iJRASET
International Journal For Research in Applied Science and Engineering Technology



INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Volume: 14 **Issue:** X **Month of publication:** October 2026

DOI: https://doi.org/10.22214/ijraset.2026.85494

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 X Oct 2026- Available at www.ijraset.com

SHIELDHER: An AI-Powered Women Safety System with Offline GSM Emergency Alert Mechanism

Fathima G.[1], Yackshita R.[2], Soumya R.[3], Swathi S.[4]
[1]*Professor, Adhiyamaan College of Engineering, Hosur* [2, 3, 4]*UG Scholars, Adhiyamaan College of Engineering, Hosur– 635109, India*

Abstract: ***A woman who suddenly finds herself in danger rarely has the time, the calm or the opportunity to unlock a phone, open an application and decide what to do next. ShieldHer is proposed to address this challenge. When danger arises, the user presses a single button on a wearable, ESP32-based device if the hardware is available, or the SOS button of a Flutter mobile application if the phone is available. Either action starts the same emergency workflow: the location is captured and shared with the saved emergency contacts, a camera frame is taken as evidence and analysed by a YOLOv8 detector for people and weapon-class objects, and the alert is stored with its source, coordinates and status in a Flask and SQLite backend until it is resolved. A five-second cancel window limits accidental alerts, an event-identifier check removes duplicates, and a SIM800L GSM module is included in the hardware design to send SMS when no internet connection is available. ShieldHer therefore brings the wearable trigger, the mobile application and the evidence-backed alert history together as one complete and user-friendly safety system. The backend passed 28 of 28 functional tests, created an alert in a median of 2.49 ms on the server.***
Keywords: women safety, SOS alert, wearable device, YOLOv8, ESP32, Flutter, Flask, GSM, evidence capture

I. INTRODUCTION

A safety tool is useful only if the person can start it at the worst possible moment. Many existing applications assume that the victim can take out a phone, unlock it, open an application and tap a button. Dedicated panic devices remove some of this friction, but they usually give the user no screen on which to review what happened afterwards, and the alert tends to remain in a message thread instead of a record that can be searched or closed. ShieldHer is built on a simple observation: the trigger, the evidence and the record should belong to one workflow.

In ShieldHer an alert can be raised in two ways. The first is a large SOS button in a Flutter mobile application. The second is a push button on an ESP32 board, intended to be worn or carried, which publishes an event to the Firebase Realtime Database; the application listens for that event and forwards it to the same backend. From that point the two sources are treated identically. The backend writes the alert to SQLite, marks it ACTIVE and keeps it in that state until the user resolves it. While the alert is being created, the application captures a camera frame and passes it to a YOLOv8 model, so that the person reviewing the alert sees not only a time and a place but also what the camera saw.

The work was carried out as an undergraduate mini project, and the claims are limited accordingly. The software path, from the SOS button to the stored record, was tested. The ESP32 and GSM parts were designed, wired and given a backend endpoint, but delivery statistics have not yet been collected for them. The contributions of this paper are as follows.

1. A unified alert pipeline in which phone and hardware triggers share one backend, one database schema and one alert history.
2. A cancel-or-confirm countdown and an event-identifier check that together reduce accidental and duplicated alerts.
3. An evidence step that attaches a camera frame and a detector summary (person count, weapon-class detections and confidence) to every alert.
4. A measured evaluation of the backend and screen-level results for the application and the detector, with the limits of each stated plainly.

The remainder of the paper is organised as follows. Section II reviews related work and compares ShieldHer with existing systems. Section III describes the architecture, workflow, backend and hardware. Section IV presents the AI threat detection module. Section V reports results, Section VI discusses them together with privacy considerations, Section VII lists limitations and the proposed evaluation protocol, and Section VIII concludes.

©IJRASET: All Rights are Reserved

II. RELATED WORK AND COMPARISON

A. Application-based systems

WoSApp [1] sends emergency messages from a smartphone to nearby people and to the police, and Abhaya [2] sends an SMS containing GPS coordinates to saved contacts. Both show that a phone alone can act as an alarm, but both require the user to operate the phone, and neither attaches visual evidence to the alert. Prashanth et al. [3] move the alert data to a real-time database so that alerts can be coordinated. In all three, the descriptions reviewed say little about preventing accidental triggers or about closing an alert once the incident is over.

