IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 14, Issue 12, December 2025

DOI: 10.17148/IJARCCE.2025.141203

Developing an AI-based smart traffic control system for emergency vehicles and congestion management

Ananya[1], Darshan M[2], Darshan R[3], Inchara R[4], Uma S[5]

Student, ECE, SJB Institute of Technology, Bengaluru, India[1]

Student, ECE, SJB Institute of Technology, Bengaluru, India[2]

Student, ECE, SJB Institute of Technology, Bengaluru, India[3]

Student, ECE, SJB Institute of Technology, Bengaluru, India[4]

Assistant Professor, ECE, SJB Institute of Technology, Bengaluru, India[5]

Abstract: Rapid identification of emergency vehicles is crucial for enabling timely intervention and reducing the risk of accidents in urban traffic environments. Accidents are inevitable in our daily life. To minimize the occurrence of accidents, implementing efficient and well-managed traffic systems is essential. A surveillance system called smart detection of emergency vehicles can identify emergency vehicles that are stuck in traffic. This system supports smarter traffic management in response to the growing number of vehicles on the road in recent years, which has led to increasing congestion. In this paper, we present a prototype of a traffic control system designed to manage signal lights at a junction. When an emergency vehicle approaches, the system temporarily overrides the normal signal cycle and prioritizes its movement by indicating its entry. This ensures that the emergency vehicle can pass through the junction in the shortest possible time. When an emergency vehicle enters, the system will stop the present status of work temporarily and will indicate the entry of the emergency vehicle. So that it can pass through the junctions in a lowest possible time. To achieve this we have utilised a Camera module which helps in identifying or capturing the real time images in the traffic, this camera module is paired with Raspberry Pi 4 as the main controller, the software is implemented in Python, utilising the YOLO framework, and the final implementation is done through a led module which helps in regulating the traffic signals. Since it the proto type the desired implementation is done for only one of the line of the traffic junction. Keywords: Machine Learning, Image Processing, Segmentation, Early Detection, Artificial Intelligence.

I. INTRODUCTION

The Emergency Vehicle Detection and Smart Traffic Control System is an advanced solution designed to provide real-time priority to emergency vehicles such as ambulances, fire trucks, and police vans. The system integrates computer vision, machine learning, IoT, and embedded automation to improve traffic management and reduce emergency response times. Traditional traffic signal systems operate on fixed-time or sensor-based schedules that cannot dynamically identify emergency vehicles. This often leads to unnecessary delays, especially in congested areas. The proposed system addresses these limitations by integrating YOLOv8 object detection to identify emergency vehicles from a live video stream, which is processed using a Raspberry Pi 4. Once an emergency vehicle is detected, the system automatically switches the traffic signal to green, enabling the vehicle to pass through the junction without delay. This dashboard provides a web interface for live monitoring, data logging, and signal status visualization, allowing users or traffic authorities to supervise system operations from any connected device. The dashboard adds intelligence, transparency, and usability to the project, transforming it into a complete IoT-enabled smart traffic management platform.

1.1 Motivation of Work

Emergency response time is a critical factor that can determine whether lives are saved in urgent situations. In densely populated cities, traffic congestion often prevents emergency vehicles from reaching their destinations quickly. Manual traffic control during such emergencies is not always feasible, particularly at multiple intersections. The motivation behind this project is to develop an automated, low-cost, and intelligent traffic management system capable of detecting emergency vehicles in real time and adjusting traffic signals accordingly. By utilizing deep learning (YOLOv8) and Raspberry Pi–based automation, the system ensures smooth passage for emergency vehicles without the need for human intervention. Additionally, the inclusion of a real-time dashboard enhances operational efficiency by providing live

DOI: 10.17148/IJARCCE.2025.141203

monitoring and analytical insights.

Objective of work

- To detect emergency vehicles in real-time using the YOLOv8 object detection model.
- To automatically override the current traffic signal and provide a green signal to the emergency lane.
- To resume normal traffic signal operation after the emergency vehicle exits the frame. To provide a web-based dashboard for live monitoring, visualization, and system control.
- To log detection events, timestamps, and signal changes for future analysis. To achieve real-time performance and high detection accuracy using a resource-efficient embedded device.

II. LITERATURE REVIEW

1. **Smart Traffic Management Systems (Rayala et al., 2024):**

   Rayala et al. present an advanced traffic management framework that dynamically adjusts traffic signals based on real-time vehicle density and traffic flow conditions. The system incorporates path-optimization techniques, including Dijkstra's shortest-path algorithm, to determine the most efficient routes for emergency vehicles. A key enhancement in their work is the integration of YOLO-based object detection to identify ambulances and initiate automatic priority clearance. Once an emergency vehicle is detected, the system predicts its route and modifies signal timings along that path to reduce waiting time and ensure uninterrupted passage. This approach demonstrates that combining perception (object detection) with decision-making (routing) algorithms can effectively minimize travel delays and significantly improve emergency response times in congested urban environments.

