Interdisciplinary Journal of AI, Machine Learning & Data Science (IJAIMLDS)

ISSN: 3139-3527 | April-June-Issue, Vol. 1, No. 2 (2026) | DOI: 10.66261/fa1cep06

Article - e005

EDGEAWARE: DISTRIBUTED THREAT DETECTION AND MONITORING SYSTEM FOR SMART CITIES

Amit Ramchandra Patil[1,2], Dr. Rajesh Bansode[3]

[1]Department of Information Technology, Thakur College of Engineering and Technology, Kandivali, Mumbai, 400101, Maharashtra, India
[2]Sr. Software Engineer, Here Technologies, Mumbai, 400063, Maharashtra, India
[3]HOD, Department of Information Technology, Thakur College of Engineering and Technology, Kandivali, Mumbai, 400101, Maharashtra, India

Received: 27/02/2026 **Revision Received**: 30/03/2026 **Accepted**: 05/04/2026

ABSTRACT

EdgeAware is a distributed threat detection and monitoring system designed for smart-city applications where quick and coordinated actions are required for public safety and infrastructure management. Traditional city monitoring systems often rely on centralized processing and single threat event processing, and it can suffer from high latency, heavy bandwidth usage, limited scalability, and reduced reliability under real-world variability such as different viewpoints and domains. Many academic prototypes lack operational support for validation of event and multi-event monitoring. To address these gaps, EdgeAware integrates edge computing with a centralized monitoring workflow. The system demonstrates two multi-view object detection models for the detection of accident and pothole events, trained on YOLOv11s architecture using diverse perspective images from dashcam, drones, and surveillance system. It improves generalization across different domains and viewpoints. For edge deployment, models are converted to TensorFlow Lite and executed in an Android application that performs real-time inference on live camera streams. The app supports IoU-based tracking to filter duplicate detections, enabling event-level reporting. It transmits only representative frame per unique event, encrypted together with metadata and location information for secure and bandwidth-efficient communication. A backend platform is used for ingesting the events and providing a monitoring dashboard with visualization using Here Maps. This is for administrative validation and coordination with response teams such as police, ambulance, and road maintenance teams.

The experimental results have also shown good quality in the detection aspect, with mAP@0.5 at 93.7%, as well as good precision and recall for the models. The edge pipeline also resulted in an average time of approximately 90 ms and a runtime end-to-end latency of 1.5 s, as well as a reduced storage footprint through compression.

KEYWORDS: Computer vision, Deep learning, Object detection, YOLO, Accident detection, Pothole detection, Edge computing, Android, Smart city.

Interdisciplinary Journal of AI, Machine Learning & Data Science (IJAIMLDS)

ISSN: 3139-3527 | April-June-Issue, Vol. 1, No. 2 (2026) | DOI: 10.66261/fa1cep06

1. INTRODUCTION

1.1 Introduction

Smart cities are designed to help citizens and municipal systems by using digital technologies into urban infrastructure and services. It needs to monitor city environments in real-time, understand threat events accurately, and react in a timely manner to avoid impact on public safety and infrastructure. Smart city applications include heterogeneous sources of inputs such as surveillance cameras, vehicle-based dashcams, drones (UAVs), and mobile devices that constantly collect large amounts of visual information. If this information is processed efficiently, it can offer intelligent insights for smart traffic control, road maintenance, public safety, and emergencies.

Most smart city monitoring systems use centralized cloud processing that may be subject to issues such as latency, network congestion and privacy concerns. For time-critical threats such as road accidents or fires, small delay can cause the effectiveness of response and rescue operations. Therefore, smart city applications demand distributed intelligence, where detection and preliminary decision-making can be performed close to the data source i.e., at the edge devices. Events like potholes are less time-sensitive but automating reporting of such events can result into faster response time than manual reporting. Edge-based processing allows rapid local inference, reduces unnecessary data transmission, and can preserve privacy by sharing only essential information rather than raw video streams [1]-[5]. Fig. 1 shows the different threat events and emergency situations in urban environment [5]. Edge computing can enable effective management system for such events in coordination with response team.

Fig. 1 Various threats in urban environment

Proposed system is designed as a multi-tier architecture that combines edge computing with centralized monitoring. The core idea behind this is to use widely available edge devices such as smartphones, drones, and embedded processors with cameras as active sensor and detection nodes. These nodes perform real-time object detection to identify threat events, generate alerts for each event, and transmit only selected evidence like, a single representative

ISSN: 3139-3527 | April-June-Issue, Vol. 1, No. 2 (2026) | DOI: 10.66261/fa1cep06

frame with encrypted metadata to monitoring system. This reduces bandwidth requirement and improves scalability of the system. It can be useful in any environment like urban or rural area, infrastructure facilities such as manufacturing units or in natural environment such as forests due to the variety of edge devices it can utilize.

The proposed system supports general smart city use cases such as prioritization of emergency dispatch by providing location specific visual evidence to monitoring teams, rapid accident awareness, traffic and rerouting support, pothole and road safety reporting for maintenance and repairing schedule, and city-wide situation awareness by notifying events and emergencies. Multi-view training increases the detection quality of model by generalizing across different camera angles, altitudes, and lighting conditions. It is useful for robust real-world deployment. Overall, EdgeAware contributes a practical framework for accurate threat detection, secure event reporting, and potential coordinated response, aligned with key smart city goals of safety, efficiency, and urban governance.

1.2 Motivation

There are three practical reasons behind the design of EdgeAware. First is response time is critical. It is vital for emergency response to accidents, as well as prompt scheduling of road maintenance for potholes, which reduces secondary accidents, congestion, and economic loss. Centralized video analytics can have a high response time because raw video needs to be transmitted to a remote server. Edge computing seeks to minimize response time by computing analytics closer to the source, thus minimizing end-to-end latency [1][2].

Second, cities must deal with multiple types of events. Most of the literature is focused on single-purpose solutions such as accident response pipelines or only pothole detection system. In real-world scenarios, there is a need for unified monitoring of diverse events by agencies such as the municipal corporation/governance teams. There is a strong literature focus on accident response using video analytics, emergency reporting, etc. [6]-[10], while other critical road threats such as potholes have received limited attention [11][12].

Third, edge computing introduces security and resource constraints. Edge devices such as mobile phones, small GPUs, embedded CPUs like Raspberry Pi, Jetson Nano are heterogeneous in nature and usually have limited resources. Reporting events through such devices must be achieved securely with minimum bandwidth, storage, and power. Surveys highlight that security, privacy, and interoperability are major challenges for edge deployment [3][4].

1.3 Problem Statement

Despite the availability of good cameras and network connectivity, current smart-city monitoring systems have several limitations: (i) Centralized processing requires continuously sending of high-volume video data, which increases bandwidth cost and increases the latency. (ii) Existing systems are mostly designed for a single threat type and cannot be easily extended to multiple events. (iii) Generalization across viewpoints is limited in the existing system. Models trained on a specific camera setting cannot perform well when deployed on different viewpoints, resolutions, or lighting variations and weather conditions. (iv) Event reporting is often insecure or inefficient in sending full clips or many frames can violate privacy, consume storage, and overload network links.

Therefore, the problem addressed in the proposed system is to create a smart-city monitoring system that detect and report multiple road threats in real time, across heterogeneous camera

ISSN: 3139-3527 | April-June-Issue, Vol. 1, No. 2 (2026) | DOI: 10.66261/fa1cep06

viewpoints, while meeting edge benchmarks of low latency, low bandwidth, limited storage, and strong security.

1.4 Objectives

The objectives of the EdgeAware project are:

1. To design a distributed architecture that performs real-time threat detection at the edge and supports potential coordinated response through a municipal backend platform.
2. To demonstrate multi-event monitoring using two object-detection models i.e. accident detection and pothole detection, trained on multi-view imagery to improve deployment flexibility.
3. To convert the trained models using TensorFlow Lite, enabling inference on android devices for edge deployment.
4. To implement an android application for demonstration that performs live multiple model inference, supports per-model confidence threshold setting, and uses IoU-based tracking and deduplication to convert frame-level detections into unique events.
5. To implement a backend system using Django and PostgreSQL that ingests event reports using REST APIs, visualizes them on dashboard with a map interface, enables administrator validation by visualizing evidence frame, and supports coordination with response teams.

To align the end-to-end pipeline with edge benchmarks including security using encrypted evidence and access control, low latency with on-device inference, low bandwidth by sending only evident frame per event rather than continuous video, and reduced storage by using compressed ML models.

