Published by :
https://www.ijert.org/
An International Peer-Reviewed Journal

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 06 , June - 2026

Implementation of a Hazardous Event Detection System using Edge AI.

(EdgeSafeAI)

Radhika Shinde
Department of Computer Engineering
Jayawantrao Sawant College of Engineering, Pune, India

Avishkar Bhusare
Department of Computer Engineering
Jayawantrao Sawant College of Engineering
Pune, India

Rutuja Hire
Department of Computer Engineering
Jayawantrao Sawant College of Engineering
Pune, India

Chaitrali Darekar
Department of Computer Engineering
Jayawantrao Sawant College of Engineering
Pune, India

Akash Gondale
Department of Computer Engineering
Jayawantrao Sawant College of Engineering
Pune, India

Abstract **- This paper documents the end-to-end implementation of a hazardous event detection system built on a Raspberry Pi 4 edge computing platform integrated with a CCTV camera input, a dual-module AI pipeline (object detection and action recognition), a Flask REST API backend with JWT security, and a Flutter-based mobile application. The implementation follows a layered architecture: Input Layer, Edge Processing Layer with AI Pipeline and Flask Backend, Security Layer, and Application Layer. Detailed implementation steps, configuration parameters, code design patterns, integration challenges encountered during development, and deployment procedures are described for reproducibility. The functional system successfully detects six categories of hazardous events in real time, issues secured HTTPS alerts to mobile clients at configurable intervals, and maintains an event log for retrospective investigation. This work serves as a reproducible reference implementation for practitioners developing edge AI surveillance systems.**

Index Terms—Edge AI implementation, Raspberry Pi surveillance, YOLOv8 deployment, Flask API, JWT authentication, Flutter mobile application, hazardous event detection, CCTV integration, TensorFlow Lite, real-time video processing.

I. INTRODUCTION

Translating a research architecture into a working implementation exposes a range of practical challenges that theoretical designs do not anticipate: driver compatibility, library version conflicts, thermal behavior under sustained load, network timing variability, and mobile platform notification delivery quirks. This implementation paper provides exhaustive, step-by-step documentation of realizing the Hazardous Event Detection system, targeting practitioners who intend to replicate or extend the system.

The system detects hazardous events from live CCTV video on a Raspberry Pi 4 using YOLOv8n for object detection and MobileNet-LSTM for action recognition. Detected events are served via a Flask REST API protected by JWT authentication and rate limiting, operating on a private local network. A Flutter mobile application polls the API at 3-second intervals and presents real-time hazard alerts, historical event logs, and a natural-language query interface.

This paper is organized as follows. Section II details the hardware and software bill of materials. Section III covers environment setup. Section IV documents the AI pipeline implementation. Section V describes the Flask backend. Section VI covers security implementation. Section VII documents mobile application development. Section VIII presents integration testing. Section IX discusses deployment considerations. Section X concludes.

II. BILL OF MATERIALS

A. Hardware Components

Edge Node: Raspberry Pi 4 Model B (8 GB RAM) [1], Google Coral USB 3.0 Accelerator [2], SanDisk Extreme 64 GB A2 microSD, GeeekPi active cooling kit (heatsink + PWM fan), Waveshare UPS HAT for uninterruptible power.

Camera: Hikvision DS-2CD2T47G2-L 4 MP ColorVu IP camera with RTSP support. Network: TP-Link TL-SG108E 8-port managed switch for VLAN isolation. Mobile Devices: Samsung. Galaxy A54 (Android 13) and Apple iPhone 14 (iOS 16) for testing.

B. Software Dependencies

Raspberry Pi OS 64-bit (Bullseye, kernel 6.1); Python 3.9.18; OpenCV 4.8.0; TensorFlow Lite Runtime 2.13.0 with Coral runtime 2.0.0 [4]; PyTorch 2.1.0 (armv8 wheel); Flask 3.0.0

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 06 , June - 2026

[5]; Gunicorn 21.2.0; Flask-JWT-Extended 4.5.3 [6]; Flask-Limiter 3.3.1 [7]; Redis 7.2; Pydantic 2.4.2.

Mobile: Flutter 3.16.0 [8]; Dart 3.2.0; packages: dio 5.3.3, flutter_local_notifications 16.1.0, sqflite 2.3.0, provider 6.1.1.

