://www.ijrpr.com) ISSN 2582-7421

## Smart CCTV Monitoring and Automated Emergency Alert Systems

**I. A. Jannathul Firthous<sup>a</sup>, Kannika Devi P<sup>b\*</sup>**

<sup>a,b</sup> Department of Information Technology, Sri Shakthi Institute of Engineering and Technology (An Autonomous Institution), Coimbatore, Tamil Nadu, India

### ABSTRACT

Traditional closed-circuit television (CCTV) systems depend on human operators to watch live feeds or review footage after an event, which is difficult when many cameras operate at once. This paper presents an AI-based smart CCTV surveillance prototype that analyzes video to help identify potential violence, dangerous objects, and suspicious activity. OpenCV reads video streams and prepares frames, a YOLO detector identifies people and selected weapon classes, a tracking algorithm associates detections across frames, and a CNN-LSTM action-recognition model is proposed for violence/non-violence classification. A configurable decision layer flags potential incidents, after which the application saves evidence frames or clips and stores incident metadata in a MySQL database. A Flask backend and web dashboard let authorized personnel review incidents, evidence, timestamps and alert status, while e-mail notifications draw attention to important events sooner. The system is designed as a decision-support tool, not an autonomous authority: AI predictions may be wrong, and human review is required before a flagged event is treated as confirmed misconduct. Performance is to be assessed on a held-out dataset using precision, recall, mAP, F1-score and inference speed.

**Keywords:** Smart CCTV; Computer Vision; YOLO; Object Detection; Action Recognition; CNN-LSTM; Person Tracking; Incident Reporting; MySQL; Flask; Automated Alerts

### 1. Introduction

Closed-circuit television is an important part of security infrastructure in educational institutions, offices, commercial buildings, transport facilities and public areas. Cameras capture continuous footage, but its value depends on whether an operator notices an incident or can locate it later. Monitoring many feeds simultaneously is demanding, and reviewing long recordings after an event takes considerable time.

Advances in computer vision and deep learning make automatic video analysis possible. Object-detection models identify visual categories in single frames, tracking algorithms associate detections over time, and action-recognition models classify activity across sequences of frames. Combining these components can help flag events that deserve attention. This paper proposes an AI-based smart CCTV prototype that processes video, detects selected objects and people, analyzes activity patterns, captures evidence for flagged events, and presents incident information through a web dashboard. The system supports security personnel; it does not replace their judgment.

#### Nomenclature

|      |                                   |
|------|-----------------------------------|
| AI   | Artificial Intelligence           |
| API  | Application Programming Interface |
| CCTV | Closed-Circuit Television         |
| CNN  | Convolutional Neural Network      |
| FPS  | Frames Per Second                 |
| LSTM | Long Short-Term Memory            |
| mAP  | Mean Average Precision            |
| ML   | Machine Learning                  |
| RTSP | Real-Time Streaming Protocol      |

|      |                               |
|------|-------------------------------|
| SMTP | Simple Mail Transfer Protocol |
| UI   | User Interface                |
| YOLO | You Only Look Once            |

### **1.1 Need for the system and problem statement**

A surveillance system may record an incident without drawing attention to it while it happens. Operators may be responsible for several feeds, and the volume of stored footage makes manual review slow. An automated pipeline can prioritize segments that match defined detection criteria. Staff also need the timestamp, camera identifier, evidence clip and alert history of each event; holding this in a structured database and a single dashboard makes review and follow-up more consistent.

The problem addressed here is the limited ability of conventional CCTV workflows to interpret video automatically and organize potential incidents for timely review. The prototype detects selected objects and activity patterns, flags possible incidents, preserves related evidence and notifies authorized users. It cannot establish criminal intent, and an AI prediction must not be treated as conclusive proof.

### **1.2 and scope**

The objectives of the project are to:

- develop a pipeline that reads live or recorded CCTV footage;
- use a YOLO-based detector to identify people and selected potentially dangerous objects;
- explore action recognition over consecutive frames to classify violence and non-violence;
- capture frames or clips and associate them with incident metadata;
- store incident records in MySQL and present them through a Flask-based dashboard; and
- send e-mail notifications for configured incidents and evaluate model performance using measured metrics.

The scope is a prototype focused on person/object detection and violence-related activity classification, with optional rules for predefined suspicious events. Performance depends on the dataset, camera viewpoint, lighting, hardware and model quality. The system does not confirm crimes or guarantee detection of every dangerous event, and any deployment involving real people must follow applicable law, institutional policy and data-protection requirements.