1.5 Contributions

The key contributions of proposed system are as follows:

- The system was developed as a complete operational workflow that connected real-time edge inference, secure event reporting, backend ingestion, map-based monitoring, and response coordination, rather than limiting the scope to offline detection evaluation.
- Developed accident and pothole detectors trained on multi-view imagery (dashcam, UAV, surveillance) to improve generalization across diverse smart-city deployment contexts within a single platform.
- Deployed YOLOv11s via TFLite on Android with per-model thresholding and IoU-based tracking to deduplicate detections and report only unique events, reducing alert noise and backend workload.
- Transmitted only selected representative encrypted frame and metadata per unique event, improving privacy, lowering bandwidth and storage needs, and enabling scalable city-wide monitoring.

ISSN: 3139-3527 | April-June-Issue, Vol. 1, No. 2 (2026) | DOI: 10.66261/fa1cep06

2. LITERATURE REVIEW

2.1 Existing Work

EdgeAware intersects three research areas: edge computing for smart-city services, video-based accident detection and emergency response, and road-condition monitoring (potholes) using computer vision.

Edge computing for smart cities: Industry surveys and systematic studies indicate an increasing adoption of hybrid edge–cloud architectures, driven by the demand for low-latency analytics in transportation and manufacturing [1][2]. However, these studies also point out unresolved challenges such as interoperability, security, and the absence of standardized evaluation benchmarks [1]-[3]. Security surveys stress that edge systems must address threats like DDoS, malware injection, privacy breaches, and insecure device onboarding, while remaining energy-efficient and deployable on diverse devices [3]. Lightweight cryptography techniques and resource-aware strategies have been investigated to enable optimized processing with minimal energy and CPU overhead on resource-constrained IoT nodes [4].

Various studies integrate sensor data and machine learning techniques to detect accidents and alert emergency services. Systems combining IoT and deep learning utilize sensors like accelerometers, impact sensors, and GPS with CNN models to detect events and trigger alerts [6]. Vision-based approaches employ CCTV or dashcam video with CNN pipelines and third-party messaging APIs for real-time event dispatch [7]. Systems that incorporate evidence reporting and visualization (e.g., sending images or 3D reconstructions) enhance situational awareness but may increase network load and security risks [8]. Deep-learning video models (I3D, ConvLSTM) and feature fusion strategies enhance robustness to weather and viewpoint variations but still face challenges under occlusion and low-light conditions [9]. Reviews of multi-angle crash video systems suggest that integrating multiple cameras improves accuracy and reduces latency, though computational cost and fog/night robustness remain significant limitations [10].

Pothole detection and road-condition monitoring: Pothole detection has been tackled using traditional image processing, vibration sensors, and more recently, object detection models. Benchmarking studies compare YOLO variants under challenging road conditions and report competitive detection accuracy (e.g., YOLOv7 mAP@0.5 ≈ 0.884) while highlighting dataset and generalization issues [11]. Recent YOLOv8-based pothole detectors demonstrate high accuracy and low inference time, but many studies remain single-view and do not incorporate secure municipal workflows [12]. UAV and aerial imagery introduce additional challenges for small-object detection, and lightweight variants modify YOLO backbones and feature pyramids to enhance small-target recall while maintaining a low parameter count for drone deployment [13].

Existing literature in this field suggests the need for an integrated, secure, multi-event monitoring platform that performs edge inference on different viewpoints across various domains and reports only relevant and concise evidence for validated response. The proposed system is designed to meet this need by combining multi-view training, edge-optimized TFLite deployment, and a centralized backend workflow for monitoring.

ISSN: 3139-3527 | April-June-Issue, Vol. 1, No. 2 (2026) | DOI: 10.66261/fa1cep06

2.2 Literature Survey

Edge-aware smart-city systems are fundamentally linked to the broader trend of edge computing. George et al. [1] integrate a literature review with cross-sector surveys and interviews to assert that hybrid edge–cloud setups can minimize latency and offer substantial ROI for transportation and industrial monitoring. Their strength lies in an adoption-centric perspective rooted in industry practices; however, their limitation is a strategic focus, acknowledging interoperability and security challenges but providing few standardized KPIs or deployable reference frameworks. Prasad et al. [2] conduct a systematic analysis of edge enablers (MEC, SDN/NFV, microservices, containerization, orchestration) and primarily evaluate benefits through simulation, indicating that containerized MEC services can decrease latency and that intelligent scheduling might lower energy consumption. The advantage is a clear linkage from components to anticipated benefits; the limitation is the lack of real-hardware validation and the absence of common benchmarks for reproducible comparisons.

Security is repeatedly cited as a first-class requirement for edge use cases. Sheikh et al. [3] provide a survey of edge security threats such as malware, DDoS attacks, privacy leakage, routing attacks, and ML-specific attacks such as poisoning and evasion. They also survey the corresponding mitigants such as AI-based intrusion detection systems, federated learning, blockchain integrity, and hardware trust anchors such as PUFs. The advantage is the development of a taxonomy for threat and mitigation; the disadvantage is the absence of benchmark security data sets and experimental evaluations, along with the requirement for energy-aware and quantum-resistant assumptions. Rozlomii et al. [4] are closer to the edge with an experimental evaluation of lightweight cryptography for constrained devices such as those in the IoT ecosystem (e.g., Arduino and STM32). The advantage is the demonstration of the potential for lightweight ciphers to reduce CPU and energy overheads and improve device up-time; the disadvantage is the lack of device and hardware specificity regarding threat modelling and the absence of standardized cross-device evaluations.

From an operational perspective, Costa et al. in [5] present an overview of emergency management in smart cities in relation to IoT-based detection-alert-mitigation architectures. The benefits include an overall system perspective that incorporates the relationships between detection, analysis, and response workflows. The disadvantages include poor consideration of cybersecurity aspects, scalability/fault tolerance verification, and data modeling standards, which become critical in handling large numbers of concurrent incident reports.

In the field of accident detection, there are studies based on pipelines using sensor-based approaches as well as those using vision-based approaches. A two-stage system is proposed by Pathik et al. in [6], using IoT sensors for force, GPS, and GSM signals, as well as transfer learning classifiers for embedded devices. The benefits include high accuracy and better generalization using ensemble approaches. The disadvantages include limited datasets for analysis, hardware dependencies and costs, as well as latency in notifications and poor consideration of secure evidence sharing. Nandini et al. in [7] propose using CCTV video streams with a custom CNN architecture and OpenCV for analysis and Twilio for notifications. The advantage is an end-to-end pipeline that demonstrates feasibility in monitored areas; limitations include viewpoint and lighting generalization issues, limited privacy/security discussion, and no multi-event scope.

Aslam et al. [8] improve accident reporting by incorporating visual evidence of situations using controllers such as ESP32, MQTT/GSM, and verification delays, which reduce false reporting. The advantage is improved situational awareness, while the limitation is increased bandwidth/privacy risk, especially with visuals, and potential verification delays conflicting

ISSN: 3139-3527 | April-June-Issue, Vol. 1, No. 2 (2026) | DOI: 10.66261/fa1cep06

with emergency response needs, without strong edge CV security/robustness. Adewopo and Elsayed [9] propose a deep learning ensemble method using a combination of RGB and motion information (optical flow) with I3D and ConvLSTM2D, showing improved robustness against weather, with possible efficiency-oriented variants. While the advantage is using motion information, there is limited performance against occlusion/low lighting, false positives with static scenes, and explainability for verification by municipalities. SAFE ROAD AI [10] surveys various methods using multiple views of crash videos, such as YOLO family, Mask R-CNN, optical flow, and tracking methods such as Kalman filters, concluding that using multiple views can help improve the accuracy and latency of response by minimizing blind spots of single cameras. It needs improvement in sparse datasets and performance in foggy or night conditions. It also lacks evaluation of edge constraints, and unified, secure reporting workflow.

For pothole detection, comparative studies have validated the YOLO family's suitability while revealing gaps in their deployment. Lieskovská et al. [11] have benchmarked YOLOv3, v5, v7 on road scenes, showing high accuracy with practical inference speeds (with YOLOv7 showing good performance). The benefits lie in evidence-based selection of models, while drawbacks include generalization challenges, issues with dataset diversity, and poor integration with municipal workflows and security requirements. Addanki and Lin [12] have trained YOLOv8, showing high mAP and inference speeds, thus making them feasible for near real-time use. Drawbacks include limited testing with multiple views, a lack of device benchmarking, and inadequate investigation of compression/quantization effects, security, and bandwidth-aware reporting, which is critical for real-world Android deployment.