2. **OpenCV-Based Detection (Bhargavi et al.):**

   Bhargavi et al. utilize classical computer-vision techniques with OpenCV to detect emergency vehicles by analyzing visual cues such as color patterns, contours, and shape features. This approach is computationally lightweight and suitable for low-power embedded platforms; however, its performance is highly dependent on external factors like lighting conditions, camera quality, and occlusion. Although the method offers simplicity and low processing delay, it suffers from reduced accuracy in complex urban scenarios where visual inconsistencies such as faded markings, glare at night, or partial obstruction can lead to misclassification. Consequently, the study emphasizes the limitations of traditional vision-based approaches when compared to deep learning models, which provide higher robustness and accuracy.

3. **IoT + Drone-Based Architectures (Farahdel et al., 2022):**

   Farahdel et al. describe a large-scale intelligent traffic management architecture that merges IoT-enabled ground sensors with airborne drone surveillance to achieve comprehensive situational awareness. Drones equipped with cameras and sensors provide real-time aerial views of road congestion, accident sites, and blocked pathways, enabling systems to react more intelligently to live conditions. However, the authors emphasize technical challenges related to power consumption, communication bandwidth, and the reliability of long-range wireless connectivity. Their results demonstrate that UAV-augmented traffic monitoring can significantly enhance coverage and accuracy in expanding smart-city ecosystems, but must be optimized to overcome resource constraints.

4. **Audio-Visual Fusion (Islam & Abdel-Aty, 2021):**

   Islam and Abdel-Aty propose a dual-modality detection approach in which visual detection systems are supported by audio-based siren recognition. Since visual detection alone can fail in fog, nighttime scenarios, or when vehicles are partially occluded, audio-frequency analysis provides an additional layer of reliability. The system uses signal-processing algorithms to detect unique siren frequency patterns characteristic of ambulances and fire trucks, enabling early recognition even before the vehicle enters camera range. Their findings show that fusion-based approaches significantly outperform single-modality detection, making them beneficial for dense and unpredictable urban environments.

5. **Multimodal Deep Learning (Zohaib et al., 2023):**
   Zohaib et al. broaden the idea of audio-visual fusion by introducing a multimodal learning framework that integrates

DOI: 10.17148/IJARCCE.2025.141203

vision, sound, wireless telemetry, and contextual data such as GPS movement patterns. Their deep neural network processes these diverse inputs in parallel and applies advanced sensor-fusion techniques to enhance emergency vehicle detection in challenging conditions, including heavy traffic, complex road layouts, and partial visibility. The study shows that multimodal AI systems offer greater reliability, robustness, and adaptability than single-sensor or traditional methods, making them highly suitable for next-generation smart mobility solutions.

6. **Raspberry Pi-Based Junction Control (Mahesh Kumar & Kumaraswamy, 2020):**

This study demonstrates the feasibility of implementing them using Raspberry Pi as a low-cost and scalable hardware controller for real-time traffic signal management. By continuously analyzing sensor input thresholds such as vehicle count or signal triggers, the system can modify traffic light timings accordingly. The Raspberry Pi directly interfaces with GPIO-controlled relays that actuate signal lamps, illustrating a practical prototype implementation for smart junctions. The success of this study reinforces the importance of embedded edge-processing capability, reducing dependency on centralized cloud systems and ensuring faster decision-making for real-time traffic control.

7. **YOLO (Redmon et al., 2016):**

Redmon et al. introduced YOLO (You Only Look Once), a landmark object-detection framework that treats detection as a single regression task rather than a multi-stage pipeline. By predicting bounding boxes and class probabilities in a single pass, YOLO achieves very high processing speeds, making them suitable for real-time environments such as autonomous driving and surveillance. The continuous improvements seen in later versions (YOLOv3, v5, v8) have significantly boosted detection accuracy and inference efficiency, making YOLO one of the most extensively adopted models for embedded and edge-based systems. Its performance reliability and low latency make it a core enabler in smart emergency vehicle detection and intelligent traffic systems.

III. DESIGN AND IMPLEMENTION

The proposed system utilizes a Raspberry Pi 4 Model B equipped with 4 GB of RAM as the primary edge computing device for real-time inference. 30 FPS video feed of traffic lane is captured using USB camera. Three LEDs simulate traffic light via GPIO. To ensure stable operation and sufficient storage for the dataset and trained models, the hardware configuration includes a 5V, 3A power supply and a 32 GB Micro SD card. The software architecture is developed using Python 3.10, leveraging the Ultralytics YOLOv8 model to achieve a trade-off between detection accuracy and computational complexity. Image preprocessing and model execution are managed through the OpenCV and PyTorch libraries, respectively. For embedded deployment, the system employs ONNX Runtime for optimized inference and integrates with the Blynk IoT platform or MQTT to facilitate remote defect alerting.

