INTERNATIONAL JOURNAL OF NOVEL RESEARCH AND DEVELOPMENT (IJNRD)
© 2026 IJNRD | Volume 11, Issue 5, May 2026 | ISSN: 2456-4184 | IJNRD.ORG

ALERT X: SMART SURVEILLANCE FOR PUBLIC SAFETY

T. Bharat Tez [1], Suvarna Agasthya [2], Mrs. J. Padmavathi [3]

[1]B.Tech Student, Department of Information Technology, Mahatma Gandhi Institute of Technology

[2]B.Tech Student, Department of Information Technology,Mahatma Gandhi Institute of Technology

[3]Assistant Professor, Department of Information Technology, Mahatma Gandhi Institute of Technology

Abstract : *The traditional CCTV systems are mostly passive in nature i.e. recordings are not looked at until an incident has already happened and then surveillance becomes a means of enforcing the law and not a preventive measure. This paper discusses AlertX, a real-time weapon detection and notification system that transforms an otherwise inert webcam feed into an on-top-of-it threat-detection layer. The system consists of a fine-tuned YOLOv8 object detection model exposed via a Flask web application. When a weapon is identified with a confidence score above a configured threshold, the system annotates the frame, stores the evidence image to disk, logs the detection to an SQLite database with a geo-tag taken at the device IP and an SMS notification is sent out to a designated phone number using the Twilio API. The annotated live feed is streamed to the web dashboard using the MJPEG protocol, and the most recent detections are displayed. JSON Web Token (JWT) authentication is used to protect the web dashboard, which runs on the MJPEG protocol, and displays running detection counters and the most recent detections. An event-driven cooldown (30 seconds) prevents an alert being issued on the same sustained threat, and the camera is automatically stopped on the first confirmed event to allow the operator to intervene. The implementation was tested on nine functional scenarios such as login, live streaming, model inference, alert dispatch, cooldown enforcement and database integrity; all scenarios were successful.*

IndexTerms - Weapon Detection, YOLOv8, Real-Time Surveillance, Flask, Twilio SMS Alert, JWT Authentication, SQLite, Public Safety, Computer Vision.

I.INTRODUCTION

The number of public-safety incidents involving handheld weapons in schools, transit hubs and congested urban areas, is only on the increase, and the closed-circuit television (CCTV) is now ubiquitous in such areas. However, the investigative capacity of standard CCTV systems has failed to match their physical size: most of the footage is looked into in hindsight, when an incident has already taken place. Any single human operator watching many screens loses meaningful situational awareness in a matter of minutes, and traditional motion-based alerting is semantically blind — all motion above a pixel-difference threshold will be alerted, producing unsustainable false-alarm rates that can quickly desensitize security personnel.

Recent generations of single-shot deep-learning object detectors (especially of the YOLO (You Only Look Once) family) have rendered real-time semantic understanding of video streams computationally feasible using commodity hardware. These detectors can run at frame rates that are suitable to live monitoring and with an acceptable accuracy on small custom datasets. This change allows it to be useful to transform an existing camera feed into a system that can recognise specific objects of interest — in this case, weapons — and respond automatically.

This paper describes AlertX, a real-time weapon detection and notification system, based on a fine-tuned YOLOv8 model and a Flask web application. The system has three tangible objectives: (i) to do per-frame weapon detection on a live camera feed with confidence-thresholded output; (ii) to send an SMS warning containing weapon type, confidence and geo-coordinates and a link to a map to designated personnel within seconds of a successful detection; and (iii) to provide an authenticated operator dashboard which streams the annotated feed and a history of all detections maintained in a

local database. The entire system runs on Python, with Flask, Flask-SQLAlchemy, Flask-JWT-Extended, the Ultralytics YOLO runtime, OpenCV, the Twilio REST client and the geocoder library, and is run on a standard laptop with an attached USB webcam.

The rest of this paper is structured in the following manner. Part II is a review of related research in the area of deep-learning-based weapon detection and CCTV analytics. Section III explains the methodology and separate modules. Section IV gives the system architecture. Section V elaborates on the workflow of the algorithm. The results of the functional test are reported in Section VI. Section VII is the conclusion of the paper and plans future work.

II. RELATED WORK

Initial CCTV analytics were based on motion sensors, pixel-differentiating sensors and human operator inspection. Such systems were semantically blank and caused chronic alert fatigue [1]. Later classical computer-vision detectors like Histograms of Oriented Gradients with Support Vector Machine classification, and Haar cascades, did not detect targets as well, but were fragile to viewpoint, lighting and occlusion, and never achieved weapon-detection performance at deployable false-alarm rates [2].

