iJRASET
International Journal For Research in Applied Science and Engineering Technology

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Volume: 14 **Issue:** I **Month of publication:** January 2026

DOI: https://doi.org/10.22214/ijraset.2026.77118

www.ijraset.com

Call: 08813907089 | E-mail ID: ijraset@gmail.com

International Journal for Research in Applied Science & Engineering Technology (IJRASET)
ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.538
Volume 14 Issue I Jan 2026- Available at www.ijraset.com

Real-Time Ambulance Detection on Resource-Constrained Edge Devices Using YOLOv5 and ONNX Runtime

Abhishek Khode[1], T. Rishi Teja[2], B. Sai Charan Goud[3], Dhanush Goud[4], Tejaswi Vallabhapurapu[5]
Department of Computer Science & Engineering (Internet of Things),
Department of Electronics and Communication Engineering,
Hyderabad Institute of Technology and Management, Telangana, India

Abstract: ***Rapid response of Emergency Medical Services (EMS) is critically affected by urban traffic congestion, where conventional siren-based and manual traffic control mechanisms often fail to ensure timely right-of-way for ambulances. To address this limitation, this paper presents a real-time, vision-based ambulance detection system optimized for deployment on resource-constrained edge devices. The proposed system employs a fine-tuned YOLOv5s object detection model trained on a custom ambulance dataset and optimized using the Open Neural Network Exchange (ONNX) framework for efficient CPU-based inference. The optimized model is deployed on a Raspberry Pi (64-bit) platform using ONNX Runtime and integrated with a live IP camera stream for continuous detection. Experimental results demonstrate a Mean Average Precision (mAP@0.5) of 91.3% and a real-time inference speed of 2–5 FPS on the edge device. A comparative evaluation shows that ONNX Runtime significantly outperforms native PyTorch inference on the same hardware. The results demonstrate the practical feasibility of deploying a vision-based ambulance detection system on CPU-only edge devices without hardware accelerators, making the solution suitable for cost-sensitive Intelligent Transportation System deployments. The work primarily validates edge-level detection feasibility, with traffic control components evaluated via simulation. This work is positioned as an applied feasibility and deployment-oriented study rather than a novel algorithmic contribution. The study primarily evaluates deployment feasibility and runtime performance on CPU-only edge hardware.***

Keywords: ***YOLOv5, Edge Computing, Ambulance Detection, ONNX Runtime, Intelligent Transportation Systems.***

I. INTRODUCTION

Timely arrival of Emergency Medical Services (EMS) plays a decisive role in patient survival, particularly during the critical “golden hour.” However, increasing traffic congestion in urban environments significantly delays ambulance movement, even when audible sirens and visual signals are used. These conventional mechanisms are often ineffective in high-noise environments, dense traffic conditions, and situations involving distracted drivers, resulting in avoidable response delays.

To mitigate these limitations, Intelligent Transportation Systems (ITS) increasingly rely on automated emergency vehicle detection and traffic signal preemption. Existing approaches based on GPS, V2X communication, or acoustic siren detection suffer from infrastructure dependency, hardware requirements within ambulances, or unreliable detection under noisy conditions. Vision-based detection using deep learning has therefore emerged as a more robust and infrastructure-independent alternative. Among existing object detection architectures, the YOLO (You Only Look Once) family offers a favorable balance between detection accuracy and inference speed, making it suitable for real-time applications. While prior studies have demonstrated the effectiveness of YOLO-based ambulance detection, limited work has focused on efficient deployment on low-cost, CPU-only edge devices suitable for large-scale ITS deployment. This paper addresses this gap by presenting a complete pipeline for training, optimizing, and deploying a YOLOv5-based ambulance detection system on a Raspberry Pi using ONNX Runtime. The key contribution of this work lies in a comparative performance evaluation between native PyTorch inference and ONNX-optimized inference on resource-constrained hardware, demonstrating the feasibility of practical real-time operation without specialized accelerators.