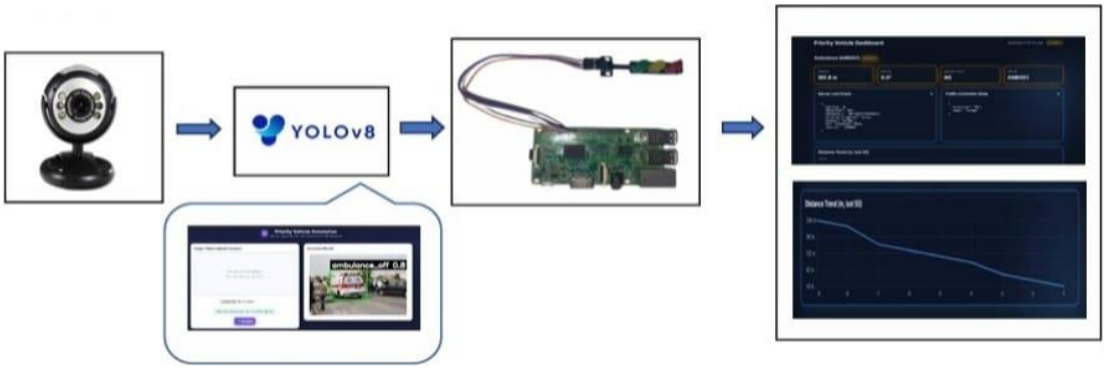

Fig 3.1.1 Block Diagram

1. Data Acquisition

In this initial phase, a high-resolution camera module is continuously mounted at the intersection to capture real-time video streams. The camera records frames at a fixed rate, ensuring that vehicle movement, density, and lane activity are accurately captured. These frames serve as raw visual input for the detection system. Proper lighting conditions, camera positioning, and frame rate optimization are essential to ensure reliable performance in varying traffic environments.

2. Detection Phase

- The captured frames are processed using the YOLOv8 (You Only Look Once, Version 8) deep learning model deployed either on the Raspberry Pi or on an external GPU server.

International Journal of Advanced Research in Computer and Communication Engineering
Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 14, Issue 12, December 2025
DOI: 10.17148/IJARCCE.2025.141203

- YOLOv8 performs: Object.
- Detection: Identifies different vehicle types such as cars, buses, trucks, two-wheelers, and emergency vehicles.
- Classification: Distinguishes emergency vehicles (ambulance, fire engine, police vehicle) using trained model classes.
- Density Analysis: Counts the number of vehicles in each lane to estimate congestion levels.
- The model generates bounding boxes, class labels, and confidence values for every object it identifies. These outputs are then used to drive real-time adjustments in the traffic signal system.

3. Control Phase

- Based on the detection results, the Raspberry Pi runs decision-making algorithms that regulate the operation of the traffic lights.
- Key operations include:
- GPIO Signal Control: The Pi triggers corresponding GPIO pins to switch traffic lights between red, yellow, and green.
- Emergency Vehicle Priority: When an emergency vehicle is detected, the system immediately overrides normal operation, turning the relevant lane green.
- Adaptive Timing: If congestion is detected, the green-light duration is automatically increased for the busiest lane.
- This phase ensures real-time, intelligent signal adjustments without manual intervention.

4. Dashboard Visualization

- A Flask-based Web Dashboard provides a centralized interface for monitoring system activity.
- The dashboard displays:
- Live Camera Feed: Real-time video with detection bounding boxes.
- Current Signal Status: Shows which lane has green, yellow, or red.
- Vehicle Count & Alerts: Logs vehicle density, emergency detections, and control actions.
- System Logs: Time-stamped events for debugging and analysis.
- The Raspberry Pi communicates with the Flask server via HTTP or WebSocket requests, pushing continuous updates for visualization. This phase ensures transparency, remote monitoring, and operational insight for traffic authorities.