Additionally, Luo et al. [13] proposed ESOD-YOLO for aerial view sensing. The authors proposed the use of the YOLOv8n baseline with a RepNIBMS backbone, WFPN fusion, a small object head, and tri-focal loss for improved accuracy with low parameters for UAVs. The advantage of this proposal lies in the explicit optimization of small object detection for drone views. The limitation lies in the absence of the integration of smart city response with the end-to-end solution and the absence of the exploration of transferability for dashcam/CCTV and aerial views. Khanam and Hussain [14] proposed an overview of the YOLOv11 architectural improvements such as the use of C3k2, SPPF, and attention modules. The advantage lies in the improved accuracy-efficiency trade-offs. The limitation lies in the absence of task-based benchmarking and the use of TFLite for edge device studies.

Collectively, these research works show that there are recurring gaps that inform the current smart city threat monitoring research and practice. For one, there is a need for better robustness with respect to variability in real-world scenarios, which includes multi-view scenes, scenes under low-light conditions, scenes with occlusion, scenes with fog or at night, as well as domain variability in different smart cities and roads. Furthermore, it is also clear from the research that smart city research and practice must look beyond single-purpose solutions and instead provide a solution for multi-event monitoring under a single workflow, allowing smart city authorities to manage multiple types of threats under a single, unified platform. Finally, there is a need for a holistic evaluation of edge computing performance, where latency, bandwidth, storage, and security/privacy are collectively optimized and validated, as opposed to isolated optimization/validation of model accuracy.

Interdisciplinary Journal of AI, Machine Learning & Data Science (IJAIMLDS)
ISSN: 3139-3527 | April-June-Issue, Vol. 1, No. 2 (2026) | DOI: 10.66261/fa1cep06

3. METHODOLOGY

EdgeAware is designed as a distributed pipeline with three layers including edge layer, communication layer and centralized control layer as shown in the Fig. 2. Edge layer includes inference on heterogeneous devices such as Mobile phones, CCTV cameras, vehicle mounted dashcam, drones etc. It is facilitated with the small GPUs and embedded CPUs. Event of various threats are generated at this layer and securely reported over the network through communication layer. Centralized control layer is responsible to receive these events and provide effective monitoring, validation, and potential response coordination.

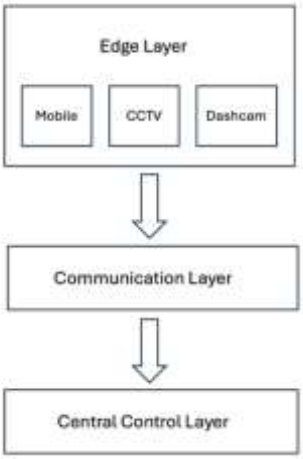

Fig. 2 Three architectural layers of EdgeAware

The main idea is to use the edge device as a first point of intelligence that performs detection and aggregation of events locally and only sends compact evidence. This minimizes bandwidth usage as well as latency compared to sending raw video data. On the other hand, the backend offers governance features such as event storage, visualization, and auditing required by municipal organizations.

The high-level architecture of EdgeAware can be described as a distributed pipeline that connects local intelligence to centralized municipal monitoring as well as coordination with response team. It has five main blocks that function as showed in Fig. 3.

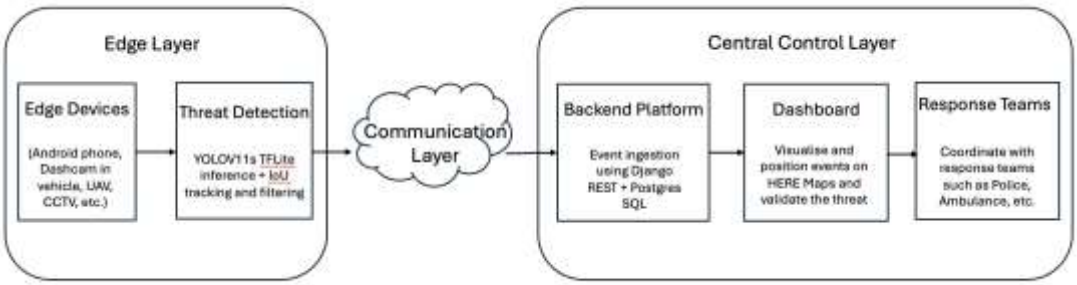

Fig. 3 High-level architecture diagram of EdgeAware, including various blocks in the three layers for complete distributed pipeline.

ISSN: 3139-3527 | April-June-Issue, Vol. 1, No. 2 (2026) | DOI: 10.66261/fa1cep06

EdgeAware starts with the first layer of sensing visuals in the smart city, in this layer multiple edge devices function as distributed observers. It includes devices such as vehicle-mounted Android phone cameras for dashcam view, UAV/drone cameras for aerial view, and surveillance cameras installed alongside roads for roadside view. The main function of this module is to capture live video stream from its viewpoint and process them locally. Since these devices operate near the area where the incident is taking place, they can quickly identify any potential threat, even if network connectivity is not always available. Each edge device uses a YOLOv11s object detection model, which is converted to TensorFlow Lite (TFLite) format. This object detection model processes video frames from the live video stream, detecting objects such as, Accident, Pothole, etc. To avoid reporting the same real-world incident multiple times in successive video frames, EdgeAware uses an IoU-based tracking and filtering module. This module uses the Intersection over Union (IoU) method to match current object detections with previous detection. It provides a unique identity for each event and filter out redundant events if any. It helps reducing false alarms and avoid repetitive notifications which minimize bandwidth usage.

After a unique threat event is identified at the edge, a small event bundle is sent to the backend. This backend uses Python's Django as a web framework, along with PostgreSQL as a data store. This part of the system is responsible for safe ingestion of events, validation of payload structure, safe storage of event metadata such as time, GPS location, device type, model used, confidence, and tracking ID. This backend acts as a central source of truth for threat events reported throughout the distributed edge nodes. This block also offers a dashboard to administrators such as those in a municipality or government organization. Events received at this backend are visualized on a HERE Maps display to show their location, allowing administrators to see events within their operational area of interest. This part of the system allows administrators to inspect events through visual confirmation of threat events through an encrypted representative frame, as well as to invalidate a threat event if it is a false alarm. This step helps to increase overall system reliability through human interaction, reducing the impact of false positives on system operation, as well as building trust in automated system responses.

After validation, the system can support coordinated response by alerting appropriate teams based on threat types and severity. This may include alerting police and ambulance teams in the event of an accident or alerting road maintenance teams in the event of a pothole. It can be further expanded to automate response rules and prioritization.

The entire system is demonstrated through an android application for threat detection such as an accident and a pothole. Fig. 4 is the android edge runtime process flow for the detection and reporting of threat events.

Interdisciplinary Journal of AI, Machine Learning & Data Science (IJAIMLDS)
ISSN: 3139-3527 | April-June-Issue, Vol. 1, No. 2 (2026) | DOI: 10.66261/fa1cep06

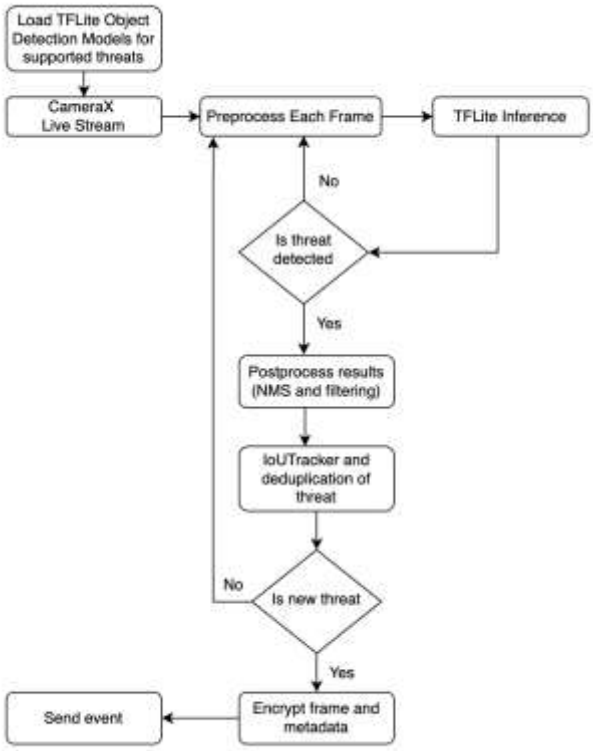

Fig. 4 Android runtime pipeline including camera stream, preprocessing, TFLite inference, post processing, tracking and secure event upload.