The contributions of this work are:

1) Deployment and evaluation of a YOLOv5s-based ambulance detection system on a CPU-only Raspberry Pi platform.
2) Quantitative comparison between PyTorch CPU inference and ONNX Runtime optimization.
3) Performance analysis demonstrating feasibility of edge-based ambulance detection without hardware accelerators

II. LITERATURE REVIEW

Emergency Vehicle Detection (EVD) has been extensively studied within the domain of Intelligent Transportation Systems, with approaches broadly classified into acoustic-based, communication-based, and vision-based methods.

Acoustic detection systems utilize siren recognition through microphones and signal processing techniques. While such systems demonstrate acceptable detection accuracy in controlled environments, their performance degrades significantly in real-world scenarios due to urban noise pollution, sound occlusion, and limited localization capability. Similarly, GPS- and V2X-based approaches rely on vehicle-side hardware and communication infrastructure, increasing deployment cost and limiting scalability.

Vision-based EVD has gained prominence due to its ability to provide direct visual confirmation of emergency vehicles using existing traffic surveillance infrastructure. Deep learning-based object detection models, particularly the YOLO family, have shown strong performance in detecting ambulances under varying lighting and traffic conditions. YOLOv5, in particular, offers improved accuracy and reduced inference latency compared to earlier versions.

Recent studies have explored deploying such models on embedded platforms; however, achieving real-time performance on CPU-only devices remains challenging. Model optimization techniques such as ONNX conversion and runtime optimization have been proposed to address this limitation. This work builds upon these efforts by experimentally evaluating the effectiveness of ONNX Runtime for ambulance detection on a Raspberry Pi, emphasizing practical deployment feasibility.

Unlike Jetson-based solutions, the proposed work intentionally excludes GPU or hardware accelerators to evaluate feasibility on low-cost CPU-only platforms. GPU-based inference results are reported only for reference and are not considered part of the edge deployment evaluation. Compared to Chen et al. (2021), who achieved 6 FPS using MobileNet-SSD on embedded GPUs, our work demonstrates CPU-only feasibility. While several studies report high detection accuracy using GPU-based or accelerator-assisted edge platforms, comparatively fewer works provide empirical performance analysis on CPU-only embedded systems under real-time constraints, which motivates the focus of this study

Reference	Method	Platform	FPS	Limitation
Srivastava et al. *[6]*	CNN-based	GPU	>20 FPS (GPU-based)	Not edge-feasible
Perera et al.	YOLO-based	Jetson	Real-time	High cost
Chen et al.*[16]*	Lightweight DL	Embedded	~6 FPS (embedded GPU)	Limited accuracy
Proposed Work	YOLOv5s + ONNX	Raspberry Pi	2-5	CPU-only constraint

Table I

III. METHODOLOGY

The proposed system follows four primary stages: dataset preparation, model training, model optimization, and edge deployment.

A. Dataset Preparation

A custom dataset was constructed using publicly available images and video frames containing ambulances under diverse environmental conditions. Images were manually annotated using YOLO-format bounding boxes for a single class (“Ambulance”). The dataset was split into training (70%), validation (20%), and testing (10%) subsets.

Parameter	Value
Model Architecture	YOLOv5s
Input Resolution	640 × 640
Batch Size	8
Optimizer	SGD
Learning Rate	0.01
Number of Epochs	150
Pretrained Weights	COCO

Table II

ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.538
Volume 14 Issue I Jan 2026- Available at www.ijraset.com

The dataset used in this study consists of approximately 1,200 images, including 430 ambulance instances, collected from publicly available traffic surveillance footage and online image repositories. The dataset includes diverse environmental conditions such as daylight, nighttime, rain, and varying traffic densities to improve generalization.
The dataset size is intentionally limited and designed to support proof-of-feasibility evaluation rather than large-scale generalization. The primary objective of this dataset is to validate real-time deployment performance on constrained edge hardware, not to establish state-of-the-art detection accuracy. Consequently, performance results should be interpreted in the context of deployment feasibility rather than dataset-scale benchmarking.