B. Hardware and IoT devices

Saranya et al. [4] combine GSM and GPS in a dedicated device, so that the user does not depend on a smartphone application. Akram et al. [5] and Kane et al. [6] add further sensing, the latter applying machine learning to body-related parameters on a Raspberry Pi. Jayaram et al. [7] pair a switch, NodeMCU, GPS and GSM with audio analysis, and SafeShe [8] adds a camera module for visual evidence. These designs are strong on independence from the phone, but they carry the cost, power and maintenance burden of extra hardware, and sensor-driven triggering raises the risk of false alarms. The surveys by Farooq et al. [9] and by Bhuvaneswari and Thangamuthu [10] reach a similar conclusion about usability and deployment.

C. Camera-based emotion monitoring

The work closest to the theme of ShieldHer is the system of Yadav et al. [11], in which a camera and OpenCV classify facial emotion, count people by gender and drive a red or green LED and a buzzer. The approach is automatic and needs no action from the user. Its decision, however, rests on reading a face, and the authors themselves note that emotion recognition is limited outside controlled conditions. ShieldHer takes a different route: the user, or the user's hardware button, makes the decision to alert, and the camera is used afterwards to collect evidence rather than to judge the person's feelings.

D. What existing systems provide and what ShieldHer adds

WoSApp and Abhaya offer an application trigger and location sharing, but they offer no camera evidence and no wearable button, and the user must still operate the phone. The devices of Saranya et al. and Jayaram et al. offer a hardware trigger with GPS and GSM, which removes the dependence on the phone, but they provide no image evidence (Jayaram et al. provide audio only) and no application on which the alert can be reviewed and closed. SafeShe adds a camera to a device, yet the descriptions do not report an offline alert path or an alert log. The system of Yadav et al. provides automatic image-based monitoring, but it offers no explicit user trigger, no location sharing and no alert record. ShieldHer combines the strengths of these groups in a single system. It offers two triggers, a wearable button and a mobile application button. It adds a camera frame with a YOLOv8 summary to every alert, stores every alert in a history with ACTIVE and RESOLVED states, and includes a GSM module in the design so that an SMS can be sent when there is no internet. Table I summarises the comparison. The entries are taken from the descriptions in the cited papers, and "Not reported" means that the point was not stated there.

Table I. Comparison With Related Systems

System	Trigger	Location	Visual evidence	Works without internet	Alert log
WoSApp	Application	Not reported	None	No	No
Abhaya	Application	GPS in SMS	None	Yes-SMS based	No
Saranya	Device	GPS	None	Yes-GSM	No
Jayaram	Switch	GPS	Audio only	Yes-GSM	No
SafeShe	Device	GPS	Camera	No	No
Yadav	Automatic (face)	No	Images	No	No
ShieldHer	Wearable button and application	Application GPS and device GPS	Camera frame with detector summary	Yes-GSM (designed, not yet measured)	Yes, with ACTIVE and RESOLVED states

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 X Oct 2026- Available at www.ijraset.com

III. SYSTEM ARCHITECTURE AND WORKFLOW

A. Architecture overview

Fig. 1 presents the architecture of the complete system, in which two input paths feed one backend. The user, shown on the left, interacts with the Flutter mobile application, whose modules are the SOS button with a five-second countdown, location sharing, emergency contact management, alert history and status, a Call 112 shortcut and the optional AI-assisted visual analysis. The application sends the alert data to the Flask backend, which contains the API endpoints, alert processing, location handling, contact management, AI model integration and the business logic. The backend stores and retrieves records in the SQLite database, which holds the emergency records, the user data and contacts, the alert history and the system logs. The hardware path is shown below the application. The physical SOS button on the ESP32 sends an event to Firebase, which provides real-time event transfer, hardware event updates and synchronisation with the mobile application. The application, which is already listening, picks up the event and forwards it to the backend. The external services on the right of the figure are the location service, the AI model for visual analysis and the national emergency number 112. The five-step emergency flow at the bottom of the figure reads: SOS trigger from the button or the application, event transfer through Firebase, processing by the backend, storage of the record in the database and finally notification of the user with the alert, the location and the contacts.

The relay through Firebase was chosen instead of letting the board call the backend directly, because the board then needs only the Firebase credentials and a network connection, and it does not need to know where the backend is running.