## **2. Literature review**

### **2.1 Intelligent CCTV surveillance and existing systems**

Intelligent surveillance research derives information from video beyond recording and playback. Common tasks are object detection, multi-object tracking, activity recognition and anomaly detection. They answer different questions: detection identifies visible categories, tracking estimates which detections belong to the same moving entity, and action recognition classifies patterns across time. A practical application must consider the whole pipeline: camera input, inference, event logic, evidence handling, notification and evaluation. A strong model in isolation does not automatically give a reliable application.

Traditional CCTV records footage for live viewing or investigation. Motion-detection systems trigger on scene changes, intrusion systems monitor predefined regions, and video-analytics products may combine detection and tracking, although available features vary by product. The proposed work is therefore described as an integration-oriented prototype rather than the first application of AI to CCTV.

### **2.2 Object detection and YOLO**

Object detection combines classification with localization: the model predicts categories and bounding boxes. Evaluation commonly uses precision, recall and mean average precision (mAP), influenced by class balance, annotation quality, object size and occlusion. Small or partly hidden objects are hard to detect in surveillance footage, and a model trained on general images may perform poorly on low-resolution CCTV frames without domain-relevant examples.

YOLO is a family of detectors designed to predict object locations and classes efficiently. A pretrained checkpoint supplies general visual features, and fine-tuning adapts it to a labelled dataset with the required classes. A lightweight variant such as YOLO11n (Ultralytics, n.d.) is evaluated here as a starting point for real-time processing; the final variant should be chosen from measured accuracy and speed on the available hardware.

### 2.3 Violence detection and CNN-LSTM action recognition

Violence is expressed through posture, movement and interaction over time, so a single frame can be ambiguous. Sequence-based classifiers are trained and evaluated on video datasets labelled violence or non-violence, and dataset composition and recording conditions affect generalization. RWF-2000 is a surveillance-video dataset designed for violence detection (Cheng, Cai, & Li, 2021), whereas UCF-Crime is a broader anomaly-detection dataset covering several categories of unusual events (Sultani, Chen, & Shah, 2018). General-purpose video collections such as Kinetics (Kay et al., 2017) and architectures such as I3D (Carreira & Zisserman, 2017) are also widely used in action recognition. Dataset licenses and usage conditions should be checked before use.

A CNN-LSTM architecture is one possible approach (Donahue et al., 2015, 2017). A convolutional network extracts spatial features from each sampled frame; the ordered feature vectors are passed to a long short-term memory network that learns patterns across the sequence, and a classification layer estimates the activity class. The approach needs labelled sequences and careful choices of frame sampling, sequence length, class balance and decision thresholds, and it should not be described as implemented until it has been trained and tested.

### 2.4 Research gap and motivation

The motivation is a practical prototype that links video analysis to incident handling. Its intended contribution is the integration of object detection, optional tracking, action recognition, evidence capture, structured storage and notification in one workflow, assessed with both model metrics and application-level tests such as false alerts, latency and usability. Table 1 summarizes the areas reviewed and their relevance to the system.

**Table 1 - Summary of related work.**

| Area               | Key idea                                                                                      | Relevance to this project                                                |
|--------------------|-----------------------------------------------------------------------------------------------|--------------------------------------------------------------------------|
| Intelligent CCTV   | Detection, tracking, activity recognition and anomaly detection derive information from video | Defines the complete pipeline followed by the prototype                  |
| Existing systems   | Recording, motion/intrusion detection, analytics; features vary by product                    | Motivates integrating AI with incident storage, evidence and a dashboard |
| Object detection   | Predicts categories and boxes; judged by precision, recall, mAP                               | Finds people and selected dangerous objects                              |
| YOLO detection     | Efficient detection; pretrained checkpoints fine-tuned on labelled data                       | YOLO11n evaluated for real-time use                                      |
| Violence detection | Activity unfolds over time; RWF-2000 and UCF-Crime available                                  | Labelled video for violence/non-violence classification                  |
| CNN-LSTM           | CNN frame features and LSTM temporal modelling                                                | Candidate sequence-based action recogniser                               |

## 3. Methodology

### 3.1 Research design and workflow

The project follows an applied prototype-development and evaluation design divided into video acquisition, data preparation, model configuration, inference pipeline development, incident management, interface development and testing. Each component is tested separately before integration. Evaluation uses a held-out test set that is not used for tuning, and partitions are created at video level, preferably at source-camera or recording-session level where related footage could otherwise fall into more than one split. The workflow is:

collect or select appropriately licensed CCTV-like videos and object images, define target classes and annotate bounding boxes; prepare training, validation and test partitions, and fine-tune a pretrained YOLO checkpoint; prepare labelled video sequences and train or fine-tune a CNN-LSTM model if sufficient data and compute are available; build a continuous pipeline that runs inference and tracks detections, combined with configurable decision rules; save evidence and metadata for flagged events and connect the pipeline to MySQL, Flask, the dashboard and e-mail; and evaluate detection quality, action-classification quality, latency and end-to-end reliability.