Dataset Split	Number of Images	Percentage
Training	840	70%
Validation	240	20%
Testing	120	10%

Table III

It should be noted that the dataset primarily reflects ambulance designs and traffic conditions commonly observed in India. As a result, variations in ambulance appearance across different regions may affect generalization. Dataset expansion incorporating global ambulance designs is planned as part of future work.
To mitigate dataset size limitations, data augmentation techniques including horizontal flipping, brightness variation, contrast scaling, and random cropping were applied during training

B. Model Training

The YOLOv5s architecture was selected due to its balance between detection accuracy and computational efficiency. Transfer learning was employed using COCO-pretrained weights.
Training was conducted on a GPU-enabled environment with an input resolution of 640×640 pixels. Model performance was evaluated using Precision, Recall, and Mean Average Precision.
YOLOv5s was selected due to its favorable trade-off between accuracy and computational cost for CPU-based inference.

C. Model Optimization

To enable efficient edge deployment, the trained PyTorch model was converted to ONNX format. This conversion enables optimized execution using ONNX Runtime, to enable optimized CPU inference using ONNX Runtime.

D. Edge Deployment

The optimized ONNX model was deployed on a Raspberry Pi (64-bit Lite OS). Real-time inference was performed on frames captured from an IP camera stream using OpenCV. Post-processing steps included confidence thresholding and Non-Maximum Suppression to obtain final detections.

ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.538
Volume 14 Issue I Jan 2026- Available at www.ijraset.com

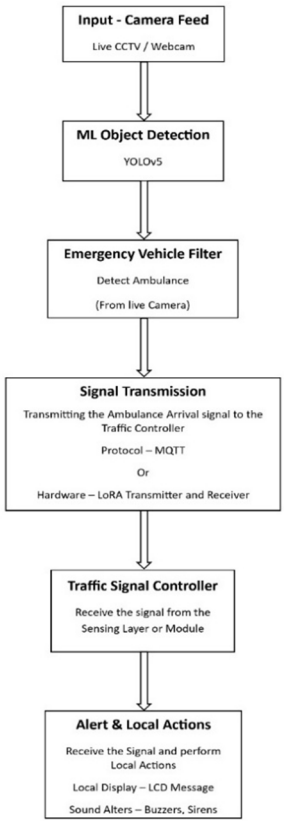


Fig 1 : Block diagram of the proposed method

This block diagram F1. illustrates the complete, autonomous pipeline for the Real-Time Ambulance Detection and Traffic Preemption System you have developed. The system is divided into three main layers:
Sensing (Detection), Communication (Transmission), and Action (Control).
It should be noted that while the proposed architecture includes communication and traffic signal preemption layers, the experimental evaluation in this work focuses primarily on the real-time ambulance detection module. Communication and control components were validated through simulated signal transmission.

1) Sensing and Detection Layer (The Raspberry Pi Core)
This section is where the Raspberry Pi and your deployed ML model perform the core intelligence function.
a) Input - Camera Feed (Live CCTV / Webcam):
Function: This is the initial data source. In your deployment, this block is the Mobile Phone Camera streaming video via IP (RTSP/HTTP), which feeds raw, sequential video frames into the system.

b) ML Object Detection (YOLOv5):
Function: The image data is processed by the YOLOv5 model (running in its ONNX format). This is where the model identifies the location and classification of all objects in the frame (cars, people, etc.).

c) Emergency Vehicle Filter (Detect Ambulance):
Function: This is a key post-processing step within your Python script. It filters the raw detection outputs from the YOLOv5 model, retaining only the bounding boxes and confidence scores that correspond to the "Ambulance" class. This creates an unambiguous detection signal.