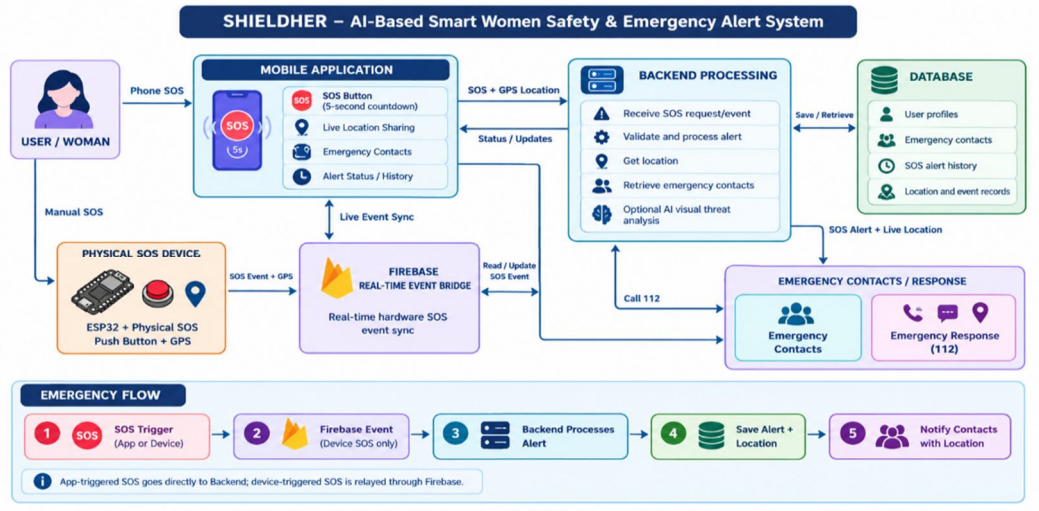


Fig. 1. ShieldHer system architecture.

B. Countdown and confirmation

Pressing SOS in the application does not send anything immediately. A five-second timer starts, decrements once per second, and either sends the alert when it reaches zero or stops when the user taps cancel. The delay is short enough to be tolerable in a real emergency and long enough for someone who has brushed the screen in a pocket to cancel. During the countdown the buzzer stays silent, so that a cancelled alert does not draw attention.

C. Backend and data model

The Flask service exposes REST endpoints [18] for registration, login, alerts, contacts and surveillance records. The alert endpoint rejects a request with no user identifier, checks that the user exists and inserts a row with the status ACTIVE. Each row holds the user, the alert type, latitude, longitude, a readable location name, the trigger source (application or hardware), the status and a timestamp. If no GPS fix is available, the record is still created and the location name is stored as “Location Unavailable”; a record without coordinates was preferred to no record at all. Passwords are stored using salted hashing instead of plain text [17]. SQLite was chosen because the prototype runs on a single machine and the database file survives a server restart.

The hardware endpoint takes a user identifier, a device identifier and an event identifier. Before inserting, the database layer checks whether the event identifier has been seen. If it has, the stored alert is returned with a duplicate flag and no new row is created. This is necessary because the same Firebase event can be delivered to the application more than once, for example after a reconnect.

©IJRASET: All Rights are Reserved | SJ Impact Factor 7.538 | ISRA Journal Impact Factor 7.894 |

D. Hardware platform

The embedded side is built on an ESP32-WROOM-32 [16], a dual-core controller with built-in Wi-Fi. It reads the emergency button, drives the buzzer, publishes the SOS event to Firebase over Wi-Fi, and communicates with the GPS receiver and the GSM modem over two serial ports. Table II lists each part and its role. Fig. 2 shows the prototype wiring on a breadboard: the NEO-6M GPS module and the SIM800L GSM modem are placed at the top of the figure and connected to the ESP32 through their serial pins, while the push button for the backup SOS and the buzzer are connected on the lower side. Table III gives the pin assignment.

Table II. Hardware Components And Their Role

Component	Role in ShieldHer
ESP32-WROOM-32	Main controller; Wi-Fi link to Firebase; handles button, buzzer, GPS and GSM
NEO-6M GPS	Supplies latitude and longitude as serial NMEA data for the alert location
SIM800L GSM modem	Sends SMS to saved contacts when no internet is available and a cellular signal exists
Push button	SOS trigger that starts the same emergency workflow as the application
Buzzer	Local audible indication; kept silent during the countdown
Breadboard, jumper wires	Prototype wiring with a common ground and shared power rails
Smartphone	Runs the Flutter application and the Firebase listener