At runtime, the Android application first loads the required TFLite object detection models for the supported threats (for example, accident and pothole). It then starts a CameraX live stream and processes the incoming video frame-by-frame. Each frame is preprocessed (such as resizing and normalization) to match the model's input format, after which TFLite inference is executed to obtain detections. If no threat is detected, the system immediately continues to the next frame to maintain real-time performance. When a threat is detected, the output is postprocessed using steps like Non-Maximum Suppression (NMS) and confidence-based filtering to remove duplicate or low-quality bounding boxes. The remaining detections are passed to an IoU-based tracker, which compares detections across consecutive frames to identify the same physical object/event and performs event-level deduplication. If the tracker decides that the detection corresponds to an already reported event, it is ignored and the pipeline continues. If it is identified as a new unique threat instance, the app selects a representative frame and encrypts the frame along with metadata (such as threat type, confidence, timestamp, and location). Finally, the encrypted payload is sent as an event report to the backend, ensuring security while reducing bandwidth by transmitting only one frame per unique threat rather than streaming continuous video.

In this system object detection models plays crucial role. Training object detection model that will support multi-view compatibility and evaluation of its feasibility for the deployment is itself a dedicated pipeline which is shown in the Fig 5.

Interdisciplinary Journal of AI, Machine Learning & Data Science (IJAIMLDS)
ISSN: 3139-3527 | April-June-Issue, Vol. 1, No. 2 (2026) | DOI: 10.66261/fa1cep06

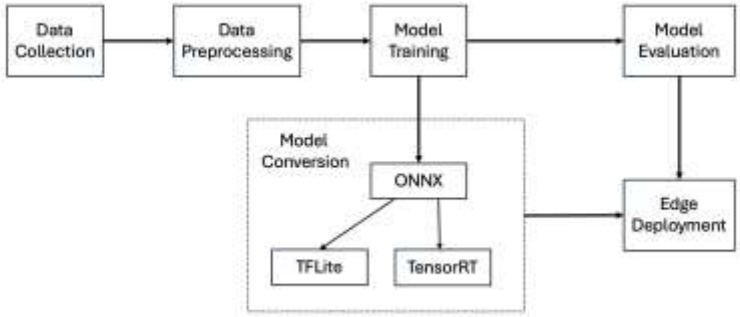

Fig. 5 Model lifecycle for multi-view training and edge deployment.

3.1 Data Collection

The demonstration focuses on two common road threats including accident representing collisions, overturned vehicles and pothole representing road-surface holes, cracks or severe pavement distress that can damage vehicles. Each threat is modeled using a dedicated YOLOv11s detector. Using separate models simplifies label space, allows different thresholds and anchors, and makes it easier to extend the platform by adding new specialized detectors such as fire, flooding, illegal dumping without retraining a single large multi-class model.

To improve deployment ability across smart-city settings, the training dataset is collected from multiple viewpoints such as street level imageries, surveillance cameras and aerial images. Street level imageries are collected from dashcams, phone-mounted vehicle cameras and augmenting the images using low angle, motion blur, changing exposures. Fixed surveillance cameras expressing high viewpoint stability, wide field-of-view and variable resolutions added to the dataset. UAV/drone imagery shows top-down or oblique view, smaller target scale and complex background which adds more diversity to the dataset.

Bounding-box annotations are created using consistent guidelines focusing on tight boxes around damaged road regions of potholes and relevant vehicle regions or collision indicators of accident. Some online videos containing potential accidents and pothole instances are gathered and frames are extracted from it to manually annotate. Also, already annotated images are gathered from online open-source datasets. Two-pass quality assurance process is used to ensure that the annotations having good quality by verifying invalid or duplicate boxes, class-name consistency, partial occlusion, nighttime glare and shadows using manual review.

Besides the annotated images that have accidents and potholes, some negative images from the same sources but without any accidents and potholes are added to the dataset. This negative data is included intentionally to provide context of what an image should look like when there are no accidents and potholes on the road. It helps to reduce false positives, meaning that the detectors will be able to identify real threats and differentiate them from the usual scenery on the road to improve the overall robustness and reliability of the detection system. Distribution of the dataset is given in the Table 1 for both the models showing the numbers of images per class.

ISSN: 3139-3527 | April-June-Issue, Vol. 1, No. 2 (2026) | DOI: 10.66261/fa1cep06

Table 1 Class distribution of dataset

Class	Accident Model	Pothole Model
True class label images (Accident/Pothole)	4237	5574
Negative images	1108	928

Each of the subset mentioned in above Table 1 contains 50% street level images captured from dashcam, 30% aerial images and 20% images from surveillance cameras. Dataset is recommended to split into 70% training, 20% testing and 10% validation set.

3.2 Data Preprocessing

Data preprocessing involves standardization of input data as well as robustness enhancement. Primary operations involve frame extraction/filtering, resolution normalization, color/illumination normalization. Frame extraction/filtering involves sampling video sources at a constant stride of 3-10 FPS to minimize redundancy while maximizing temporal diversity.

Images are resized to a size of 640x640, which is the input size during training, while maintaining the aspect ratio through letterboxing. Optionally, histogram equalization/adaptive normalization can also be performed to minimize exposure variation effects.

Data augmentation plays a crucial role in multi-view generalization. Geometric as well as photometric augmentations are part of a typical pipeline. Random scaling, translation, rotation, perspective transformation are part of geometric data augmentation. Brightness/contrast adjustment, blur effects, rain/fog simulation, color adjustment are part of photometric data augmentation. Care is taken to avoid unrealistic scenarios that can induce noise in labels such as extreme rotations of road images. Albumentations, Scikit-image, OpenCV libraries are used for image data augmentation.

3.3 Model Training

YOLO models are popular for real-time object detection tasks owing to their good accuracy-speed trade-offs. The system proposed utilizes the YOLOv11s model for a balance of accuracy and deployment constraints for mobile/edge computing. An overview of YOLOv11 was published, which discussed the architectural advancements of the model for better efficiency and performance, such as optimized convolutional blocks and attention mechanisms, among others [14]. YOLO models learn bounding boxes, objectness, and class probabilities. The model is trained using the Ultralytics tool. This simplifies the training of the model. It needs a dataset configuration file (YAML) which is created with class names and paths for the train, valid, and test datasets. The model is then trained with minimal setup and configuration of hyperparameters. Table 2 is a representation of the system configuration for training the model.

ISSN: 3139-3527 | April-June-Issue, Vol. 1, No. 2 (2026) | DOI: 10.66261/fa1cep06

Table 2 System requirements for model training

Requirement	Version
Python	3.12
Ultralytics	8.3.117
Torch	2.9.0
GPU	NVIDIA GeForce
CUDA	12.6

After that, training is performed by executing a single command that mention the various hyperparameters such as learning rate schedule, batch size, input resolution, optimizer, and augmentation strength. Early stopping is done by using validation mAP. Accident and pothole detectors are trained independently to allow tailored augmentation and sampling. Hyperparameters used for both models are given in the Table 3. Accident data is sampled to include normal traffic scenes to reduce false positives, while pothole data includes varied road textures to avoid overfitting to a particular pavement style. Both tasks can be imbalanced, especially if accidents are rare. To handle that strategies such as balanced sampling and hard-negative mining is used. Some confusing non-event scenes are added to the dataset.

Table 3 Hyperparameters used for model training for each model

Parameter	Accident	Pothole
Input image	640x640	640x640
Epochs	200	1000
Batch size	16	16
Optimizer	AdamW	AdamW
Learning rate	0.002	0.002
Augmentations	scale, crop, blur, brightness,	scale, crop, brightness,
Classes	Accident	Pothole
Split ratio	70/10/20 (train/valid/test)	70/10/20 (train/valid/test)

3.4 Model Evaluation

Evaluation is conducted along two dimensions; one is object detection quality and other is edge-deployment performance. The primary object-detection metrics are precision, recall, and mean Average Precision (mAP). For a given confidence threshold:

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

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

Average Precision (AP) is computed as the area under the precision–recall curve. mAP averages AP across classes and IoU thresholds. Two commonly reported settings are mAP@0.5 and mAP@[0.5:0.95].