Object detection changed with the help of convolutional neural networks. Two stage R-CNN variants continued to achieve higher accuracy on natural-image benchmarks [3] and single-shot detectors like SSD [4] and the YOLO family [5] were able to achieve real-time frame rates by predicting boxes and classes simultaneously in one forward pass. The YOLOv8 release [6] refined the backbone (CSPDarknet with C2f blocks), the neck (PAN-FPN) and the detection head (anchor-free, decoupled) to give an excellent accuracy–latency trade-off and is exposed through a high-level Python interface that loads a .pt weights file and runs inference on a frame with a single call.

Olmos et al. presented Faster R-CNN-based handgun detections on still images, and Bhatti et al. tested YOLO on CCTV-style images, and showed significant improvements over classical baselines [7] and [8]. Even with this advancement, end-to-end deployable systems combining detection, an authenticated operator interface, and persistent logging and a real outbound alert channel are relatively uncommon in the literature; most published work terminates with offline benchmark numbers. To fill this gap, AlertX combines a fine-tuned YOLOv8 detector with a Flask web layer, a SQLite audit trail and a Twilio SMS dispatch path in a working system.

III. METHODOLOGY

The proposed system consists of six collaborating modules: the Authentication Module, the Frame Acquisition Module, the Detection Module, the Geo-location Module, the Alert Dispatch Module and the Persistence and Dashboard Module. The modules are independent components in the codebase and communicate via well-defined Python interfaces.

3.1 Authentication Module

Access to users is managed by a username and password login, supported by Flask-JWT-Extended. No passwords are ever stored in plaintext: the Werkzeug-generated password hash is stored along with a salt. Upon a successful POST to the /api/login endpoint, the server generates a JSON Web Token whose expiry is one hour and it gets stored in the browser in the localStorage. Later requests to the protected video stream endpoint include the token in the query string and the server checks it with the help of verify_jwt_in_request (locations=['query-string']) before opening the MJPEG response.

3.2 Frame Acquisition Module

Using OpenCV, each frame is captured with the default USB web camera. The capture loop is run within a Flask streaming generator such that each frame is coded as JPEG and is yielded with a boundary of frame and a MIME wrapper of multipart/x-mixed-replace. When the loop is ended or the operator halts the feed, the capture handle is released in a clean way.

3.3 Detection Module

The Detection Module loads the fine-tuned YOLOv8 weights at the path models/weapon-detector.pt with the Ultralytics YOLO class. Each captured frame f is fed through the model, which produces a list of detections in the form of (class, confidence, bbox). Only a detection with a confidence that meets confidence $\geq \tau$ is retained, where the configured parameter is CONFIDENCE_THRESHOLD = 0.6. The surviving detections are added to the frame as red bounding boxes and a label with the class-and-confidence using cv2.rectangle and cv2.putText functions. The annotated frame is then re-encoded as JPEG and handed over to the dashboard.

3.4 Geo-location Module

Once a detection is verified, the system adds a location to the event. The get_location routine first attempts to check a manual override via environment variables (LOCAL_LATITUDE, LOCAL_LONGITUDE, LOCAL_LOCATION_NAME), which are set up for indoor or LAN-only environments, and otherwise falls back to IP-based geocoding using geocoder.ip('me'). The outcome is a dictionary of latitude, longitude and a human-readable location name, which is stored in the detection record and embedded in the outbound SMS.

3.5 Alert Dispatch Module

The alerts are sent using the Twilio Programmable Messaging API. A client is created on startup with the environment variables TWILIO_ACCOUNT_SID and TWILIO_AUTH_TOKEN. The SMS text includes the type of weapon used, the rounded percentage of confidence, the name of the location, the current time and a link to https://maps.google.com/?q= for the location. In order to avoid alert storms in the case of a sustained event, a cooldown of 30 seconds is imposed: a new SMS is sent only when current_time - last_alert_time > 30. Upon successful dispatch, an alert flag named _active_alert is set on the detector instance so that the dashboard can show a full-screen alarm overlay.

3.6 Persistence and Dashboard Module

All detections (with or without SMS being dispatched) are written to a local SQLite database through Flask-SQLAlchemy. Each record stores the saved image path, the UTC-timestamp, latitude, longitude, location name, weapon type, confidence and a boolean alert_sent flag. Detected annotated images are written to static/detections/detection_YYYYMMDDHHMMSS.jpg. The dashboard polls three endpoints — /api/stats, /api/detections and /api/alert_status — every three seconds and refreshes the counters, recent-detection list, and alarm overlay.