### 3.2 Video acquisition and dataset preparation

Video is obtained from a local file, a webcam or a compatible network camera stream. OpenCV reads frames, verifies that capture succeeded, and resizes them to the model input size using the same preprocessing as in training (OpenCV, n.d.). For network cameras, interruptions, reconnection, buffering and timestamp handling must be considered. Real-time operation is an engineering target, not an automatic result of reading frames in a loop; the measured processing rate must be compared with the camera input rate.

The dataset must reflect the intended environment. For detection, images should contain positive examples of every target class and realistic negative scenes, varied in distance, lighting, camera angle, resolution and occlusion. Each target object needs a bounding box and class identifier, and annotations are checked for missing boxes, wrong classes, inconsistent boundaries, duplicates and corrupt files. For action recognition, clips are sampled into fixed-length frame sequences with consistent resizing and normalization, and clip-level labels must match the clip content. Table 2 summarizes the data plan.

**Table 2 - Dataset and annotation plan.**

| Data type        | Required content and labels                                                                           | Purpose                              |
|------------------|-------------------------------------------------------------------------------------------------------|--------------------------------------|
| Detection images | Positive examples of every class plus negative scenes, varied in distance, light, angle and occlusion | Train and evaluate YOLO detector     |
| Box annotations  | Bounding box and class ID per object; names mapped in dataset configuration                           | Supervise detection; audit quality   |
| Action videos    | Violence/non-violence clips (e.g., RWF-2000) sampled to fixed-length sequences                        | Train and evaluate CNN-LSTM          |
| Partitions       | Train, validation and held-out test sets at video or camera/session level                             | Prevent leakage; unbiased evaluation |

### 3.3 Object detection using YOLO

The detector starts from a pretrained YOLO checkpoint and is fine-tuned on a custom dataset. The training configuration specifies the dataset path, class names, epochs, input size, batch size and hardware. The best checkpoint is chosen by validation performance rather than training loss. After training, the model is evaluated on the held-out set using precision, recall, mAP@0.5, mAP@0.5:0.95 and inference speed, with per-class results and failure examples examined for safety-relevant objects.

### 3.4 Action recognition using CNN-LSTM

The action-recognition module samples a sequence of frames from each clip. A CNN backbone converts each frame into a feature representation, the ordered sequence is passed to an LSTM, and a classification layer outputs a violence/non-violence score or another set of trained classes. The training procedure defines sequence length, sampling rate, backbone, LSTM hidden size, optimizer, learning rate, batch size and epochs. Evaluation uses a confusion matrix, accuracy, precision, recall and F1-score.

### 3.5 Person tracking

A tracking algorithm such as ByteTrack or DeepSORT associates detections across frames. Each track receives a temporary identifier that helps estimate trajectories and avoid repeated event records. Tracking is imperfect when people overlap, leave the scene or are occluded, and track identifiers must not be treated as real-world identities or as facial recognition.

### 3.6 Incident decision and alert logic

The decision layer combines model outputs according to explicit rules. For example, an action-recognition score may have to exceed a configurable threshold for several consecutive windows before an event is flagged. Object detections add context, but the presence of an object alone does not prove harmful intent. The rules are tuned on validation data and documented, and a cooldown or incident-merging rule prevents a new record for every frame of the same event. A flagged incident is a request for review, not a conclusion that a crime occurred.

### 3.7 Ethical considerations and limitations

Surveillance footage may contain identifiable people and sensitive activity. Data should be collected and processed only with authorization, access to the dashboard and evidence restricted, credentials protected, retention periods defined, and stored footage minimized to what the stated purpose needs. The system can produce false predictions and may perform unevenly across camera conditions, so human review is required before consequential action.

The prototype may be limited by the amount and diversity of labelled data, available compute, camera quality, occlusion and differences between training and deployment environments. Violence is context-dependent, and action recognition may confuse sport, play, sudden movement or harmless contact with violence. These limitations are documented during testing.