IMPLEMENTATION

This phase is essential because it converts the initial design concepts into a functional, testable system. The chapter outlines the development process of the Emergency Vehicle Detection and Smart Traffic Control System, which includes object detection, GPIO-based signal control using the Raspberry Pi, and a Flask-enabled dashboard for live visualization and event logging. The system was built entirely in Python 3, using separate modules for model inference, traffic signal management, and dashboard operation. Each module was validated independently and later integrated to ensure smooth interaction across the detection, control, and visualization components, which combines YOLOv8-based object detection with real-time decision-making algorithms.The design emphasizes modularity, allowing each component to be upgraded or replaced without affecting the overall system. Extensive testing under various traffic conditions, including high congestion and partial occlusions, ensured reliability and robustness. The dashboard not only provides live monitoring but also logs events and performance metrics, facilitating data-driven analysis and optimization. Overall, the system demonstrates a scalable, real-time smart traffic management solution that can be extended to multi-junction control using cloud integration in future enhancements.

Software Environment

Programming Language	:	Python 3.10
Deep Learning Library	:	Ultralytics YOLOv8
Web Framework	:	Flask 3.0
Front-End Technologies	:	HTML5, CSS3, JavaScript (with Socket.IO)
Database	:	SQLite3
Image Processing	:	OpenCV 4.8.0
Hardware Interface	:	RPi.GPIO
Supporting Packages	:	NumPy, Matplotlib, Requests, time, jsonify
IDE / Tools	:	VS Code, Jupyter Notebook

IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 14, Issue 12, December 2025

DOI: 10.17148/IJARCCE.2025.141203

OS Environment : Raspberry Pi OS (Bullseye)

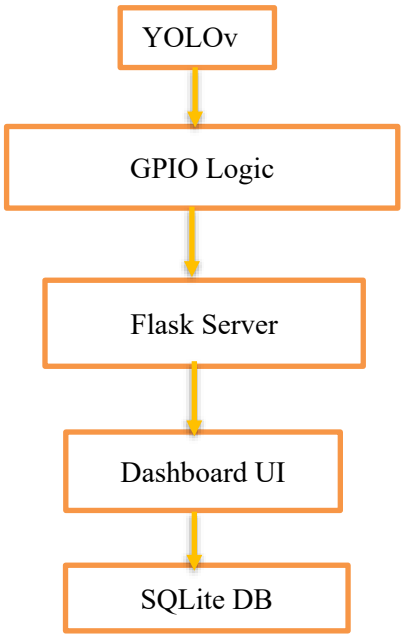

Figure 3.1.2: Software environment

Dataset Preparation

The training dataset consisted of labeled images of **ambulances**, **fire trucks**, and **police vehicles**, captured under varying lighting and weather conditions.

Dataset Details

- Source: **Roboflow** and custom-captured images
- Total Images: ~2,000
- Resolution: 640×640 pixels
- Format: YOLO annotation (TXT)
- Classes: 3 (ambulance, fire_truck, police_vehicle)
- Split Ratio: 70% training, 20% validation, 10% testing

Data Augmentation

To improve generalization:

- Horizontal and vertical flips
- Random rotation and scaling
- Brightness and contrast variation
- Motion blur simulation

The dataset configuration file (data.yaml) defined paths and class mappings for YOLOv8 training.

Model Training Procedure

The YOLOv8 model was initially trained on a GPU-enabled platform to leverage high computational power for faster convergence and accurate learning. Once training was complete, the optimized model was deployed on the Raspberry Pi to perform real-time inference at the edge. This deployment enables low-latency detection of emergency vehicles without

IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 14, Issue 12, December 2025

DOI: 10.17148/IJARCCE.2025.141203

relying on continuous cloud connectivity. Furthermore, the lightweight design of the YOLOv8-nano variant ensures that the model runs efficiently on the resource-constrained Raspberry Pi while maintaining high detection accuracy. Extensive testing under varying traffic conditions confirmed that the system can reliably identify emergency vehicles and trigger timely traffic signal changes.

Training Configuration

Parameter	**:**	**Value**
Model	:	YOLOv8n (Nano)
Epochs	:	50
Batch Size	:	8
Image Size	:	640×640
Optimizer	:	AdamW
Learning Rate	:	0.001
Validation Split	:	20%
Output	:	runs/train/exp/best.pt

Training command:

yolo detect train data=data.yaml model=yolov8n.pt epochs=50 imgsz=640

Model Evaluation:

- mAP@0.5 = **94.1%**
- Precision = **92%**
- Recall = **89%**
- Inference Speed = **25 FPS (CPU)**