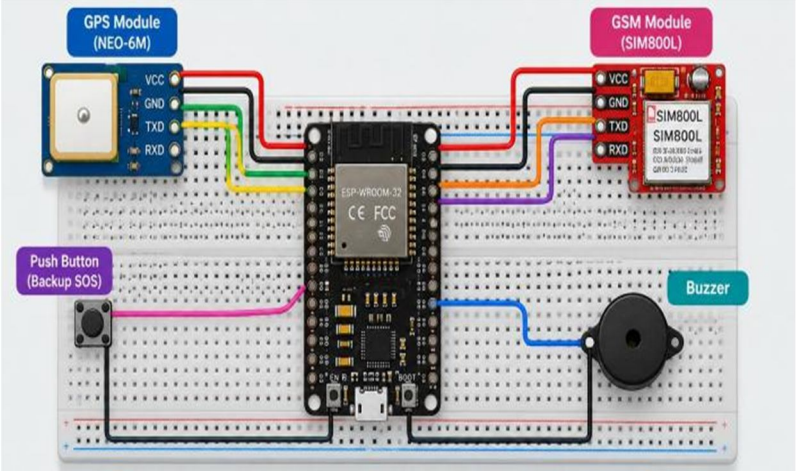


Fig. 2. Prototype wiring of the ESP32 with the NEO-6M GPS module, SIM800L GSM

The GPS receiver and the GSM modem each use a pair of pins on the secondary serial interfaces of the ESP32, so the two never compete for the same port. The button and the buzzer use one general-purpose pin each. All grounds are tied together, and the circuit uses no external resistors. The SIM800L draws short, high current peaks during transmission, so it is fed from its own 4.0–4.2 V supply and not from the ESP32 pin; an undersized supply makes the modem reset in the middle of a message, which is a common cause of unreliable GSM prototypes. When the internet is down but the cellular network is available, the firmware path is intended to send the SMS through the modem. This path has been wired, but its delivery rate has not yet been measured.

TABLE III. ESP32 PIN ASSIGNMENT

Module	Signal	ESP32 pin
NEO-6M GPS	TX to RX2	GPIO 16
NEO-6M GPS	RX from TX2	GPIO 17
SIM800L GSM	TXD	GPIO 26
SIM800L GSM	RXD	GPIO 27
Push button (backup SOS)	Input	GPIO 4
Buzzer	Output	GPIO 18

E. Alert life cycle

An alert passes through the small number of steps listed in Table IV. Only two states exist, ACTIVE and RESOLVED. The model was kept small on purpose: during an emergency nobody will operate a workflow with many stages, and a reviewer afterwards needs to know only whether the case is still open. An alert stays ACTIVE until someone taps Resolve in the history screen, so an unattended case remains visible at the top of the list instead of disappearing after a timeout.

Table IV. Steps of A Single Alert

Step	What happens	Component
1	SOS pressed in the application, or button pressed on the board	Flutter / ESP32
2	Five-second countdown; user may cancel (application path)	Flutter
3	Hardware event published and picked up by the application (button path)	Firebase
4	Request validated; user checked; event identifier checked	Flask
5	Row inserted with status ACTIVE and trigger source	SQLite
6	Camera frame captured and passed to the detector	Flutter / YOLOv8
7	Alert, evidence and threat state shown in Alert Details	Flutter
8	User marks the case RESOLVED	Flask / SQLite

F. Alert processing procedure

Algorithm 1 summarises how the backend handles a request from either source. The order of the checks is deliberate: the cheap checks on the request come first, the database lookups follow, and the detector is run last because its failure must never stop an alert from being saved.

ALGORITHM 1. HANDLING AN INCOMING SOS REQUEST

1) Receive request R from the application or the ESP32 endpoint.
2) If R has no user_id, return error 400.
3) If user_id is not in the users table, return error 404.
4) If R comes from hardware and its event_id is already stored, return the stored alert with duplicate = true.
5) Read latitude and longitude; if missing, set the location name to “Location Unavailable”.

©IJRASET: All Rights are Reserved | SJ Impact Factor 7.538 | ISRA Journal Impact Factor 7.894 |

6) Insert a row with status = ACTIVE and trigger_source, and obtain the alert id.
7) Capture frame F, run the detector on F, and compute the person count and weapon-class hits.
8) If the detector fails, keep the alert and mark the evidence as unavailable.
9) If there is a weapon-class hit or the person count is 2 or more, store a surveillance record with the snapshot path.
10) Return the alert (HTTP 201, or 200 for a duplicate).

IV. AI THREAT DETECTION

The detection module wraps a YOLOv8 model [12]. For every frame, the service calls the model with a confidence threshold, draws the boxes on a copy of the frame and walks through the returned boxes. For each box it reads the class index, converts it to a class name, records the confidence and the pixel coordinates, and increments a counter when the class is "person". The result is a list of detections, a person count and the annotated image. A separate service decides whether the result justifies a surveillance alert; if so, the camera identifier, alert type, person count, confidence, message and snapshot path are written to the surveillance table so that the entry can be reviewed later.