IV. SYSTEM ARCHITECTURE

The design is based on a layered architecture where each layer has a single and well-defined responsibility. The Browser Layer manages the login, displays the dashboard, displays the MJPEG stream and shows the alarm overlay. The Flask Application Layer routes the HTTP requests, validates the JWTs, exposes the JSON APIs and streams the response. The Detection Layer encloses the YOLOv8 neural network and the OpenCV capture loop. The Integration Layer contains the Twilio client and the IP-geocoder. A single SQLite file accessed over SQLAlchemy ORM models constitutes the Persistence Layer. The alarm video and the saved detection JPEGs are static assets under the static/ directory.

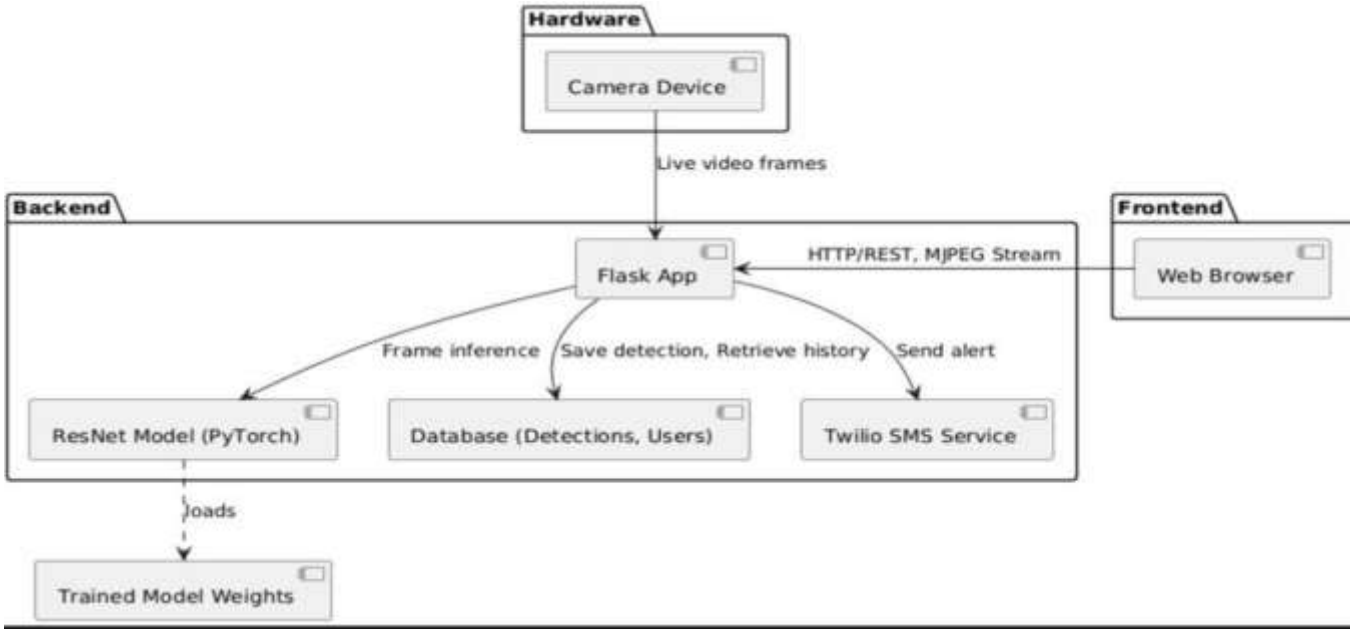

Fig. 1: System architecture of AlertX with the browser, Flask routes, detector, Twilio and SQLite layers

4.1 Workflow

The end-to-end flow follows six steps:

1. The operator uses the /api/login endpoint to authenticate.
2. The browser requests the server to start the capture loop via /video_feed?token=...
3. Each frame is passed through the YOLOv8 detector, and any surviving detections (confidence ≥ 0.6) are drawn on the frame.
4. For any frame with a detection, the location is determined, the annotated frame is stored, and a row is added to the Detection table.
5. An SMS sent by the Twilio client is subject to the cooldown; on success the alert_sent flag in the row is updated.
6. The capture loop reads the alarm state, sets camera_stopped = True and breaks to release the camera; the dashboard polls the alarm state three seconds later and shows the alarm overlay.

V. ALGORITHMIC WORKFLOW

The end-to-end algorithm, implemented in app.py, camera.py and models.py, is represented as the following sequence of steps:

5.1 Initialisation

Load configuration from the .env file, instantiate the Flask app, register JWT and CORS, create the static/detections/ directory if missing, instantiate the WeaponDetector which loads weapon_detector.pt, and call db.create_all().