Bounding-box overlap is measured using Intersection over Union (IoU):

$$IoU = \frac{Area(BBox_{pt} \cap BBox_{gt})}{Area(BBox_{pt} \cup BBox_{gt})} \quad (3)$$

ISSN: 3139-3527 | April-June-Issue, Vol. 1, No. 2 (2026) | DOI: 10.66261/fa1cep06

where $BBox_{pt}$ is the predicted bounding box and $BBox_{gt}$ is the ground-truth bounding box. IoU thresholds determine whether a prediction counts as a true positive.

For edge readiness, the various metrics are measured on target device such as, model size (in MB) before and after conversion, inference latency measured in ms/frame and throughput measured by FPS, End-to-end pipeline latency calculated by the total of capture, preprocess, inference and postprocess, tracking time along with time send event to backend system and render on UI.

A simple latency calculation is expressed as:

$$T_{total} = T_{capture} + T_{pre} + T_{infer} + T_{post} + T_{tracker} + T_{event} + T_{ui} \quad (4)$$

Where $T_{capture}$ is the time need to capture frame using CameraX feed, T_{pre} is the preprocessing time of each frame, T_{infer} is the model inference time, T_{post} is the postprocessing time, $T_{tracker}$ is the time taken by tracker, T_{event} is the time to call API to send event, T_{ui} is the time to render bounding box on UI and T_{total} is the total latency of the system.

3.5 Model Conversion

To run YOLOv11s on Android efficiently, the trained model is converted to TensorFlow Lite (TFLite). The conversion pipeline first exports the trained YOLO model from the training framework to an interoperable representation such as saved model or ONNX. And later convert to a TFLite flatbuffer (.tflite) using the TensorFlow Lite converter. Optimization techniques such as post-training quantization can also be applied to reduce model size and improve CPU/NPU throughput. TensorFlow Lite supports several post-training quantization modes, including dynamic range quantization and full integer quantization. It reduces model size and can improve inference speed on mobile hardware, but at the same time it can cost potential accuracy degradation [15]. Therefore, proposed system does not use any quantization technique.

In the proposed architecture mobile applications compress assets to reduce APK size. However, compressed model files may require special handling at runtime to first uncompress it and then use it for the inference. In EdgeAware system, models can be distributed in a compressed form of .tflite.gz format and decompressed on first run to reduce installation size, depending on deployment constraints.

After conversion, model outputs are validated on a held-out test set by comparing TFLite inference outputs with the original framework outputs. Small numerical deviations are expected, but detection behavior should remain consistent.

3.6 Edge Deployment

The EdgeAware demonstration uses an Android application written in Kotlin. The application performs real-time inference over a live camera stream, renders detections on the preview, aggregates detections into events, and uploads event reports to the backend. This is done using CameraX library that gives us lifecycle-aware camera use cases such as Preview and ImageAnalysis. ImageAnalysis feeds frames to the inference thread while providing a responsive UI [16][17]. There are multiple models available, such as accident and pothole. Users can choose which models to run, and there is a confidence threshold set per model. Thresholding is used to maintain trade-off between sensitivity and false alarms. In a real-world scenario, it can be set depending on region, time of day, etc. Inference results are then decoded, where Non-Maximum Suppression is used to remove redundant detections, such as

ISSN: 3139-3527 | April-June-Issue, Vol. 1, No. 2 (2026) | DOI: 10.66261/fa1cep06

those where there is overlap of bounding box of multiple detections. For each class, the best bounding box is kept, while others are suppressed if their IoU is above a certain threshold.

When driving, if the device is mounted on a moving vehicle, there is a possibility that, a pothole or accident event could be present in many consecutive frames. To avoid reporting these multiple times, there is a tracker present that groups detections together to provide unique instances using IoU overlap between consecutive frames, as shown in Algorithm 1.

Algorithm 1 provides pseudocode for event tracking and de-duplication.

Algorithm 1: IoU-based event tracking and de-duplication
Input: detections D_t for frame t, IoU threshold τ, age of track A
State: active tracks $\{T_i\}$ with last boxes for respective tracks $\{BBox_i\}$ and age of tracks $\{Age_j\}$
1. For each detection d in D_t:
   Find track T_j with maximum IoU($BBox_j$, $BBox_d$) as IoU_{max}
   If $IoU_{max} \geq \tau$: update BBox of T_j with $BBox_d$
   Else: create new track with $BBox_d$
2. Increment age for all tracks; if age > A, remove track from active tracks

Rather than sending entire videos, the client sends a single frame for each completed event, encrypts it along with metadata in then send it. This saves bandwidth by a factor of approximately N/K, where N is the number of frames that would have been transmitted due to event detection and K is the number of unique events. The proposed system uses the AES-GCM algorithm with a fixed secret key and changing payload for each event, which is recommendations for the security of resource-constrained edge devices [3][4].

3.7 Backend System

The backend uses Python's Django framework with a REST API and a PostgreSQL database and a web-based dashboard for municipal-level monitoring. Each event contains information such as threat/event type such as accident or pothole, date and time, geo-location such as latitude and longitude, encrypted image frame evidence in base64 format, and encrypted device information such as device id, model, IP address, etc. PostgreSQL is used for reliability and structured querying. Common details are stored as independant relational columns, and metadata is stored as JSONB [18].

The web-based dashboard is intended for a report administrator such as a municipal or government-level monitoring organization. Global Admin can configure report admin for administrative areas using various fields such as city, district, sub-district. Respective report admin can monitor and validate events in their area. Using HERE Maps, events are shown as markers on a map. The report administrator may open each event to view evidence frame and validate it. Valid events can be dispatched to police, hospital/ambulance, or road maintenance. And invalid events will be removed from dashboard. The initial design will be a simple dispatch system, and future extensions may include automated systems for response coordination.

Interdisciplinary Journal of AI, Machine Learning & Data Science (IJAIMLDS)
ISSN: 3139-3527 | April-June-Issue, Vol. 1, No. 2 (2026) | DOI: 10.66261/fa1cep06

4. RESULTS AND ANALYSIS

4.1 Object Detection Model Quality

The performance evaluation of object detection models was done by comparing various metrics for performance evaluation. Precision measures how many identified threats are correct to manage low false alarms using Equation (1) and Recall measures how many true threats are identified to manage low missed detection using Equation (2). mAP@0.5 represents localization and classification model performance at a specified IoU threshold of 0.5, while mAP@0.5:0.95 is a stricter evaluation that averages model performance across multiple IoU thresholds from 0.5 to 0.95 and is a better indicator of box localization quality. Moreover, internal calculation of IoU is done using Equation (3). In addition, model size is also reported for edge device suitability assessment. Table 4 represents various metrics for performance evaluation of proposed accident and pothole detection models.

Table 4 Model performance evaluation metrics

Performance Metric	Accident Model	Pothole Model
Precision (%)	90	97.4
Recall (%)	91.2	87
mAP@0.5 (%)	93.7	93.7
mAP@0.5:0.95 (%)	75.6	81.1
Model Size (MB)	19.3	19.3

Both models showed high accuracy, which is appropriate for real-time smart city monitoring applications. While pothole model is more conservative and precise, the accident model is slightly more accurate in detecting true events which represented higher recall. This is beneficial for a multi-event monitoring system such as EdgeAware, where events have different operational priorities.

The performance of the model is closely monitored using critical performance analysis tools such as PR Curve, F1 Curve, and Confusion Matrix, etc. Fig. 6 shows the visualization of these metrics.