III. ENVIRONMENT SETUP

A. Raspberry Pi OS Installation and Configuration

Flash the 64-bit Raspberry Pi OS image to the microSD card using Raspberry Pi Imager [1]. Enable SSH and configure Wi-Fi credentials in the imager's advanced settings prior to flashing. On first boot, execute a full system upgrade, then configure static IP assignment via /etc/dhcpcd.conf to ensure consistent Flask endpoint addressing. Set the GPU memory split to 16 MB via raspi-config to maximise RAM availability for inference.

B. Python Virtual Environment

Create an isolated Python environment to avoid system-level dependency conflicts by initialising a virtual environment named hazedet_env. Install PyPI dependencies from a pinned requirements.txt to ensure reproducibility. Install the Coral runtime following Google's Coral documentation [2], which requires adding the packages.cloud.google.com/apt Debian repository and installing libedgetpu1-std.

C. RTSP Stream Validation

Confirm CCTV stream accessibility before AI pipeline integration by opening the RTSP URL through OpenCV's VideoCapture interface. A True response confirms connectivity. Configure the camera RTSP buffer size (CAP_PROP_BUFFERSIZE = 1) to minimise frame queuing latency.

IV. AI PIPELINE IMPLEMENTATION

A. Object Detection Module

Download the pre-trained YOLOv8n PyTorch model from Ultralytics [3] and export to TensorFlow Lite INT8 format [4]. Compile the resulting .tflite file to Edge TPU format using the Coral compiler [2]. The compiled _edgetpu.tflite model loads via the pycoral.utils.edgetpu module for hardware-accelerated inference.

The detection inference function accepts a pre-processed 640x640 normalised NumPy array, invokes the interpreter, and post-processes output tensors to extract bounding boxes, class IDs, and confidence scores. Non-maximum suppression with IoU threshold 0.5 filters redundant detections. The function returns detection dictionaries with keys: class_name, confidence, and normalised bbox coordinates (x1, y1, x2, y2).

B. Action Recognition Module

The MobileNetV2 backbone, pre-trained on ImageNet, is loaded via PyTorch's torchvision library and truncated at the penultimate global average pooling layer to yield 1280-dimensional feature vectors. These vectors are accumulated in a deque of length 16 (approximately 2 seconds at 8 FPS). A two-layer LSTM (input: 1280, hidden: 256, layers: 2, dropout: 0.3) followed by a linear classification head (256 to 6 classes) consumes the feature sequence. The model was fine-tuned for 40 epochs with Adam optimiser (lr = 1e-4, weight decay = 1e-5) and cross-entropy loss.

At inference, the deque is converted to a (1, 16, 1280) tensor and passed through the LSTM in no_grad mode. Softmax over the 6-class output yields action probabilities. An action is flagged hazardous if any hazard class probability exceeds 0.70. The deque uses a sliding window with 1-frame temporal stride.

C. Final Decision Logic

A Decision class merges outputs from both modules. Hazard is declared True when an object is detected above the confidence threshold OR a hazardous action is recognised. Event severity is scored on a 1-5 scale: fire/smoke, violence/assault, and handgun score 5; fall scores 4; intrusion scores 3; running in a restricted zone scores 2. The decision output dictionary, serialised as JSON, forms the payload returned by the Flask API.

V. FLASK BACKEND IMPLEMENTATION

A. Application Structure

The Flask application [5] follows a modular factory pattern to support testing isolation. The directory structure includes: app/__init__.py (application factory), app/routes/detect.py (detection endpoint), app/services/pipeline.py (AI pipeline singleton), app/models/event.py (Pydantic event schema), app/utils/auth.py (JWT helpers), app/utils/limiter.py (rate limiter configuration), config.py (environment-based configuration), and run.py (Gunicorn entry point).

B. /detect Endpoint

The POST /detect route accepts Content-Type: application/json with a body containing the base64-encoded JPEG frame, camera_id, and ISO 8601 timestamp fields, validated by Pydantic's DetectRequest model. Frame decoding proceeds through base64 decode, numpy frombuffer, cv2.imdecode, and BGR array conversion. HTTP 200 returns the result; HTTP 422 on malformed input; HTTP 429 on rate limit exceeded; HTTP 401 on missing or invalid JWT [9].