Two rules turn detections into a threat label. The first is based on class: when the model reports an object from a weapon-type class, the alert is marked with the threat type "weapons" and the highest confidence is shown. The second is a deliberately coarse rule used in the demonstration build: two or more people in the frame are reported as a potential threat. The interface states that the pretrained detector cannot classify an assault or an attack, so this rule only draws attention to a situation and cannot decide whether it is dangerous. It will produce false positives in any crowded scene and is treated as a placeholder for a proper activity model.

Detection quality depends on conditions that a phone camera meets in daily use. Low light increases noise and lowers confidence, a frame taken while the phone is moving can be blurred, and objects that are small, partly hidden by a hand or seen from an unusual angle are more likely to be missed. The confidence threshold is a trade-off in the same way: a low value catches more genuine cases but fills the review list with false ones, and a high value does the opposite. In a safety setting a missed weapon is usually worse than a false flag, so a lower threshold is preferred, and the human reviewer is relied upon to dismiss mistakes.

The detector output is attached to the alert as information. It never blocks, delays or cancels an alert; the SOS record is created whether or not the frame contains anything the model recognises. The alert detail screen shows one of two states: "No Threat Detected", or a threat summary with the threat type, confidence and number of detected objects.

V. RESULTS

A. Backend evaluation

The backend was exercised with 28 functional test cases covering registration and login, alert creation with and without coordinates, rejection of missing or unknown users, contact operations and the ACTIVE-to-RESOLVED transition. All 28 passed. Timing was measured on the server side only, with a median creation time of 2.49 ms and a 95th percentile of 2.88 ms. In the concurrency test, 50 clients together submitted 200 SOS requests, and all 200 were found in the database afterwards. These numbers describe the Flask and SQLite layer on the development machine. They do not include the phone, the network or Firebase, and they should not be read as the time between pressing a button and a contact being informed.

TABLE V. BACKEND TEST RESULTS

Measurement	Result
Functional test cases	28 run, 28 passed
Alert creation, median (server side)	2.49 ms
Alert creation, 95th percentile	2.88 ms
Concurrent load	50 clients, 200 requests, 200 stored
Lost or rejected valid alerts	0

B. Application screens

Fig. 3(a) shows the ShieldHer dashboard, which is the home screen of the application. The status card reports active protection, the location card shows the source of the coordinates, and the ESP32 card reads “Boundary Ready”. That label means the application has an integration point for the board; it does not mean that a board was connected during that session. Below these cards are the Call 112 shortcut, the evidence capture shortcut and the large SOS button. The Call 112 button opens the phone dialer for the national emergency number. It does not place a call or send information by itself, so it is not counted as an integration with emergency services.

Fig. 3(b) shows the result of a confirmed SOS, alert #9 of type Manual Emergency SOS, with its location, time and the evidence frame captured at that moment. The YOLOv8 panel under the frame reports a person count of 2 with a confidence of 88.0% and marks a potential threat under the two-or-more-people rule. The panel itself states that the rule is a prototype and that the pretrained detector does not classify assault, which is the honest reading of that result. Fig. 3(c) shows the AI surveillance status screen: AI surveillance is active, the model is YOLOv8 and live detection is on, followed by the stored, automatically captured SOS evidence with its capture time. The location in these runs is labelled as demo GPS for Anna Nagar, Chennai, and should not be read as a measured GPS fix.