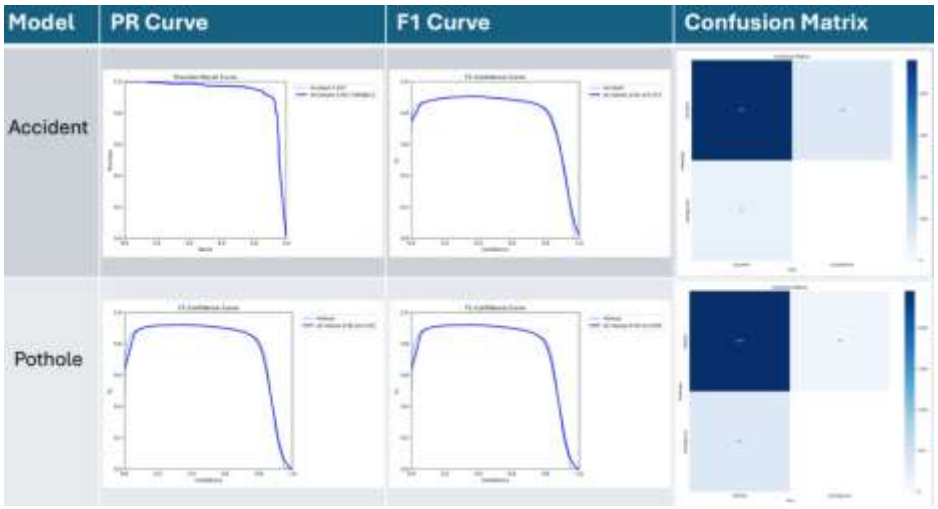

Fig. 6 Results from performance analysis tools PR Curve, F1 Curve and confusion matrix for Accident and Pothole Detection Model

Interdisciplinary Journal of AI, Machine Learning & Data Science (IJAIMLDS)
ISSN: 3139-3527 | April-June-Issue, Vol. 1, No. 2 (2026) | DOI: 10.66261/fa1cep06

From the PR curves of both models, there is strong detection behavior, as indicated by high precision values at a wide range of recall values, which is also supported by high mAP@0.5 scores of 0.937 for both models. From the F1-confidence curves, moderate confidence values offer a good balance between false alarm rates and missed detections, with optimal operating points at 0.37 (Accident, F1≈0.91) and 0.35 (Pothole, F1≈0.92).

From the confusion matrix of the accident model, there are high correct detection rates (TP = 961), fewer missed accidents (FN = 67), but also some false alarm rates (FP = 130). For the pothole model, there are very high correct detection rates (TP = 2453), fewer false alarm rates (FP = 96), but also more missed pothole instances (FN = 326), which is also indicated by a lower recall value but also a high precision value.

Qualitatively, both models offer stable results, which are also visually accurate, especially in normal driving scenarios, with some help from non-maximum suppression to remove redundant bounding boxes. The overall performance of the model is satisfactory for different viewpoints, providing consistent detections as shown in the Fig. 7.

Fig. 7 Sample detections from accident and pothole detection model with true label vs predicted label comparison showing qualitative results of proposed models

However, the performance of the model may be affected in extreme weather conditions, for example, during rainy, foggy, or nighttime weather. For accident detection, it may be challenging during dense traffic or when vehicles suddenly stop. For pothole detection, there might be confusion with small potholes, road shadows, worn areas, and water puddles. Fig. 8 shows the images under extreme weather conditions where model could not detect the accident events correctly. The first image in the Fig 8 has night vision in snowy condition and second image in blurry, foggy condition gives less context to the model to understand the scene, while third image has clear visual but due to the occlusion created by the pole model cannot distinguish the normal vehicle movement and accident event.

Interdisciplinary Journal of AI, Machine Learning & Data Science (IJAIMLDS)
ISSN: 3139-3527 | April-June-Issue, Vol. 1, No. 2 (2026) | DOI: 10.66261/fa1cep06

Fig. 8 True label images containing accident events which model cannot detect properly

Similarly, Fig. 9 shows the different scenarios where pothole model failed to detect the feature. First image in the Fig 9 shows the pothole under the shadow which is not clearly visible to the model, second image is little far away from the vehicle which does not provide the complete context of pothole for the detection model, but in subsequent frames pothole becomes clearly visible to detect the same correctly, while in the third image pothole is occluded by the car wiper because of which model could not detect the pothole correctly.

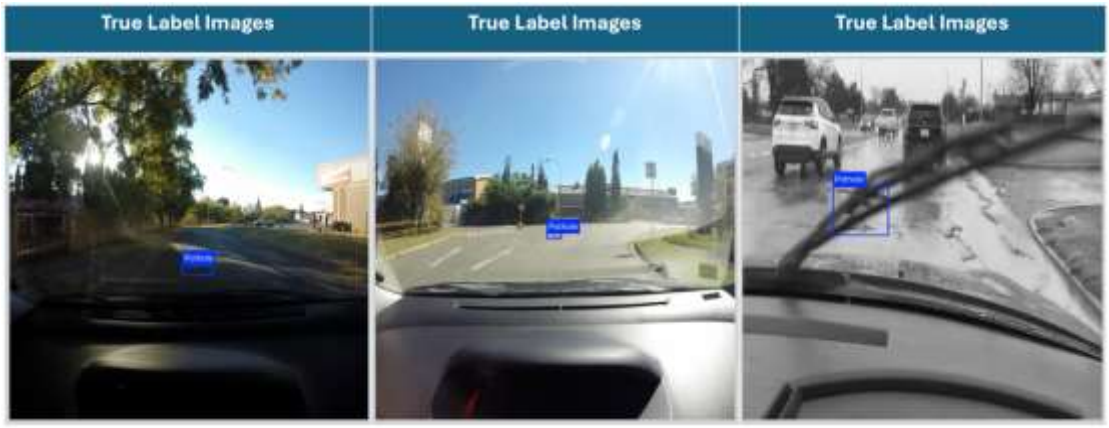

Fig. 9 True label images containing pothole events which model cannot detect properly

4.2 Edge Performance and Benchmarks

The deployment of the edge was also demonstrated by deploying an android application, which showed that the proposed system can meet certain criteria for the edge, making it applicable for use in the real world. Table 5 shows the performance metrics of the android app, which was used for the demonstration. For an input of 640 x 640, the average TFLite inference is ~90 ms. The end-to-end runtime latency is about 1.5 s, which includes camera capture, preprocessing, model inference, NMS postprocessing, IoU tracking, UI rendering, and event ingestion. The confidence thresholds were set at 0.37 for Accident and 0.35 for Pothole, as seen in the F1 curve near the best F1 operating points, which provide a good balance of false alarms and missed detections. For storage and distribution, the compressed models resulted in a reduction of the app's total size from 179 MB to 167 MB, with individual model sizes reduced from 38 MB to 32.6 MB, making it easy for the app to install.

ISSN: 3139-3527 | April-June-Issue, Vol. 1, No. 2 (2026) | DOI: 10.66261/fa1cep06

Table 5 performance metrics achieved through the android app

Parameter	Value
End-to-end runtime latency	1.5 s
Average inference time of TFLite model	90 ms
Input resolution	640 × 640
Confidence threshold (Accident)	0.37
Confidence threshold (Pothole)	0.35
App size (with model compression)	167 MB
App size (without model compression)	179 MB
TFLite model size (each, uncompressed)	38 MB
TFLite model size (each, compressed)	32.6 MB

Edge Aware also supports other edge benchmark requirements such as bandwidth efficiency, security, privacy, and scalability. Bandwidth efficiency is attained through event-level reporting, where instead of transmitting video, only one representative frame is transmitted per unique threat event, minimizing bandwidth usage due to continuous heavy upload, and enabling operation even in low connectivity scenarios. Security and privacy are attained through encryption of the frame and metadata prior to transmission, ensuring that sensitive information is not compromised during network transfer. Resources were saved through local inference, thus minimizing cloud computing requirements, and through reduced events detected from multiple frames in the video stream using IoU tracking, minimizing repeated events, repeated uploads, and backend requirements. Finally, scalability is attained through operation, where multiple edge devices operate independently, thus creating a validated event, and minimizing server requirements, map/dashboard clutter, and enabling teams within a municipality to operate over larger regions with fewer false positives and manageable network/storage requirements.

4.3 Backend Monitoring Workflow Discussion

A key improvement in EdgeAware was the explicit validation workflow designed for municipal monitoring organizations. Many academic prototypes showed detection and notification of independent events; but, for efficiency of systems, human-in-the-loop verification is required to avoid wasting resources because of false alarms and increase stakeholder trust. The implemented system included the monitoring interface for a login page providing authenticated access as shown in the Fig. 10 and a dedicated monitoring page where reported events were displayed and managed, as showed in the Fig 11.