## 5. System analysis and design

### 5.1 Existing system and its limitations

Conventional workflows use camera recording, live monitoring, motion alerts and manual review; more advanced products add object detection, tracking, intrusion analytics or activity analysis, depending on the camera and software. Their main limitations are that:

Continuous monitoring is difficult when many feeds are active, and searching long recordings manually takes substantial time;

Motion alone does not distinguish harmless activity from a security incident, and object detection alone does not explain activity over time;

Incident notes, evidence and alerts are often kept in separate workflows; and

Model predictions can be wrong, so automated systems require review and monitoring.

### 5.2 Proposed system and architecture

The proposed prototype links video processing, object detection, tracking, action recognition, decision logic, evidence capture, a database and a dashboard. It flags potential events and organizes the relevant information so that an authorized operator can inspect the evidence and decide what action is appropriate. As shown in Fig. 1, the architecture is organized into layers. The input layer receives video; the computer-vision layer uses OpenCV to extract frames; the AI layer runs object detection and action recognition; the tracking and decision layer associates detections and decides whether to flag an event; the evidence layer saves frames or clips; the application layer exposes data through Flask; MySQL stores structured records; and the frontend presents them to authorized users.

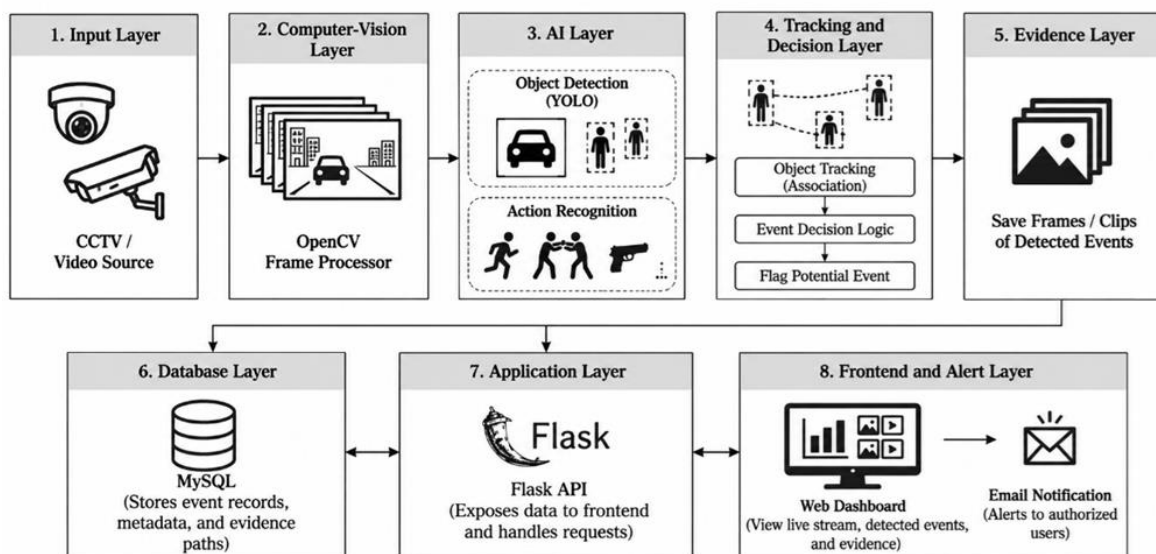


Fig. 1 - Overall system architecture.

### 5.3 System requirements

The hardware requirements are a computer with a modern multi-core CPU; at least 8 GB RAM for a small prototype (more for training); a supported GPU for training or faster inference; a webcam, video files or authorized access to a compatible CCTV/RTSP stream; and storage for datasets, checkpoints, evidence clips and database backups. The software requirements are a Python environment with compatible OpenCV and Ultralytics packages, PyTorch (PyTorch, n.d.), Flask (Pallets Projects, n.d.), a MySQL server with a Python connector (Oracle, n.d.), a modern web browser and Git.

Table 3 lists the functional requirements. The non-functional requirements are: latency measured on target hardware; stream, database and e-mail failures handled without crashing; credentials protected and evidence access restricted to authorized users; clear timestamps, event type, status and evidence links on dashboard pages; modular detection, tracking, recognition, database and interface code; and collection and retention of only authorized data needed for the project purpose.