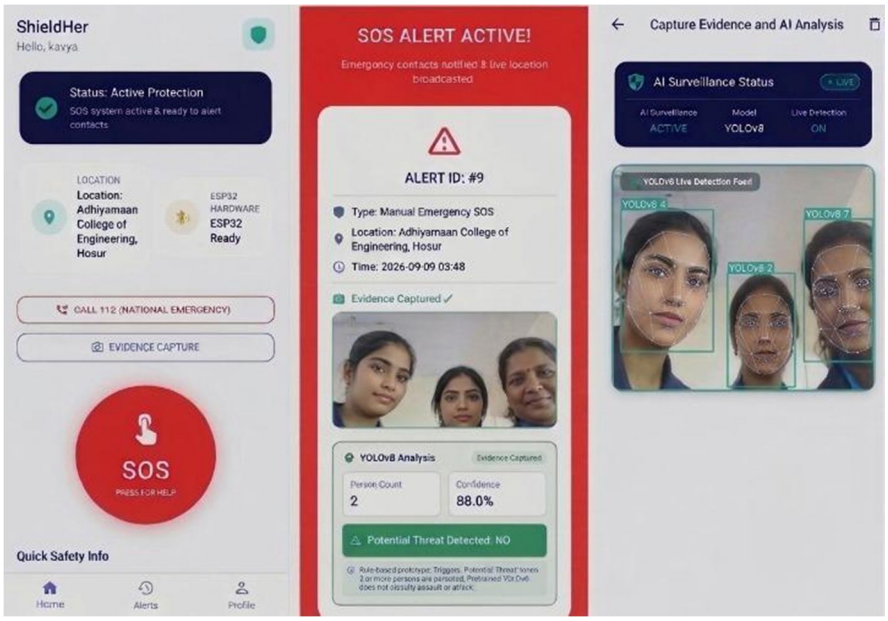


Fig. 3. ShieldHer application screens: (a) dashboard, (b) SOS alert active, (c) AI surveillance status.

C. Alert record from the application

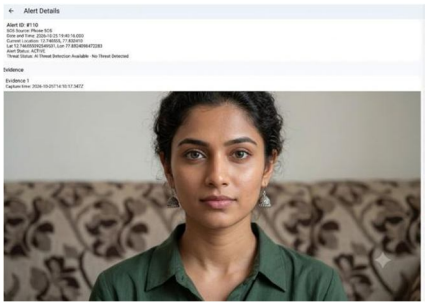


Fig. 4. Alert Details screen for alert #110.

Fig. 4 shows the alert detail screen for alert #110, raised from the phone SOS on 25 October 2026 at 19:40:16. The screen presents the trigger source, the date and time, the coordinates (12.746555, 77.832410), the status ACTIVE and the threat state “No Threat Detected”. Below this block, the evidence list holds the image captured at the time of the alert. Everything the responder needs to open the case is on one screen, which was the main aim of the design. The same layout is used for hardware-triggered alerts, with only the source field changed.

D. Weapon detection case

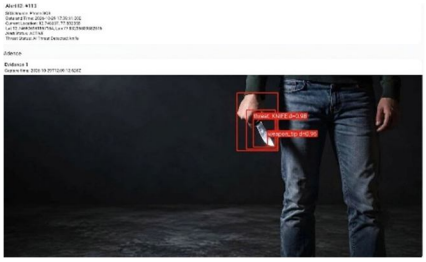


Fig. 5. Evidence 2 with the threat summary.

Fig. 5 shows a second evidence frame, taken on 11 October 2026, in which a pair of scissors is held towards the camera. The detector marked the scissors twice: one box around the whole object with a score of 0.98, and a smaller box around the handle with a score of 0.96. The summary above the image reports the threat type “weapons”, a confidence of 98% (the larger score) and seven detected objects in the frame. The case shows that the evidence step does what it was designed to do: a flagged frame comes with the box, the class and the score, so the person reviewing it can see why the system raised a flag.

The same case also shows the limit of the approach. Scissors are an ordinary household tool, and the model reacts to the shape, not to what the person intends. A single correct detection therefore says nothing about how often the detector is wrong, and no accuracy figure is claimed. The result should be read as a demonstration that a weapon-class detection reaches the alert screen with its evidence. A fair accuracy study would need a labelled test set with varied lighting, distance and occlusion, and would report precision and recall per class.

VI. DISCUSSION

A. Observations

Three points stand out from these results. First, the dual-trigger idea works at the level of the record: because an application alert and a hardware alert become the same kind of row, the history screen, the resolve action and the evidence view did not need to be written twice. Most of the effort in the prototype went into the boundary between the two paths, namely the relay through Firebase and the duplicate check, and not into the alert itself.

Second, the cost of the backend is negligible next to everything around it. At roughly 2.5 ms per alert, the server is not what will decide how fast help arrives. The phone’s network, the time Firebase takes to deliver an event and the SMS route will dominate, and those are exactly the parts still to be measured. It is better to say this plainly than to quote the 2.49 ms figure as if it were a response time.

Third, compared with the emotion-driven design of [11], an explicit trigger gives up some automation and gains clarity. A system that reads a face can act when the user cannot press anything, but it also has to decide whether a frightened expression means danger, a film or a bad day. In ShieldHer a person decides to ask for help, and the detector only adds context. The two ideas are not exclusive, and a later version could let a reliable distress signal start the same countdown that a button press starts today.