5.2 Authentication

On a POST request to /api/login, verify the user by username, and issue a JWT signed with the key JWT_SECRET.

5.3 Open a Stream

On GET /video_feed?token=..., check the JWT in the query parameter and return a Response with body containing what the generator returns on detector.generate_frames(db, Detection).

5.4 Capture

Open cv2.VideoCapture(0). During the loop execution, read a frame; in case of read failure, break.

5.5 Inference

Run model(frame, conf=0.6) and transform each surviving box into a triple of (class, confidence, bbox).

5.6 Annotation and Persistence

If at least one detection survives: find the position using get_location; store the annotated frame to static/detections/detection_{timestamp}.jpg; add a Detection entry by inserting a row within app.app_context().

5.7 Alert Dispatch

Each time a detection is confirmed, call send_sms_alert(). When the cooldown window has expired, construct the message body, invoke twilio_client.messages.create(...) and update the row with an alert_sent flag.

5.8 Auto-Stop

Encode the annotated frame as JPEG, yield it to the client, set camera_stopped = True and break the loop so the camera is released through the finally clause.

5.9 Dashboard Polling

The browser queries the API at a rate of every three seconds: /api/stats, /api/detections and /api/alert_status. When alert_status returns active = true, the alarm overlay is displayed and the alarm video begins to play.

VI. RESULTS

6.1 Implementation Environment

The main dependencies, frozen in requirements.txt, are: Flask 3.0.0, Flask-SQLAlchemy 3.1.1, Flask-Cors 4.0.0, Flask-JWT-Extended 4.6.0, opencv-python 4.10.0.84, ultralytics 8.0.196, twilio 8.10.0, python-dotenv 1.0.0, geocoder 1.38.1 and Werkzeug 3.0.1. Upon startup, the Ultralytics runtime loads the fine-tuned weights file weapon_detector.pt. The default Flask development server is started on port 5000 with threaded=True in order to serve both the streaming response and the JSON polling endpoints simultaneously.

6.2 Output Screenshots and System Execution

The below following shows the results of the project execution:

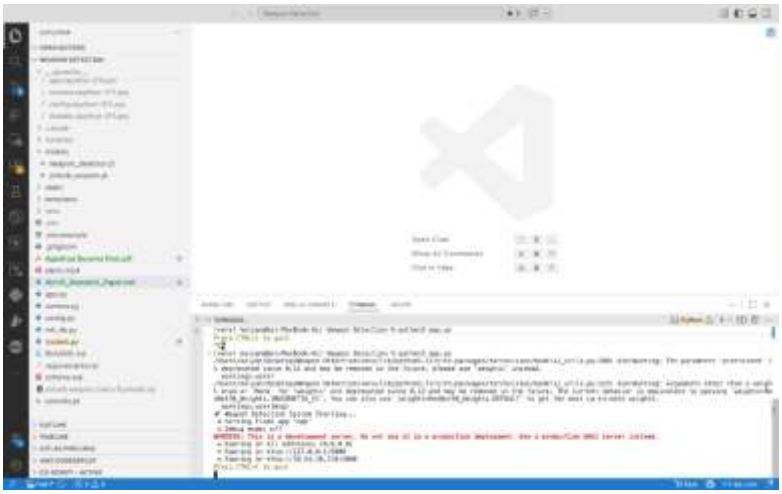

Fig. 2: Compiling in VS code

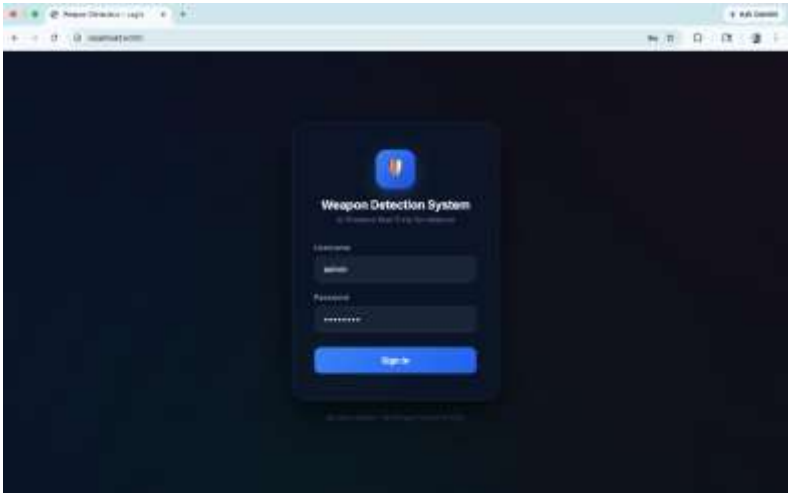

Fig. 3: Home page of Alert X

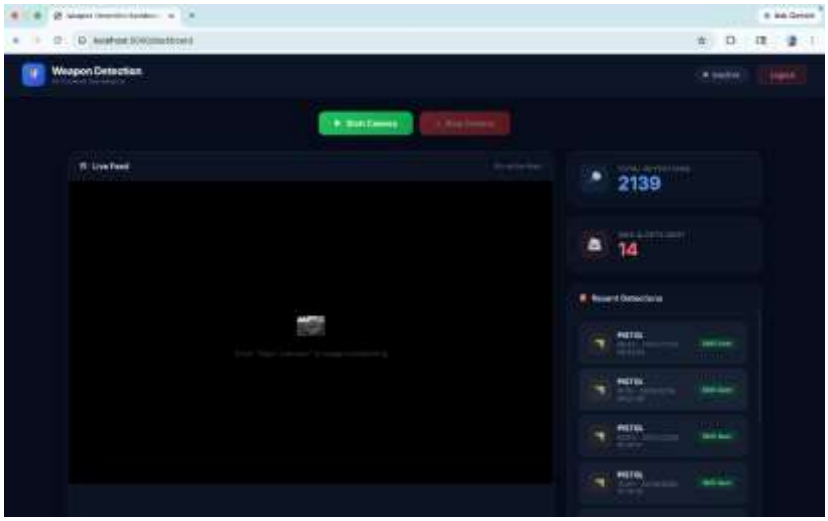

Fig. 4: Page after successful login as admin

Fig. 5: Image of a Pistol(firearm) detected by alert-x

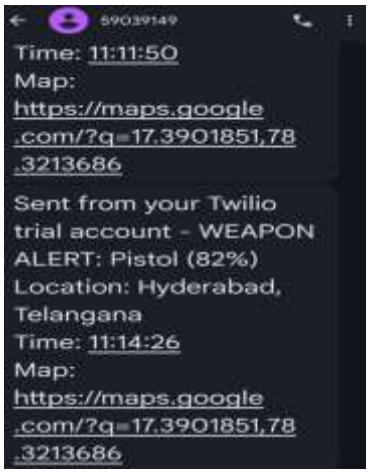

Fig. 6: Warning message at admin after firearm detected

Fig. 7: Screenshot of the SMS received to the admin's phone

ACKNOWLEDGEMENTS

The authors wish to thank the guide for their invaluable guidance and supervision throughout this project. The authors also acknowledge the Head of the Department and the project coordinators, who provided the necessary infrastructure and feedback during the work.

VIII. REFERENCES

[1] M. Valera and S. A. Velastin, "Intelligent distributed surveillance systems: a review," IEE Proceedings — Vision, Image and Signal Processing, vol. 152, no. 2, pp. 192–204, 2005.
[2] N. Dalal and B. Triggs, "Histograms of oriented gradients for human detection," in Proc. IEEE Conf. Computer Vision and Pattern Recognition (CVPR), 2005, pp. 886–893.
[3] S. Ren, K. He, R. Girshick and J. Sun, "Faster R-CNN: Towards real-time object detection with region proposal networks," in Advances in Neural Information Processing Systems (NeurIPS), 2015.
[4] W. Liu et al., "SSD: Single Shot MultiBox Detector," in Proc. European Conf. Computer Vision (ECCV), 2016, pp. 21–37.
[5] J. Redmon, S. Divvala, R. Girshick and A. Farhadi, "You Only Look Once: Unified, real-time object detection," in Proc. IEEE CVPR, 2016, pp. 779–788.
[6] G. Jocher, A. Chaurasia and J. Qiu, "YOLO by Ultralytics (YOLOv8)," Ultralytics, 2023. [Online]. Available: https://github.com/ultralytics/ultralytics
[7] R. Olmos, S. Tabik and F. Herrera, "Automatic handgun detection alarm in videos using deep learning," Neurocomputing, vol. 275, pp. 66–72, 2018.
[8] M. T. Bhatti, M. G. Khan, M. Aslam and M. J. Fiaz, "Weapon detection in real-time CCTV videos using deep learning," IEEE Access, vol. 9, pp. 34366–34382, 2021.
[9] T.-Y. Lin et al., "Microsoft COCO: Common objects in context," in Proc. ECCV, 2014, pp. 740–755.
[10] M. Grinberg, Flask Web Development, 2nd ed. Sebastopol, CA: O'Reilly Media, 2018.