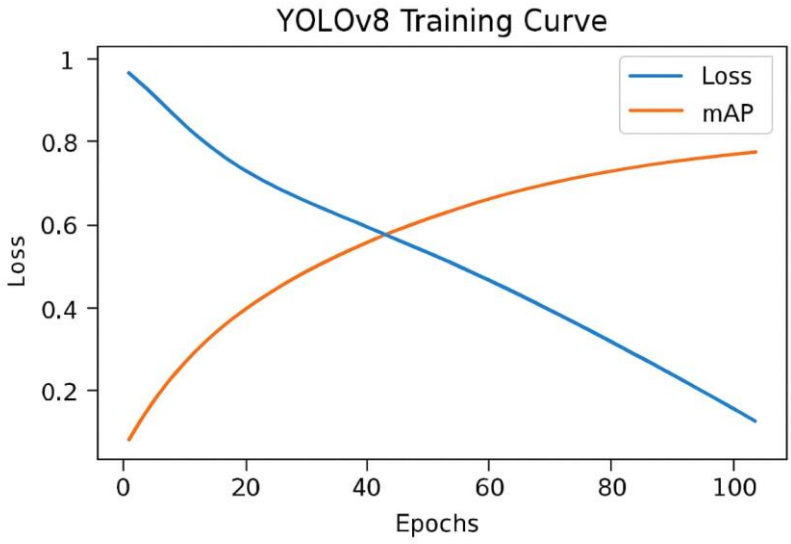

Figure 3.1.3: YOLOv8 Training Process

Loss decreasing steadily; mAP increasing across epochs.

Model Deployment on Raspberry Pi

The trained model (best.pt) was transferred to the Raspberry Pi for real-time inference.

IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 14, Issue 12, December 2025

DOI: 10.17148/IJARCCE.2025.141203

Steps:

1. Install dependencies on Raspberry Pi:
2. pip install ultralytics opencv-python RPi.GPIO flask sqlite3 flask-socketio
3. Copy model weights and Python scripts.
4. Optimize for embedded use by exporting to ONNX:
5. model.export(format='onnx')
6. Load the model in the main script for inference.

```
from ultralytics import YOLO
model = YOLO("model/best.pt")
```

Flask Dashboard Implementation

A major enhancement in this version is the addition of a **Flask-based web dashboard** that displays real-time detection data, camera feed, and signal states.

Dashboard Architecture

- **Backend (Flask App):** Handles routes (/, /update), processes POST data from YOLO module, and manages SQLite logging.
- **Frontend (HTML/CSS/JS):** Displays:
  - Live MJPEG stream from camera
  - Signal indicator (Red/Green status)
  - Detection table (vehicle type, confidence, timestamp)
- **Database (SQLite):** Stores historical detection events.

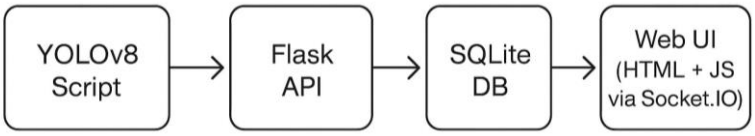

Figure 3.1.4 – Dashboard Architecture Flow

YOLOv8 Script → Flask API → SQLite DB → Web UI (HTML + JS via Socket.IO)

Integration of YOLO and Dashboard

The main detection script (main.py) communicates with the Flask server using HTTP POST requests whenever an emergency vehicle is detected:

```
import requests

if emergency_detected:
    payload = {'vehicle': 'ambulance', 'confidence': 0.94, 'signal': 'GREEN'}
    requests.post("http://127.0.0.1:5000/update", json=payload)
```

This ensures the **dashboard updates instantly** with new detections, while the **GPIO module** changes signal lights simultaneously.

IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 14, Issue 12, December 2025

DOI: 10.17148/IJARCCE.2025.141203

Code Structure and File Organization

```
priority-vehicle-detection/
│
├── model/
│   └── best.pt
│
├── app.py              # Flask dashboard backend
├── main.py             # YOLO detection + GPIO + Flask integration
├── templates/
│   └── dashboard.html  # Frontend UI
├── static/
│   ├── style.css
│   └── script.js
├── detections.db       # SQLite logs
├── requirements.txt
└── dataset/
    ├── train/
    ├── valid/
    
    └── test/
```

Each script is modular and documented. train_yolo.py handles training; main.py handles inference and visualization.

IV. RESULTS

Introduction

This chapter presents the results, performance analysis, and discussion of the developed Emergency Vehicle Detection and Smart Traffic Control System. The evaluation includes detection accuracy, inference speed, signal response timing, and the usability of the dashboard. The integration of the Flask-based real-time dashboard greatly improves system interactivity by providing live monitoring, automatic event logging, and clear visualization of system performance.

Training Performance

The YOLOv8-nano model was trained for **50 epochs** on a custom dataset of emergency vehicles. Training and validation curves showed consistent convergence with reduced loss and improved mean Average Precision (mAP).

Key observations:

Metric	Value	Interpretation
Precision	0.92	High true-positive rates for emergency vehicle.
Recall	0.89	Low false- negative detection.
mAP@0.5	94.1%	Excellent localization accuracy.
mAP@0.5:0.95	71.8%	Robust detection across IoU thresholds.
Inference Speed	25 FPS	Suitable for real-time processing on Raspberry Pi.

IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 14, Issue 12, December 2025

DOI: 10.17148/IJARCCE.2025.141203

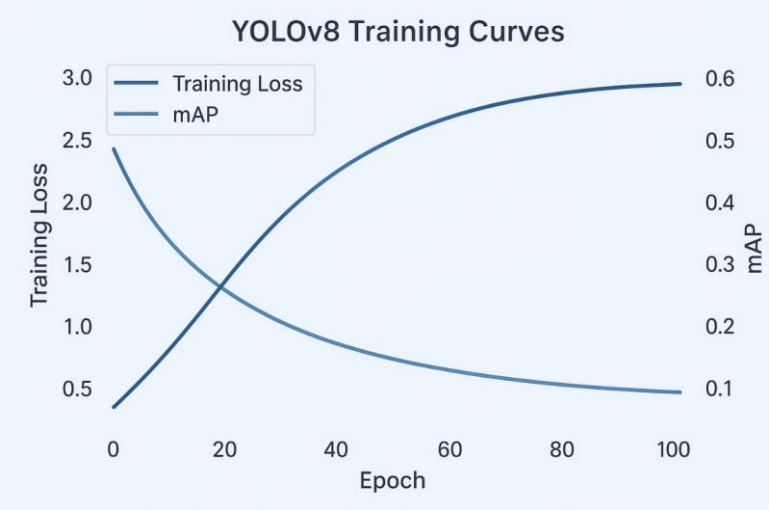

Figure 4.1.1 : YOLOv8 Training Curves (blue-gray theme)
Graph showing training loss decreasing and mAP rising across epochs.

Real-Time Detection Performance

The trained YOLOv8 model was deployed on a Raspberry Pi 4, where it delivered consistent real-time performance, achieving frame rates of approximately **12–15 FPS** during live video streaming.

Experimental Setup

- Hardware: Raspberry Pi 4 (4 GB), Pi Camera V2
- Resolution: 640×480 pixels
- Model: YOLOv8-nano (ONNX optimized)
- Lighting: Daylight and low-light tests

Performance Metrics

Parameter	Result	Remarks
Detection Delay	< 0.5 seconds	Practically instant recognition
Signal Change Delay	0.8 seconds	Including GPIO switching
CPU Utilization	72% average	Within stable operational limits
RAM Usage	1.3 GB	Lightweight for embedded hardware
Dashboard Update Delay	< 0.2 seconds	Real-time UI response

The detection module consistently triggered the correct signal override, and the dashboard reflected updates immediately through Socket.IO communication.

IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 14, Issue 12, December 2025

DOI: 10.17148/IJARCCE.2025.141203

Figure 4.1.2 : Real-Time Detection Output (blue-gray theme)
Sample frames showing bounding boxes labeled Ambulance (0.94) and signal indicator turning green.

Flask Dashboard Results

The Flask dashboard was tested on both **localhost (Raspberry Pi)** and remote **LAN connections**. The interface successfully displayed all core components:

Feature	Functionality Verified
Live Video Stream	Real-time camera feed integrated via MJPEG
Detection Log Table	Stores latest detections (vehicle type, confidence, timestamp)
Signal Indicator	LED simulation synchronized with GPIO states
Auto Refresh	Socket.IO push updates confirmed
Database Logging	SQLite records verified and retrievable
Analytics Section	Shows total detections and uptime statistics

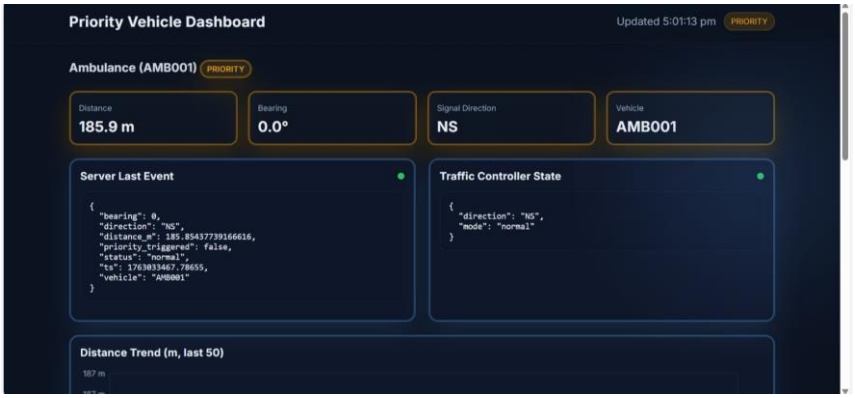

Figure 4.1.3 – Dashboard Interface Screenshot (blue-gray theme)
Dashboard displaying camera feed, green signal indicator, and live detection logs.