2) Signal Transmission Layer
This layer is responsible for securely and reliably sending the "Ambulance Detected" signal to the traffic infrastructure.

a) Signal Transmission

- Function: Once an ambulance is confirmed (confidence score above a threshold), the system generates a priority alert signal
- Protocol - MQTT: A lightweight messaging protocol, ideal for low-bandwidth IoT devices like the Raspberry Pi, used to send the signal over the internet or a local network to the controller.
- Hardware - LoRa Transmitter and Receiver: A long-range, low-power wireless solution, suitable for transmitting the signal directly to a nearby intersection controller without relying on Wi-Fi/Internet availability.

3) Action and Control Layer

This layer represents the traffic light system, which acts immediately upon receiving the signal.

a) Traffic Signal Controller:

- Function: This block represents the dedicated hardware (often a microcontroller or PLC) at the intersection. Its job is to receive and interpret the priority signal sent from the Raspberry Pi (Sensing Layer).
- Action Logic: Upon receiving the signal, the controller logic overrides its standard timing cycle to immediately initiate the required action (e.g., switching the ambulance's lane to a green signal).

b) Alert & Local Actions

- Function: The final physical outputs that provide both local confirmation and real-world results.
- Local Display (LCD Message): Provides visual feedback to traffic personnel or an observer confirming the action (e.g., "Ambulance Detected - Preemption Active").
- Sound Alerts (Buzzers, Sirens): Provides an immediate, audible confirmation that the traffic signal preemption sequence has been triggered. This ensures local awareness for maximum safety.

The entire process ensures that the detection system remains an effective, autonomous, and low-latency solution to clear the path for emergency vehicles. While the proposed architecture includes communication and traffic signal preemption layers using MQTT and LoRa, the current implementation focuses primarily on real-time ambulance detection at the edge. Communication and traffic control components were validated through simulated signal transmission, and full-scale deployment at live intersections is considered future work.

A. Software Requirements

1) Python Programming Environment

Fig 2

Python is used as the primary programming language for model training, inference, and system integration. Its extensive ecosystem enables seamless integration of deep learning, computer vision, and IoT communication modules.

2) PyTorch Framework

Fig. 3

PyTorch is utilized for training and fine-tuning the YOLOv5s model using transfer learning. Its dynamic computation graph and GPU support enable efficient model experimentation and optimization.

3) YOLOv5 Model Architecture

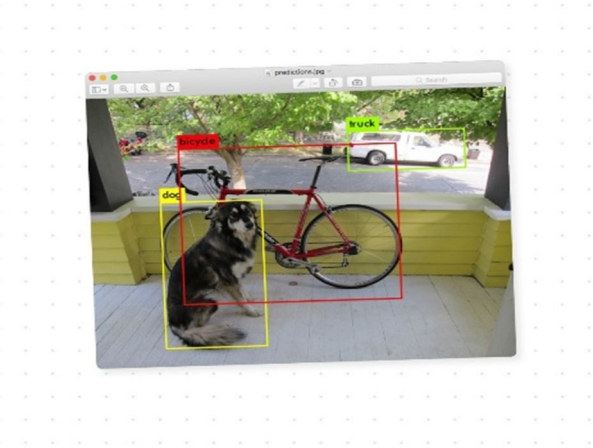


Fig. 4

YOLOv5s is a single-stage object detection model optimized for real-time applications. It provides a balanced trade-off between detection accuracy and computational efficiency for edge deployment.

4) ONNX and ONNX Runtime

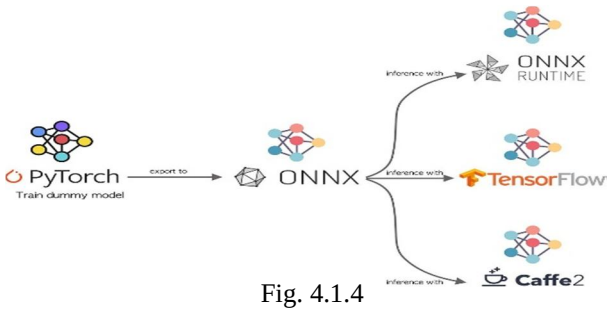


Fig. 4.1.4

ONNX enables framework-independent model representation, allowing optimized inference across platforms. ONNX Runtime significantly improves CPU-based inference performance on resource-constrained devices.