C. Gunicorn Deployment

Gunicorn is launched with 4 workers bound to 0.0.0.0:5000, a 30-second timeout, and 5-second keep-alive. The AI pipeline singleton is instantiated once per worker in the worker_init hook to avoid repeated model loading overhead. A systemd unit file (hazedet-api.service) ensures automatic restart on failure and on Raspberry Pi boot.

VI. SECURITY LAYER IMPLEMENTATION

A. JWT Authentication

Flask-JWT-Extended [6] is configured with JWT_SECRET_KEY read from an environment variable (never hard-coded), JWT_ACCESS_TOKEN_EXPIRES set to 24 hours, and JWT_ALGORITHM = HS256. The POST /auth/token endpoint verifies credentials against a bcrypt-hashed store and issues a signed access token [9]. All protected endpoints use @jwt_required(). Token revocation

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 06 , June - 2026

uses a Redis-backed blocklist keyed by token JTI with TTL matching remaining validity.

B. Rate Limiting

Flask-Limiter [7] is initialised with a Redis storage backend and default limits of 200 requests per day and 60 per minute. The /detect endpoint carries a stricter limit of 30 per minute to prevent pipeline abuse. Rate limit response headers (X-RateLimit-Limit, X-RateLimit-Remaining, X-RateLimit-Reset) are included in all responses for client-side throttle handling.

C. Network Isolation and TLS

VLAN 10 (192.168.10.0/24) isolates the Raspberry Pi and CCTV camera from the general LAN. iptables rules restrict port 5000 to VLAN 10 hosts and the WireGuard tunnel endpoint. TLS termination is handled by an Nginx reverse proxy [10] using a self-signed 4096-bit RSA certificate. The mobile application pins the certificate via Flutter's SecurityContext.setTrustedCertificatesBytes, preventing man-in-the-middle attacks on the local network.

VII. MOBILE APPLICATION IMPLEMENTATION

A. Architecture

The Flutter application [8] follows the Provider state management pattern. Core providers: AlertProvider (manages hazard state and polling timer), EventLogProvider (SQLite-backed event history), and AuthProvider (JWT token storage in Flutter Secure Storage). Navigation uses GoRouter with three primary routes: /dashboard (live monitoring), /log (event history), and /query (natural-language interface).

B. Real-Time Alert Polling

A periodic timer with a 3-second interval drives the polling loop. The callback issues an HTTPS POST to /detect with the most recent camera frame buffered on the mobile device. On receiving a hazard response, AlertProvider notifies all listeners, triggering a full-screen alert overlay and a local push notification via flutter_local_notifications with a category-specific sound asset.

C. Event Log

Detected events are persisted to a local SQLite database via the sqflite package. The events table schema: id (INTEGER PRIMARY KEY), event_type (TEXT), severity (INTEGER), confidence (REAL), camera_id (TEXT), timestamp (TEXT), thumbnail_b64 (TEXT). The EventLogProvider exposes filtered query methods for the log screen. Export to CSV is supported for forensic handoff.

D. Natural-Language Query Interface

The query screen accepts free-text input such as 'Show fire events from last night' and constructs SQL WHERE clauses via regex-based intent extraction, including event type keyword matching and temporal expression parsing. This rule-based NLU avoids inference latency on the mobile device; a future version will integrate a local LLM via llama.cpp Flutter bindings.

VIII. INTEGRATION TESTING

Integration testing employed a three-phase protocol. Phase 1 (Unit): each module was tested in isolation — object detection on 500 held-out labelled frames, action recognition on 200 temporal test clips, Flask API against 45 Postman automated test cases, and the mobile application against a mock server.

Phase 2 (System): the complete pipeline was exercised with a live CCTV feed in a controlled indoor environment using staged hazard scenarios (fire extinguisher spray as a smoke substitute, trained actors for violence and fall events).

Phase 3 (Endurance): 72-hour continuous operation with synthetic alert injection every 45 seconds. Three edge cases were resolved: (a) simultaneous Coral and PyTorch inference on the same USB bus caused occasional resets, resolved by a dedicated-power USB hub; (b) JWT clock skew exceeding 5 seconds caused intermittent 401 errors, resolved by NTP synchronisation; (c) Android 13 required an explicit POST_NOTIFICATIONS runtime permission request.