**Table 3 - Functional requirements.**

| ID    | Requirement                                                   |
|-------|---------------------------------------------------------------|
| FR-01 | Accept video from a supported source.                         |
| FR-02 | Process frames and run configured object detection.           |
| FR-03 | Optionally track detections across frames.                    |
| FR-04 | Run the configured action-recognition model when available.   |
| FR-05 | Apply documented decision rules to flag potential incidents.  |
| FR-06 | Save evidence and incident metadata for flagged events.       |
| FR-07 | Store incident records in MySQL.                              |
| FR-08 | Let authorized users view incident records and evidence.      |
| FR-09 | Send e-mail notifications when configured conditions are met. |
| FR-10 | Log errors and processing status.                             |

### 5.4 Database design and user interface

A relational database organizes users, cameras, incidents, evidence and alerts, with primary keys identifying records and foreign keys connecting related data (Table 4). Evidence file paths or object-storage references are stored rather than large binary videos in ordinary incident rows. Passwords must be stored as secure hashes, not plaintext (OWASP Foundation, n.d.), and camera stream URLs, which may contain credentials, should be kept out of source control and logs.

**Table 4 - Database tables.**

| Table     | Example fields                                                                  |
|-----------|---------------------------------------------------------------------------------|
| Users     | user_id, username, password_hash, role, created_at                              |
| Cameras   | camera_id, camera_name, source_uri, location_label, status                      |
| Incidents | incident_id, camera_id, event_type, confidence, timestamp, status, review_notes |
| Evidence  | evidence_id, incident_id, file_path, media_type, created_at                     |
| Alerts    | alert_id, incident_id, channel, recipient, sent_at, delivery_status             |

The dashboard gives a concise overview of recent potential incidents with timestamps, camera labels, event categories, review status and links to evidence. A detail view shows the saved frame or clip, the model score and operator review controls, and filters by date, camera, category and status help locate records. The interface labels AI predictions as unverified until reviewed. The operator, not the AI, verifies an incident: the operator opens the dashboard, filters history, opens evidence and changes incident status, while the system performs processing, inference, incident creation, storage and notification.

---

## 6. Implementation

### 6.1 Development environment and video processing

The modules are developed separately and tested locally before integration. VS Code is used for application development, and Google Colab or a local GPU environment for experiments. A Python virtual environment pins dependency versions, and Git tracks source code, configuration examples and documentation, with large datasets and trained models managed separately according to repository limits and licensing.

The video module opens the selected source and reads frames in a loop, checking the return status of each read, handling end-of-file or stream failure, and forwarding valid frames to inference. Frames may be resized for the model while the original frame is retained for evidence capture, and the module releases its resources cleanly on shutdown.

### 6.2 YOLO model configuration and training

The detector is initialized from the pretrained YOLO11n checkpoint and trained with a dataset configuration (data.yaml) that defines the training and validation images and class names. The weapon-detection dataset, obtained programmatically from a public Kaggle repository, defines nine classes: automatic rifle, bazooka, grenade launcher, handgun, knife, shotgun, SMG, sniper and sword. Before training, annotation files are backed up and audited: each label file name is cross-referenced with the expected class identifier and mismatched YOLO-format class indices are corrected automatically. Bounding boxes are then visualized to confirm that the normalized coordinates align with objects in pixel space. Fine-tuning runs for 50 epochs at an image size of 640 with a batch size of 4, and the best checkpoint is saved for deployment.

At inference time the detector runs on a camera feed or test frame with a confidence threshold of 0.20. For every detection it reports the class and confidence, draws annotated bounding boxes, and raises a security alert when a class in the threat list (knife, handgun, automatic rifle, shotgun, SMG, sniper, sword, bazooka) is present.

### 6.3 Action recognition and person tracking

For the CNN-LSTM implementation, each clip is converted to a fixed-length sequence of sampled frames. A CNN backbone produces a feature vector per frame, the sequence of vectors is fed to an LSTM, and a final dense layer predicts the activity label. Preprocessing at inference must match training. During live inference a sliding-window buffer holds the latest frames; when enough frames are available the model predicts the current window, and overlapping windows let the application update predictions without waiting for the video to end. Sequence length, sampling interval and stride are validated for latency and accuracy.

The tracking module receives detections and assigns track identifiers across frames, which helps associate movement and merge repeated detections into one incident. Tracking is reset or recovered gracefully after stream interruptions.

### 6.4 Real-time incident detection and evidence capture

The video reader, detector, tracker and action-recognition window run as a continuous pipeline (Fig. 2). The decision layer uses validated thresholds, persistence rules and cooldown intervals; when a potential event meets the configured conditions, the event manager creates an incident record and requests evidence capture. Real-time performance is measured as end-to-end latency and sustained frames per second. If inference takes longer than the camera frame interval, the system can process selected frames, reduce the input size, use a lighter model, or separate capture and inference into asynchronous stages, avoiding unbounded queues that would delay alerts.