Interdisciplinary Journal of AI, Machine Learning & Data Science (IJAIMLDS)

ISSN: 3139-3527 | April-June-Issue, Vol. 1, No. 2 (2026) | DOI: 10.66261/fa1cep06

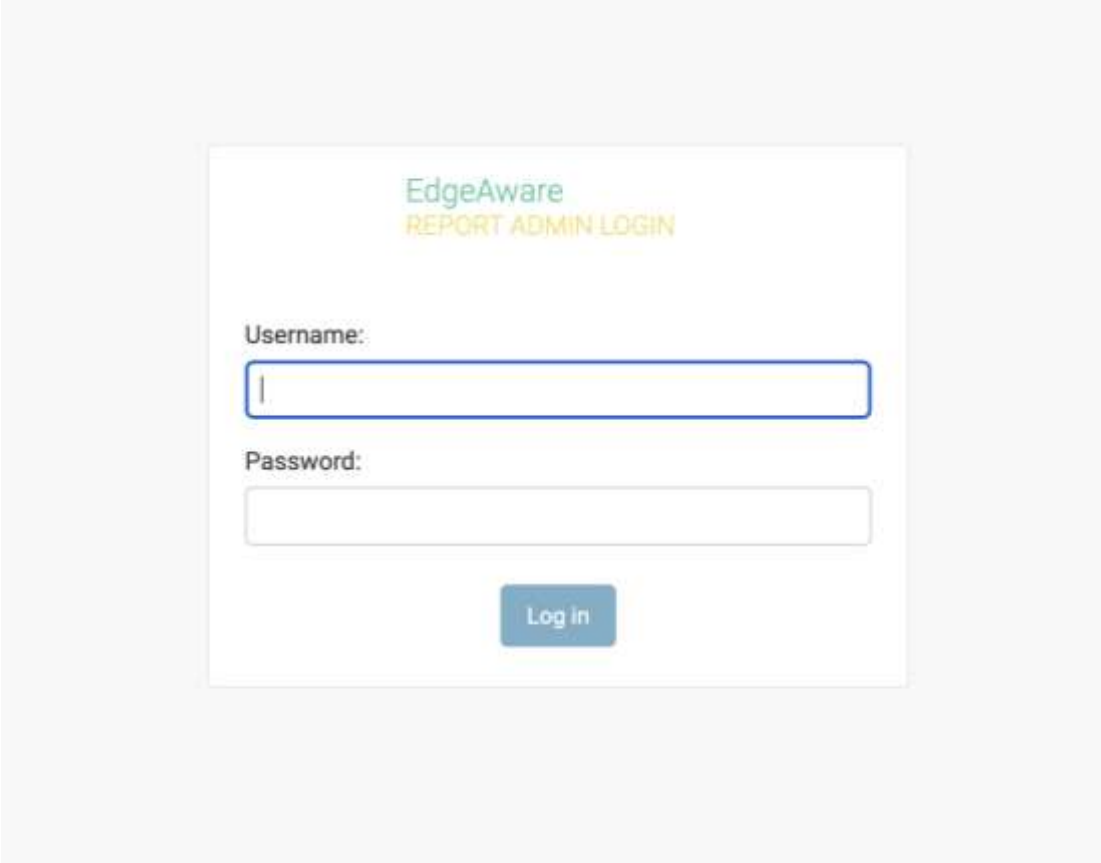

Fig. 10 Login page of EdgeAware for authenticated access of administrators

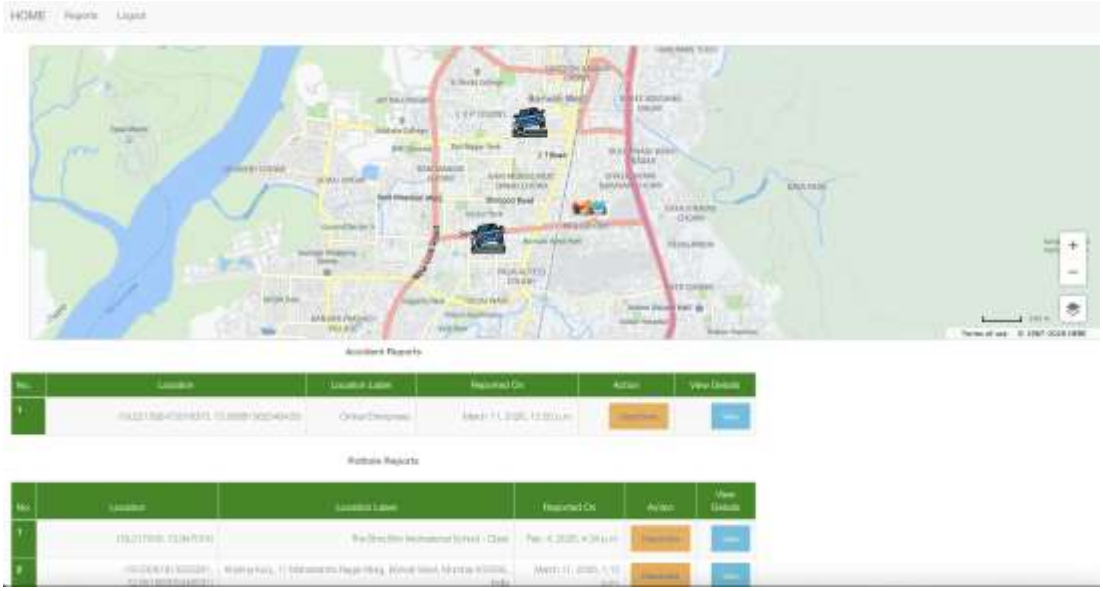

Fig. 11 Event monitoring dashboard page of EdgeAware with incident markers placed on the map for reported events

ISSN: 3139-3527 | April-June-Issue, Vol. 1, No. 2 (2026) | DOI: 10.66261/fa1cep06

For the validation of event, the administrator can review the uploaded evidence frames and invalidate incorrect detections by clicking on 'Deactivate' button showed in the Fig. 11. This addressed a limitation highlighted in literature survey, where false positives occurred in static scenes or visually confusing patterns needs to validate [9]. Post validation, events could be prioritized based on severity and priorities, such as major collisions vs minor stoppages, or deep potholes vs surface-level damage or small cracks, and by location criticality, for example highways and high-traffic junctions vs residential roads.

Integration with response teams is the critical part of the end-to-end system, the demonstrated version supported manual coordination by administrators, while future extensions could automate response coordination through contact directories or dispatch APIs. Emergency management literature had emphasized layered architectures where sensing triggered alerts and mitigation actions [5]; EdgeAware had followed this structure while reducing upstream bandwidth and storage overhead by performing inference at the edge and transmitting only compact, event-level evidence rather than continuous video streams.

4.4 O^2I^2 Results

In Table 6 O^2 I^2 analysis, the outcome represented the practical performance and deployment gains delivered by the proposed system.

Table 6 O^2I^2 results table

Technical Parameter	Output	Baseline output from existing approach	Outcome	Impact		Improvement %
				With Proposed Methodology	Without Proposed Methodology	
mAP@0.5 (Accident) %	93.7	**87**	**6.7;** used **multi-view training** and **robust augmentations** to improve localization and generalization	More accurate detection and bounding boxes for accidents	Lower localization and generalization in varied scenes	**7.7**
mAP@0.5 (Pothole) %	93.7	**83.3**	**10.4;** achieved using **multi-view road imagery** and **hard-negative exposure** (road textures/patches) to improve pothole discrimination and box quality	Stronger pothole detection across different road surfaces	More failures on new road types and viewpoints	**12.4**
Precision	90.0	**80**	**10.0;** achieved	Fewer false	More false	**12.5**

ISSN: 3139-3527 | April-June-Issue, Vol. 1, No. 2 (2026) | DOI: 10.66261/fa1cep06

(Accident) %			using **confidence-threshold tuning (0.37)** and **NMS postprocessing** to suppress low-quality and duplicate boxes	accident alerts and cleaner monitoring	alarms cause alert fatigue	
Precision (Pothole) %	97.4	**84**	**13.4;** used **confidence-threshold tuning** and **training with diverse road backgrounds** to reduce false triggers on non-pothole textures	Highly reliable pothole reports with minimal noise	More false pothole reports from shadows/patches	**15.95**
Recall (Accident) %	91.2	**80**	**11.2;** achieved using **multi-view coverage** and **augmentation for occlusion/lighting variation**, enabling detection of more true accident instances	Fewer missed accidents and better safety monitoring	Higher miss rate delays response	**14**
Recall (Pothole) %	87.0	**81.2**	**5.8;** achieved using **multi-view and small/partial pothole exposure** and **lower optimal threshold (0.35)** to improve sensitivity	Better pothole coverage for maintenance planning	More potholes missed, especially small ones	**7.14**
TFLite model size MB	32.6	**38**	**5.4;** achieved using **model compression in the Android package** (compressed	Reduced storage cost and faster distribution	Higher footprint limits scalable rollout	**14.21**

ISSN: 3139-3527 | April-June-Issue, Vol. 1, No. 2 (2026) | DOI: 10.66261/fa1cep06

			storage and runtime decompression), reducing footprint without changing inference pipeline			
Android app size MB	167	**179**	**12;** achieved by storing **compressed TFLite assets** and unpacking at runtime, reducing installation/update overhead	Easier deployment across many devices	Larger app reduces adoption and increases update friction	**6.7**

4.4.1 Overall Validation:

As a result, with EdgeAware's proposed methodology, the system was able to achieve better-quality detections with enhanced mAP, precision, recall, and a more deployable edge package with a smaller model size and app size. It is evident that EdgeAware surpassed baseline performance for both accident and pothole detection quality, as well as deployment ability on mobile edge devices. The accuracy of the system was significantly enhanced, especially in terms of pothole precision and mAP, for reliable multi-event monitoring. In addition, model compression resulted in a 14.21% smaller size for the TFLite model and a 6.70% smaller size for the app, directly enhancing storage efficiency and deployment, especially in a scale where updates and models are frequently needed.