IX. RESULT ANALYSIS

Metric	Measured Value	Remarks
Object Detection Accuracy (YOLOv8n)	97.2%	Tested on 500 labeled frames
Action Recognition Accuracy (MobileNet-LSTM)	91.8%	Tested on 200 video clips
Average Inference Time	185 ms	Per frame on Raspberry Pi 4
API Response Time	72 ms	Flask API processing
End-to-End Latency	357 ms	Camera to mobile alert
Mobile Alert Delivery Time	1.2 s	Average notification delay

Table I. System performance results

The experimental evaluation demonstrates that the proposed Edge AI-based hazardous event detection system is capable of delivering accurate and timely hazard identification in real-world environments. The integration of YOLOv8n and MobileNet-LSTM enables reliable detection of both hazardous objects and activities while maintaining low inference latency. Performance results indicate that the system can operate efficiently on resource-constrained edge hardware without significant degradation in accuracy. The achieved response time and high system availability confirm its suitability for continuous surveillance applications. Overall, the results validate the effectiveness of the proposed architecture for real-time safety monitoring and emergency alert generation.

IX. DEPLOYMENT CONSIDERATIONS

A. Thermal Management

Sustained dual-model inference raises the ARM Cortex-A72 junction temperature to 72°C without active cooling,

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 06 , June - 2026

approaching the 80°C throttle threshold. The GeeekPi PWM fan activates at 60°C, maintaining temperature below 68°C in ambient conditions up to 35°C. Deployments above 35°C ambient require a dedicated enclosure with forced air circulation.

B. Power Resilience

The Waveshare UPS HAT provides 30 minutes of battery backup, enabling graceful shutdown via a Python atexit hook. The systemd service is configured with Restart=always and RestartSec=5 to recover automatically from crashes and prevent SD card corruption on abrupt power loss.

C. SD Card Longevity

Write-intensive workloads are mitigated by mounting /tmp as tmpfs, disabling Redis AOF persistence (snapshot-only mode), and flushing the SQLite event log at 60-second intervals rather than per event. For high-write deployments, an SSD connected via USB 3.0 is strongly recommended.

X. CONCLUSION

This paper provided a comprehensive implementation guide for a Raspberry Pi-based edge AI hazardous event detection system, covering hardware assembly, environment configuration, AI model deployment, Flask API development, security hardening, mobile application construction, integration testing, and deployment operations. The documented system is fully functional and successfully detects hazardous events in real time with sub-400 ms end-to-end latency.

Practitioners can use this guide as a complete reference for deploying equivalent systems in industrial, commercial, or institutional settings. Future directions include multi-camera federation, federated learning-based model improvement, and integration with building management systems for automated emergency response.

REFERENCES

[1] Raspberry Pi Foundation, 'Raspberry Pi 4 Model B Datasheet,' 2023. [Online]. Available: https://datasheets.raspberrypi.com/rpi4/raspberry-pi-4-datasheet.pdf

[2] Google LLC, 'Get started with the Coral USB Accelerator,' 2023. [Online]. Available: https://coral.ai/docs/accelerator/get-started/

[3] G. Jocher, A. Chaurasia, and J. Qiu, 'Ultralytics YOLOv8,' 2023. [Online]. Available: https://github.com/ultralytics/ultralytics

[4] TensorFlow Lite, 'Post-training quantization,' Google LLC, 2023. [Online]. Available: https://www.tensorflow.org/lite/performance/post_training_quantization

[5] A. Ronacher, 'Flask documentation 3.0.x,' Pallets Project, 2023. [Online]. Available: https://flask.palletsprojects.com/en/3.0.x/

[6] vimalloc, 'Flask-JWT-Extended documentation,' 2023. [Online]. Available: https://flask-jwt-extended.readthedocs.io

[7] alisaifee, 'Flask-Limiter documentation,' 2023. [Online]. Available: https://flask-limiter.readthedocs.io

[8] Flutter Dev Team, 'Flutter documentation,' Google LLC, 2023. [Online]. Available: https://docs.flutter.dev

[9] M. Jones, J. Bradley, and N. Sakimura, 'JSON Web Token (JWT),' RFC 7519, IETF, May 2015.

[10] Nginx Inc., 'Nginx: Configuring HTTPS servers,' 2023. [Online]. Available: https://nginx.org/en/docs/http/configuring_https_servers.html