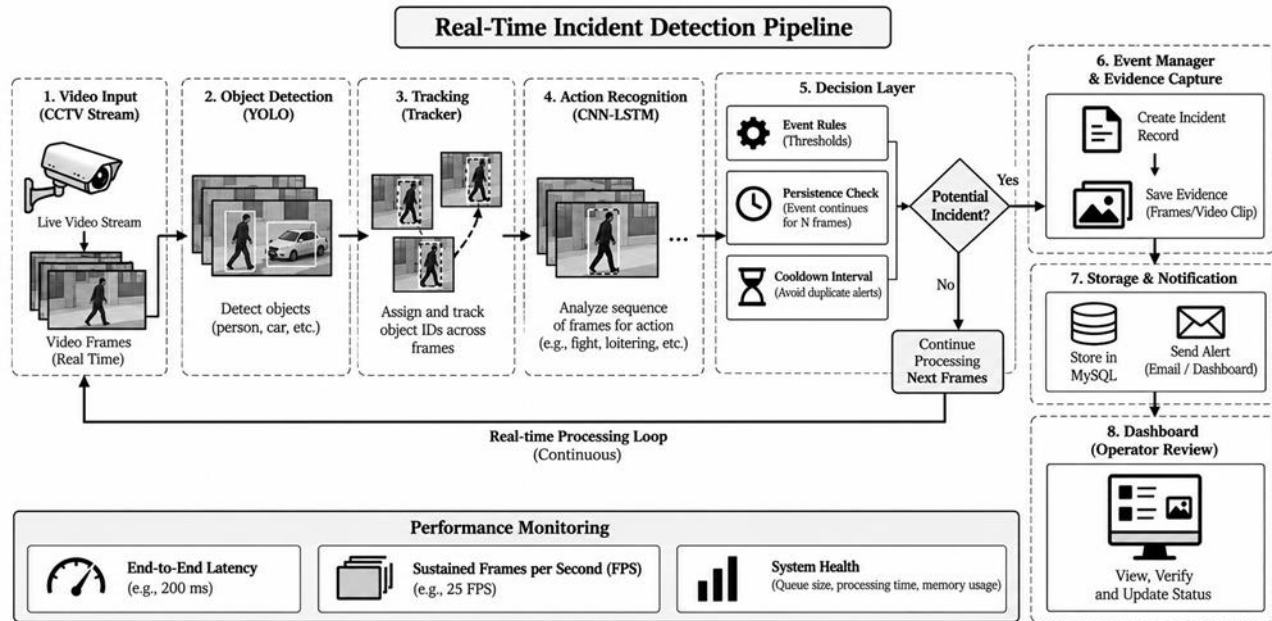


Fig. 2 - Real-time video-processing pipeline.

Evidence capture saves the relevant frame or a short segment around the flagged event together with the timestamp, camera identifier, predicted event type, model score and evidence path. Evidence files use unique names and a defined retention policy, and the system verifies that file writes succeed before reporting that evidence has been saved.

#### 6.5 Database, backend, dashboard and e-mail alerts

The application uses a database connector to create and query records, with parameterized queries to avoid SQL injection and handling of connection failures and transactions. Incident and evidence records are linked by identifiers, and repeated predictions for one event are merged where appropriate. Flask provides routes for dashboard data, incident details, evidence references and review-status updates; inference code is kept separate from request handlers so that a slow model operation does not block unrelated requests, and authentication protects all incident and evidence endpoints.

The frontend uses HTML for structure, CSS for layout and JavaScript for interaction and periodic refresh. It shows the most important fields first, distinguishes a new AI flag from a human-reviewed event, and handles error states, empty results and inaccessible evidence links. E-mail notifications are sent through SMTP when a configured rule triggers; the message carries the event category, timestamp, camera label and a secure reference to the dashboard record, sensitive video is not attached unless policy permits, and credentials are kept in environment variables or a secret manager.

#### 6.6 Integration of components

Integration begins with the video module and detector, then adds tracking, action recognition, decision logic, evidence storage, database operations, the Flask API, the dashboard and e-mail. Each stage has clear inputs and outputs and logs its errors. End-to-end tests confirm that a test event produces one incident record, the expected evidence, a visible dashboard entry and an alert when configured.

## 7. Testing and performance evaluation

### 7.1 Testing methodology

Testing covers software correctness, model performance, integration and real-time behavior. Each test records the input, expected behavior, actual behavior, pass/fail result and notes. Unit tests verify that the video input opens and returns valid frames, model output is parsed into class, confidence and bounding-box format, incident rules respond correctly to scores above and below the threshold, evidence files are created and their paths stored, database create/read/update operations work, and e-mail failures are logged without crashing video processing. Integration tests run a sample video through the whole pipeline and confirm that detections appear, an incident is created only when the test conditions are met, evidence is saved, a database row is created, the dashboard shows the record and an e-mail is sent when enabled, including cases where no incident should be created. Table 5 lists the planned test cases.