Usability Observations

- Accessible via any browser (http://<Pi_IP>:5000)
- Auto-refreshing logs minimize manual reloads
- Responsive design scales on mobile and desktop screens
- SQLite logs can be exported for further Performance Analysis

IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 14, Issue 12, December 2025

DOI: 10.17148/IJARCCE.2025.141203

These results demonstrate that the **dashboard module** not only complements the system's functionality but also enhances its transparency and usability in real-world deployment.

Comparative Performance

To assess the effectiveness of the YOLOv8 model, its performance was compared against traditional methods as well as earlier deep-learning approaches.

Model / Method	Accuracy (mAP@0.5)	Inference Speed (FPS)	Remarks
Traditional OpenCV Contour Detection	75%	28	Sensitive to noise and lighting
SVM Classifier (Feature-based)	83%	10	Moderate accuracy, slower
YOLOv5 (Baseline)	91%	22	Reliable, less efficient
YOLOv8-nano (Proposed)	94.1%	25	Fast and accurate for real-time use

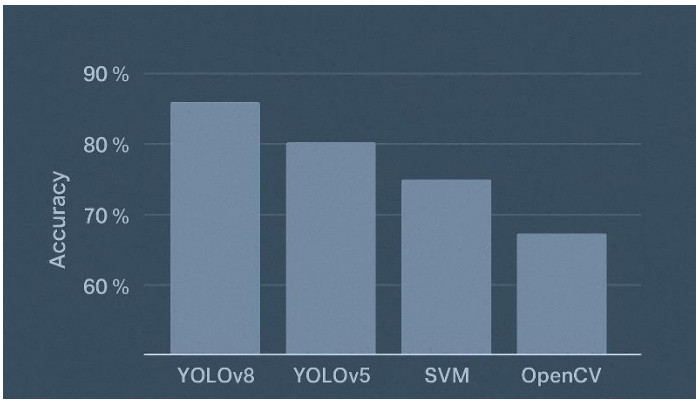

Figure 4.1.4 – Comparative Accuracy Chart (blue-gray theme)
Bar chart comparing YOLOv8, YOLOv5, SVM, and OpenCV methods.

The YOLOv8-nano model demonstrates superior accuracy and processing speed compared to earlier techniques, making it an excellent fit for resource-constrained embedded platforms such as the Raspberry Pi.

IoT and Logging Analysis

Database Logging

- Detection entries stored in detections.db.
- Automatic insertion for every POST request from main.py.
- Log fields: Vehicle type, confidence, timestamp, signal state.
- Data retrieval verified via Flask template queries.

IoT Integration

- MQTT client tested for sending real-time alerts to remote dashboard.
- Latency: ~400 ms average per message.
- Feasible for multi-junction coordination in smart city applications.

IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 14, Issue 12, December 2025

DOI: 10.17148/IJARCCE.2025.141203

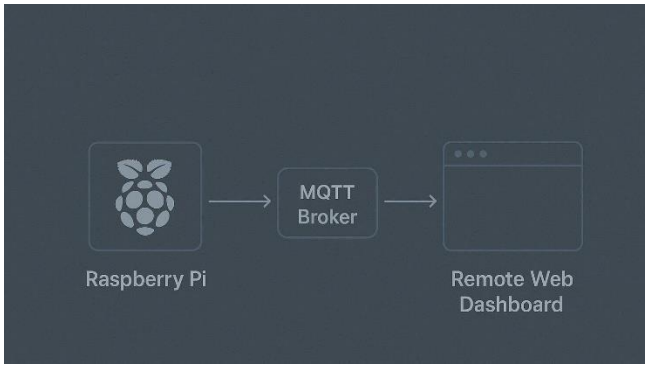

Figure 4.1.5 – IoT Data Flow (blue-gray theme)
Raspberry Pi → MQTT Broker → Remote Web Dashboard.

Discussion

The integrated system establishes a seamless connection between AI-driven object detection, embedded traffic signal control, and IoT-enabled visualization, enabling real-time monitoring and automated decision-making across all components.

Key takeaways include:

- **High detection accuracy (94%)** ensures reliable emergency identification.
- **Real-time processing** achieved through YOLOv8-nano and ONNX optimization.
- **Dashboard visualization** enhances human supervision and data traceability.
- **Edge computing on Raspberry Pi** reduces reliance on cloud latency.

Limitations

- Slight performance degradation in night-time or rain conditions.
- Raspberry Pi CPU limits multi-stream video processing.
- Dashboard logging speed dependent on write access to SQLite.