B. Privacy and ethical considerations

An alert contains a location, a face and the identity of the user, which together make a sensitive record. What is kept is limited to what the review needs: one frame per alert, the detector summary and the coordinates. Frames should be shown only to the owner and to the contacts the owner has chosen, and they should be deleted after a retention period set by the user. Any test image that shows a person should come from a volunteer who has agreed to its use. The frames in the figures are sample frames from the authors’ own test sessions.

VII. CONCLUSION AND FUTURE WORK

ShieldHer joins a wearable hardware button, a phone button, a location, a camera frame and an alert history into a single pipeline. The software path was tested thoroughly for a prototype: all 28 functional tests passed, alert creation took about 2.5 ms on the server, and no alert was lost in a 200-request concurrent test. Screens from the application show that a phone alert and a weapon detection both arrive as complete records with evidence attached. The hardware, GPS and GSM parts are the next thing to prove, and the detector needs a proper accuracy study.

©IJRASET: All Rights are Reserved | SJ Impact Factor 7.538 | ISRA Journal Impact Factor 7.894 |

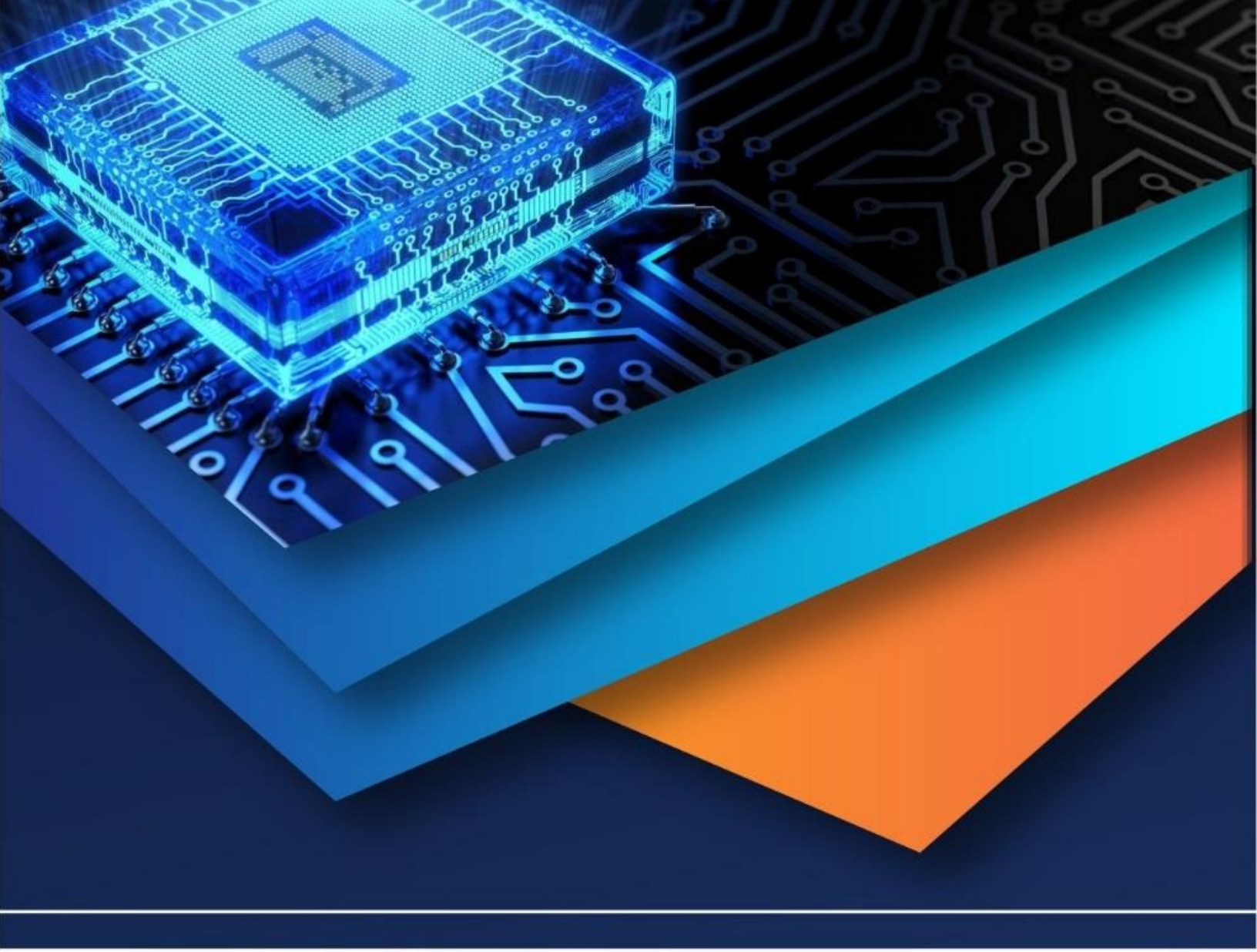
The next steps are to measure end-to-end latency from the button press to the SMS on a contact's phone with the internet disabled, to test GPS fix time indoors and outdoors, to replace the person-count rule with a model trained on activity or distress cues using consented data, and to move the backend to authenticated, encrypted communication. Direct hand-over to emergency services would also need an official interface and should be added only with legal advice.