**Table 5 - Test cases.**

| ID    | Scenario                          | Expected behavior                        |
|-------|-----------------------------------|------------------------------------------|
| TC-01 | Valid video file                  | Frames are read successfully             |
| TC-02 | Video source unavailable          | Error is logged and handled              |
| TC-03 | Detector finds target class       | Detection output is displayed/logged     |
| TC-04 | No event meets decision threshold | No incident record is created            |
| TC-05 | Event meets configured rules      | Evidence and incident record are created |
| TC-06 | Database unavailable              | Failure is logged and handled            |
| TC-07 | E-mail provider unavailable       | Notification failure is logged           |
| TC-08 | Unauthorized dashboard request    | Access is denied                         |

### 7.2 Evaluation metrics

Object detection is evaluated on a held-out annotated dataset. Precision measures the proportion of predicted positives that are correct, recall the proportion of labelled objects found, and mAP summarizes precision across classes and localization thresholds. With TP, FP and FN denoting true positives, false positives and false negatives:

$$\text{Precision} = TP / (TP + FP) \quad (1)$$

$$\text{Recall} = TP / (TP + FN) \quad (2)$$

$$F1 = 2 \times \text{Precision} \times \text{Recall} / (\text{Precision} + \text{Recall}) \quad (3)$$

The action classifier is evaluated with accuracy, precision, recall, F1-score and a confusion matrix. The class distribution is stated because accuracy can be misleading when one class dominates, the test videos are independent of those used in training, and false alarms from normal physical activity and missed incidents are examined. Real-time performance is measured as sustained frames per second and end-to-end delay from the relevant video window to incident creation or alert, recording the hardware, input resolution, checkpoint and GPU status with each measurement. The whole pipeline is measured as well as individual models, because database writes, video encoding and e-mail delivery also add delay. Table 6 summarizes the metrics for each component.

**Table 6 - Model evaluation metrics.**

| Component           | Metric                             | Test configuration                |
|---------------------|------------------------------------|-----------------------------------|
| YOLO detector       | Precision / Recall / mAP           | Held-out object test set          |
| CNN-LSTM classifier | Accuracy / Precision / Recall / F1 | Held-out video test set           |
| Video pipeline      | FPS / end-to-end latency           | Specified hardware and resolution |
| Incident workflow   | Successful end-to-end cases        | Integration test set              |

### 7.3 Results and discussion

This paper defines the design, implementation and evaluation protocol of the prototype. Final claims about accuracy and real-time performance must rest on measured results from the held-out test sets and the hardware configuration described above; no performance figures are asserted here beyond that protocol. The metrics in Table 6 are the quantities to be reported, together with per-class results for rare or safety-relevant weapon classes and false-positive and false-negative examples that explain observed weaknesses.

#### 7.4 Limitations and challenges

Likely challenges include limited labelled examples, small objects, occlusion, low light, camera shake, compression artifacts, and differences between training footage and real deployment. Action recognition may confuse harmless movement with violence, multiple models can increase latency, and network interruptions and e-mail-service failures can affect alert delivery.

### 8. Conclusion and future scope

This paper presented an AI-based smart CCTV surveillance prototype that links video processing with object detection, action recognition, optional person tracking, incident decision logic, evidence capture, database storage, a web dashboard and e-mail notifications. The intended benefit is to help authorized personnel locate and review potential incidents more efficiently than by manually reviewing long recordings. Reliability depends on model quality, data coverage, processing speed and careful event rules, and AI outputs should be treated as indicators for human review, not as conclusions about intent or whether a crime occurred.

Future work includes:

Expanding the labelled dataset to cover more camera viewpoints and lighting conditions, and comparing newer detection and video-action models on the same test set;

Optimizing inference through model-size selection, frame sampling, batching or hardware acceleration;

Adding multi-camera management, health monitoring of disconnected streams, better incident de-duplication, evidence retention and audit logs;

Adding role-based access control and stronger authentication, and exploring local or privacy-preserving processing to reduce exposure of sensitive footage; and conducting a larger controlled evaluation with human reviewers before any real deployment.