4.5 Future Enhancements

EdgeAware is designed as a deployable prototype, but some limitations motivate future research and engineering. Literature repeatedly notes degraded performance under fog, night scenes, and occlusion [9][10]. Future work should include targeted data collection for nighttime, rain, fog, and heavy traffic, along with domain adaptation techniques. Multi-modal sensing such as audio, accelerometer, radar can also reduce ambiguity. A practical smart-city platform should monitor additional events such as vehicle fire, flooding, fallen trees, illegal parking obstruction, and crowd hazards. This extension can be achieved easily by adding more detectors for different events or by adopting a modular multi-task learning approach to detect different events in single model.

The next milestone is rigorous field testing across varied routes and camera placements, followed by statistical evaluation of event-level detection rate, false alarm rate, and dispatch latency. This can also include the automation of coordination with response teams such as ambulance, police, fire departments.

5. DISCUSSION AND CONCLUSION

ISSN: 3139-3527 | April-June-Issue, Vol. 1, No. 2 (2026) | DOI: 10.66261/fa1cep06

The proposed system is designed to be used in smart cities through its ability to provide real-time intelligence to detect threat events such as accident, pothole based on its edge computing capabilities integrated with vision-based solution, along with its centralized municipal monitoring process. The system was developed based on several shortcomings reported in existing approaches, including reliance on cloud computing, robustness of viewpoints, single-event monitoring, and lack of operational validation processes. To address these shortcomings, EdgeAware utilized YOLOv11s detectors trained on multi-view images for accidents and potholes, integrated them on Android application using TensorFlow Lite for scalable sensing of events through vehicle-mounted phone cameras. It also supports the adoptability in other devices such as UAVs, surveillance and other cameras.

The experimental results of this system showed that it is effective in detecting accidents and potholes with an mAP@0.5 of 93.7%, precision of 90.0% and 97.4%, whereas recall of 91.2% and 87.0% for accident and potholes respectively. These are reliable detection capabilities ready for smart cities, where false alarms are not permissible to provide public and infrastructure safety. The overall edge performance was good since the average inference time was ~90 ms. The overall latency of ~1.5 s included the camera capture, inference, tracking, and event ingestion also reached benchmark for edge deployment. The storage issues associated with edge devices were addressed by compressing the model. This reduced the TFLite model size to 32.6 MB and the overall app size to 167 MB and made the app deployment easier on resource-constrained edge devices.

Besides the detection features, EdgeAware included an IoU-based tracking system. This allowed the selection of a representative frame for each event to enhance the operational feasibility. Additionally, the secure transmission of the selected frame along with the event metadata using the encryption technique was included. The backend system used the Django web development framework along with the PostgreSQL database to provide visualization of the event using HERE maps. EdgeAware provided a secure architecture to perform multi-event monitoring using edge computing.

ACKNOWLEDGMENTS

The authors declare that no financial or institutional support was received for this research.

CONFLICT OF INTEREST STATEMENT

The authors declare no conflict of interest.

REFERENCES:

[1] A. Shaji George, A. S. Hovan George, and T. Baskar, “Edge Computing and the Future of Cloud Computing: Industry Perspectives and Predictions,” Partners Universal International Research Journal, vol. 2, no. 2, pp. 203–209, 2023, doi:.

[2] D. Prasad, A. Sharma, M. A. Kadhum, M. K. Khan, M. Kahani, and P. K. R. Maddikunta, “Edge Computing Technology Enablers: A Systematic Lecture Study,” in Enablers for

ISSN: 3139-3527 | April-June-Issue, Vol. 1, No. 2 (2026) | DOI: 10.66261/fa1cep06

Industry 4.0: Role of Artificial Intelligence and Machine Learning, Lecture Notes in Networks and Systems, vol. 832. Cham, Switzerland: Springer Nature, 2024, pp. 55–94

[3] A. M. Sheikh, M. R. Islam, M. H. Habaebi, S. A. Zabidi, A. R. Najeeb, and A. Kabbani, "A Survey on Edge Computing Security Challenges: Classification, Threats, and Mitigation Strategies," Future Internet, vol. 17, no. 4, p. 175, 2025, doi: 10.3390/fi17040175.

[4] I. Rozlomii, M. B. Banerjee, A. Bhattacharya, and K. K. Chari, "Innovative Resource-Saving Security Strategies for IoT Devices," Journal of Edge Computing, vol. 4, no. 1, pp. 1–24, 2025

[5] D. G. Costa, D. F. Peixoto, A. D. De Souza, and J. P. D. A. Lima, "A Survey of Emergencies Management Systems in Smart Cities," IEEE Access, vol. 10, pp. 618–648, 2022

[6] N. Pathik, R. K. Gupta, Y. Sahu, A. Sharma, M. Masud, and M. Baz, "AI Enabled Accident Detection and Alert System Using IoT and Deep Learning for Smart Cities," Sustainability, vol. 14, no. 13, p. 7701, 2022, doi: 10.3390/su14137701.

[7] K. Nandini, P. S. V. P. Ashok, and A. P. Jeyananthan, "Real-Time Accident Detection and Emergency Response System Using CCTV Video Analysis," Industrial Engineering Journal, vol. 54, no. 3, pp. 376–380, 2025.

[8] S. Aslam, M. Muzammal, M. A. Shah, A. A. Khan, A. E. A. Ibrahim, and M. M. Faroug, "An IoT-based Automatic Vehicle Accident Detection and Visual Situation Reporting System," Journal of Advanced Transportation, vol. 2024, Art. ID 4719669, 2024, doi: 10.1155/2024/4719669.

[9] V. A. Adewopo and N. Elsayed, "Smart City Transportation: Deep Learning Ensemble Approach for Traffic Accident Detection," IEEE Access, vol. 12, pp. 59134–59146, 2024

[10] P. Kumar and D. R. Tiwari, "SAFE ROAD AI: Real-time Accident Detection from Multi-angle Crash Videos – A Review," International Research Journal of Engineering and Technology (IRJET), vol. 11, no. 9, pp. 128–134, 2024.

[11] E. Lieskovská, M. Jakubec, B. Bučko, and K. Zábovská, "Automatic Pothole Detection," Transportation Research Procedia, vol. 74, pp. 1164–1170, 2023, doi: 10.1016/j.trpro.2023.11.257.

[12] A. R. Addanki and J. Lin, "Pothole Detection with YOLOV8," Yeshiva University, Technical Report, Dec. 2023.

[13] J. Luo et al., "Efficient Small Object Detection You Only Look Once: A Small Object Detection Algorithm for Aerial Images," Sensors, vol. 24, no. 21, p. 7067, 2024, doi: 10.3390/s24217067.

[14] R. Khanam and M. Hussain, "YOLOV11: An Overview of the Key Architectural Enhancements," arXiv:2410.17725, Oct. 2024.

[15] TensorFlow Model Optimization, "Quantization aware training – Model optimization," TensorFlow Documentation, 2024. (Accessed: Jan. 31, 2026).

[16] Android Developers, "CameraX overview," Android Developer Documentation. (Accessed: Dec. 11, 2025).

[17] Android Developers, "CameraX ImageAnalysis use case," Android Developer Documentation. (Accessed: Dec. 11, 2025).

ISSN: 3139-3527 | April-June-Issue, Vol. 1, No. 2 (2026) | DOI: 10.66261/fa1cep06

[18] PostgreSQL Global Development Group, “JSON types,” PostgreSQL Documentation. (Accessed: Jan. 13, 2026).