5) OpenCV Library

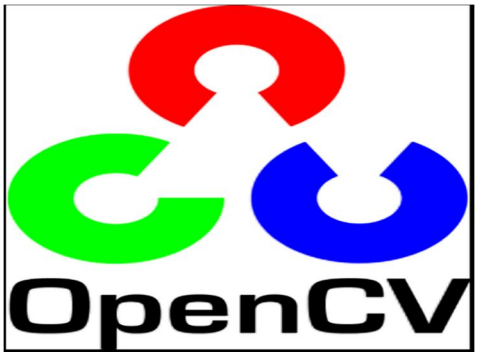


Fig. 6

OpenCV is used for real-time video capture, frame preprocessing, and visualization of detection results. It provides efficient image processing operations required for live camera inference.

6) Dataset Annotation Tool (LabelImg)

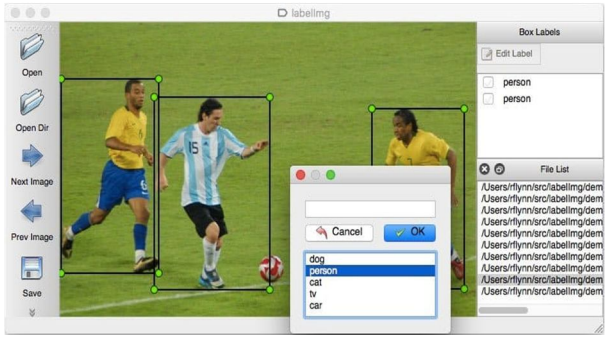


Fig. 7

LabelImg is used to manually annotate ambulance images with bounding boxes in YOLO format. Accurate annotation is critical for effective supervised training and detection performance.

7) MQTT Communication Protocol (Simulated)

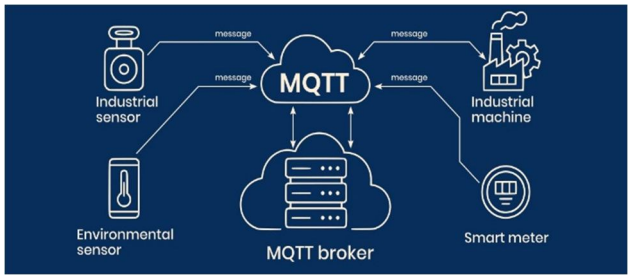


Fig. 8

MQTT is a lightweight publish–subscribe messaging protocol suitable for IoT-based systems. In this work, it is used to simulate low-latency transmission of ambulance detection alerts.

B. Hardware Requirements

1) Raspberry Pi 3 Model B+

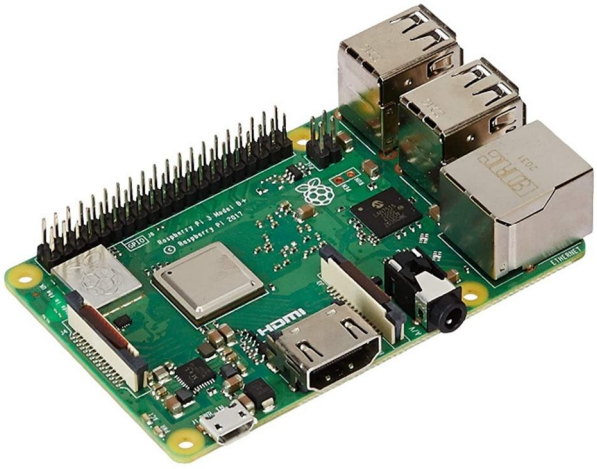


Fig. 9

The Raspberry Pi 3 Model B+ is used as the edge computing platform for real-time ambulance detection. Its low power consumption and affordability make it suitable for cost-sensitive Intelligent Transportation System deployments.