VIII. ACKNOWLEDGMENT

The authors thank Dr. G. Fathima, Professor and Head of the Department of Computer Science and Engineering, for her guidance as supervisor, Dr. R. Radhakrishnan, Principal, and the faculty of Adhiyamaan College of Engineering for the facilities and support provided during this work.

REFERENCES

[1] D. Chand, S. Nayak, K. S. Bhat, S. Parikh, Y. Singh, and A. A. Kamath, "A mobile application for women's safety: WoSApp," in *Proc. IEEE TENCON*, 2015, pp. 1–5.
[2] R. S. Yarrabothu and B. Thota, "Abhaya: An Android app for the safety of women," in *Proc. IEEE INDICON*, 2015, pp. 1–4.
[3] D. S. Prashanth, G. Patel, and B. Bharathi, "Research and development of a mobile based women safety application with real-time database and data-stream network," in *Proc. IEEE ICCPCT*, 2017.
[4] N. Saranya, R. Aakash, K. Aakash, and K. Marimuthu, "A smart friendly IoT device for women safety with GSM and GPS location tracking," in *Proc. IEEE ICECA*, 2021, pp. 409–414.
[5] W. Akram, M. Jain, et al., "Design of a smart safety device for women using IoT," *Procedia Comput. Sci.*, vol. 165, pp. 656–662, 2019.
[6] A. Kane, T. Maktum, V. Mane, and N. Pulgam, "IoT based automatic women's safety device for enhanced personal security," in *Proc. IC3Com*, 2024, pp. 46–51.
[7] K. Jayaram, G. Krishna K., C. Hamsani, and K. Karishma, "Modern safety device for women's protection using IoT," in *Proc. ICRDICCT*, 2025, pp. 72–77.
[8] A. Joyce, S. Mohanalakshmi, P. Rohini, and S. Mohana, "SafeShe: IoT-enabled women security and well-being," in *Proc. ICRDICCT*, 2025, pp. 155–160.
[9] M. S. Farooq, A. Masooma, U. Omer, R. Tehseen, et al., "The role of IoT in women's safety: A systematic literature review," *IEEE Access*, 2023, doi: 10.1109/ACCESS.2023.3252903.
[10] V. S. S. Bhuvaneswari and M. Thangamuthu, "Towards intelligent safety: A systematic review on assault detection and technologies," *Sensors*, vol. 25, no. 13, Art. 3985, 2025.
[11] P. M. Yadav, V. A. Shingare, and N. C. Yadav, "AI-based intelligent women safety and surveillance system," *TIJER*, vol. 13, no. 8, pp. a622–a625, Aug. 2026.
[12] G. Jocher, A. Chaurasia, and J. Qiu, "Ultralytics YOLOv8," 2023. [Online]. Available: https://github.com/ultralytics/ultralytics
[13] Google, "Flutter documentation." [Online]. Available: https://docs.flutter.dev
[14] Pallets Projects, "Flask documentation." [Online]. Available: https://flask.palletsprojects.com
[15] SQLite Consortium, "About SQLite." [Online]. Available: https://sqlite.org/about.html
[16] Espressif Systems, "ESP32 series datasheet." [Online]. Available: https://www.espressif.com
[17] OWASP Foundation, "Password storage cheat sheet," OWASP Cheat Sheet Series. [Online]. Available: https://cheatsheetseries.owasp.org
[18] R. T. Fielding, "Architectural styles and the design of network-based software architectures," Ph.D. dissertation, Univ. California, Irvine, CA, USA, 2000.



10.22214/IJRASET

45.98

IMPACT FACTOR: 7.129

IMPACT FACTOR: 7.429

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Call : 08813907089 (24*7 Support on Whatsapp)