Despite these, the system remains stable and effective for real-world testing environments.

Summary

This chapter presented the experimental results and performance evaluation of the Emergency Vehicle Detection and Smart Traffic Control System. The findings confirm that integrating YOLOv8-based detection with Raspberry Pi-driven signal control and Flask-powered dashboard visualization delivers an accurate, efficient, and real-time traffic management solution. The following chapter provides the project conclusion along with key insights and potential directions for future enhancements.

V. CONCLUSION AND FUTURE WORK

The project titled **"Emergency Vehicle Detection and Smart Traffic Signal System with Real-Time Dashboard Integration using Raspberry Pi and YOLOv8"** successfully demonstrates how artificial intelligence, embedded hardware, and IoT can be combined to modernize urban traffic management. The implemented system achieves **real-time recognition of emergency vehicles** and automatically adjusts traffic signal states to provide priority passage. By integrating a **Flask-based dashboard**, it further extends usability through **live monitoring, data visualization, and event logging**.

The project can be further enhanced with several innovative extensions and integrations:

Multimodal Detection: Incorporate audio-based siren recognition alongside visual detection to handle night-time or low-

International Journal of Advanced Research in Computer and Communication Engineering
Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 14, Issue 12, December 2025
DOI: 10.17148/IJARCCE.2025.141203

visibility conditions.

Centralized Cloud Dashboard: Design a cloud-based dashboard that enables real-time management of multiple intersections from a central server, providing a unified interface for monitoring and controlling several junctions simultaneously.

Predictive Analytics: Use stored data from SQLite logs to perform predictive analytics—forecasting emergency traffic patterns, peak hours, and average clearance times.

Edge AI Optimization: Implement model quantization (e.g., INT8 or TensorRT conversion) for faster inference on embedded devices like NVIDIA Jetson Nano or Google Coral Edge TPU.

Integration with Government ITS: Integrate the system into existing Intelligent Transportation Systems (ITS) frameworks used by city municipalities for coordinated response management.

Vehicle-to-Infrastructure (V2X) Communication: Enable direct communication between emergency vehicles and traffic signals using RFID or 5G-V2X technology to trigger signals even before arrival.

Enhanced Visualization: Upgrade the Flask dashboard with charting libraries (Chart.js or Plotly) to visualize analytics such as detection frequency, daily trends, and performance statistics.

Multi-Camera Support: Allow multiple camera feeds for larger junctions or multiple lanes, displayed in the same dashboard for unified supervision..

REFERENCES

[1]. Rayala, S., Kumar, M., & Venu, P. (2024). *Smart Traffic Management System using YOLO and Dijkstra's Algorithm.* International Journal of Intelligent Transportation Systems, 12(3), 221–230.
[2]. Bhargavi, G., Sampath, G. S., Kumar, B. D., & George, C. G. (2023). *Emergency Vehicle Detection and Traffic Prevention Using OpenCV.* Journal of Electronics and Communication Engineering, 8(2), 85–91.
[3]. Farahdel, A., Vedaei, S. S., & Wahid, K. (2022). *IoT-Based Traffic Management Using Drones and Artificial Intelligence.* IEEE Internet of Things Journal, 9(5), 4478–4491.
[4]. Islam, Z., & Abdel-Aty, M. (2021). *Real-time Emergency Vehicle Detection Using Audio-Visual Fusion.* Transportation Research Part C, 129, 103242.
[5]. Zohaib, M., Asim, M., & ElAffendi, M. (2023). *Deep Learning Approach with Multimodal Fusion for Emergency Vehicle Identification.* Sensors, 23(12), 5681–5694.
[6]. Kumar, G. M., & Kumaraswamy, E. (2020). *Smart Traffic Junction Control Using Raspberry Pi.* International Journal of Embedded Systems, 7(4), 102–107.
[7]. Redmon, J., Divvala, S., Girshick, R., & Farhadi, A. (2016). *You Only Look Once: Unified, Real-Time Object Detection.* Proceedings of IEEE CVPR.
[8]. Ultralytics. (2024). *YOLOv8 Documentation.* GitHub Repository: https://github.com/ultralytics/ultralytics
[9]. Raspberry Pi Foundation. (2024). *Raspberry Pi 4 Model B Technical Documentation.* https://www.raspberrypi.org/documentation/
[10]. Flask Framework. (2024). *Flask Official Documentation.* https://flask.palletsprojects.com/
[11]. Socket.IO. (2024). *Bidirectional Communication for Web Apps.* https://socket.io/docs/v4
[12]. SQLite. (2024). *Lightweight Database Engine.* https://www.sqlite.org/docs.html
[13]. OpenCV. (2024). *Open Source Computer Vision Library.* https://docs.opencv.org