#### Acknowledgements

The authors thank the Chairman, Secretary and Joint Secretary of Sri Shakthi Institute of Engineering and Technology for providing the platform and infrastructure for this work, the Principal, Dr. N. M. Saravana Kumar, for his support and for allocating time and resources, and the Head of the Department of Information Technology, Dr. S. Prakash, for guidance on the scope and systematic execution of the project. Special thanks are due to the project guide, Mrs. I. A. Jannathul Firthous, and the project co-ordinator, Dr. M. Deepa, for their valuable guidance and cooperation throughout the project.

#### Annexure A - Summary of project components

Table 7 lists the main technologies used in the prototype and the function of each.

**Table 7 - Summary of project components.**

| Component                 | Technology                                                                              | Primary function                                                                   |
|---------------------------|-----------------------------------------------------------------------------------------|------------------------------------------------------------------------------------|
| Object detection model    | YOLO11 Nano (yolo11n.pt)                                                                | Real-time weapon detection with low inference latency                              |
| Detection classes (9)     | Automatic rifle, bazooka, grenade launcher, handgun, knife, shotgun, SMG, sniper, sword | Detection taxonomy for threat mitigation                                           |
| Pre-processing and alerts | OpenCV (cv2)                                                                            | Frame capture, resizing, thresholding, bounding-box rendering and automated alerts |
| Data cleansing            | Python, Pandas, Shutil                                                                  | Audits annotation files and corrects label-index discrepancies                     |
| Action recognition        | CNN-LSTM (PyTorch)                                                                      | Candidate sequence-based violence classifier                                       |
| Tracking                  | ByteTrack / DeepSORT                                                                    | Associates detections across frames                                                |
| Backend and storage       | Flask, MySQL                                                                            | API, incident records and evidence references                                      |
| Dashboard and alerts      | HTML/CSS/JavaScript, SMTP                                                               | Incident review and e-mail notification                                            |
| Trained artifacts         | models/best.pt, models/best2.pt                                                         | Fine-tuned weights for deployment                                                  |

## Annexure B- Sample incident record

Table 8 shows the fields of an example incident record as stored by the system.

**Table 8 - Example incident record.**

| Field         | Example value                    |
|---------------|----------------------------------|
| incident_id   | Generated database ID            |
| camera_id     | CAM-01                           |
| event_type    | Potential violence               |
| timestamp     | Timestamp from the video/system  |
| confidence    | Actual model score, if available |
| status        | Pending human review             |
| evidence_path | Stored evidence file reference   |

## References

- Carreira, J., & Zisserman, A. (2017). Quo vadis, action recognition? A new model and the Kinetics dataset. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 6299–6308. <https://doi.org/10.1109/CVPR.2017.502>
- Cheng, M., Cai, K., & Li, M. (2021). RWF-2000: An open large scale video database for violence detection. *Proceedings of the 25th International Conference on Pattern Recognition (ICPR)*, 4183–4190. <https://doi.org/10.1109/ICPR48806.2021.9412502>
- Donahue, J., Hendricks, L. A., Guadarrama, S., Rohrbach, M., Venugopalan, S., Saenko, K., & Darrell, T. (2015). Long-term recurrent convolutional networks for visual recognition and description. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2625–2634. <https://doi.org/10.1109/CVPR.2015.7298878>
- Donahue, J., Hendricks, L. A., Guadarrama, S., Rohrbach, M., Venugopalan, S., Saenko, K., & Darrell, T. (2017). Long-term recurrent convolutional networks for visual recognition and description. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 39(4), 677–691. <https://doi.org/10.1109/TPAMI.2016.2599174>
- Kay, W., et al. (2017). The Kinetics human action video dataset. *arXiv preprint arXiv:1705.06950*. <https://arxiv.org/abs/1705.06950>
- OpenCV. (n.d.). *OpenCV documentation*. <https://docs.opencv.org/>
- Oracle. (n.d.). *MySQL reference manual*. <https://dev.mysql.com/doc/>
- OWASP Foundation. (n.d.). *Password storage cheat sheet*. [https://cheatsheetseries.owasp.org/cheatsheets/Password\\_Storage\\_Cheat\\_Sheet.html](https://cheatsheetseries.owasp.org/cheatsheets/Password_Storage_Cheat_Sheet.html)
- Pallets Projects. (n.d.). *Flask documentation*. <https://flask.palletsprojects.com/>
- PyTorch. (n.d.). *PyTorch documentation*. <https://pytorch.org/docs/>
- Sultani, W., Chen, C., & Shah, M. (2018). Real-world anomaly detection in surveillance videos. *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 6479–6488. <https://doi.org/10.1109/CVPR.2018.00678>
- Ultralytics. (n.d.). *YOLO11*. Ultralytics documentation. <https://docs.ultralytics.com/models/yolo11/>