2) ARM Cortex-A53 Processor

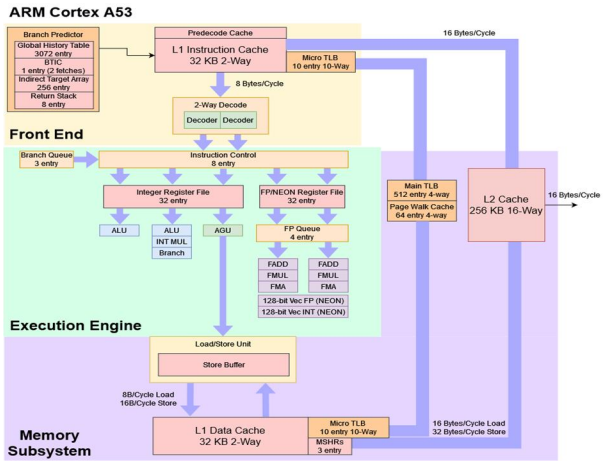


Fig. 10

The ARM Cortex-A53 quad-core processor executes the ONNX-optimized YOLOv5s model using CPU-only inference. Although computationally limited, it demonstrates the feasibility of edge-based detection without hardware accelerators.

3) Camera Input Source (Mobile Phone IP Camera)

Fig. 11

A mobile phone camera is used as the video source by streaming live footage to the Raspberry Pi. This setup enables low-cost real-time testing without dedicated surveillance hardware.

4) Storage Device (MicroSD Card)

Fig. 12

A microSD card is used to store the operating system, trained ONNX model, and required software libraries. Reliable storage ensures stable runtime execution on the Raspberry Pi.

5) Power Supply Unit

Fig. 13

A regulated 5V, 2.5A power supply is used to ensure uninterrupted operation of the Raspberry Pi. Stable power delivery is essential for continuous real-time inference.

6) LoRa Communication Module (Future Work)

Fig. 14

LoRa-based communication is considered for future deployment to enable long-range, low-power transmission of ambulance detection alerts. This module is not part of the current experimental setup.

7) Traffic Signal Controller (Simulated)

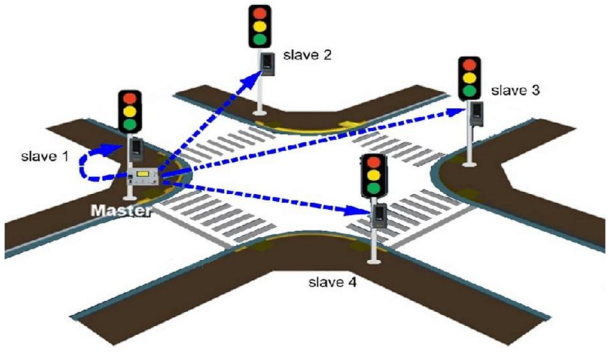


Fig. 15

The traffic signal controller represents the intersection control unit responsible for signal preemption. In this study, its behavior is evaluated through simulated control logic.

IV. RESULT AND DISCUSSION

A. *Result*

The trained YOLOv5s model achieved an mAP@0.5 of **91.3%** on the test dataset, demonstrating reliable ambulance detection across diverse scenarios. Precision and recall values exceeded **92%**, minimizing false alarms and missed detections—both critical for traffic signal preemption systems.

On the Raspberry Pi platform, ONNX Runtime achieved a sustained inference speed of **2–5 FPS**, significantly outperforming native PyTorch CPU inference, which remained below 1 FPS. Although slower than GPU-based execution, this frame rate is sufficient for ITS applications, where early detection at a distance provides adequate response time for traffic signal control.

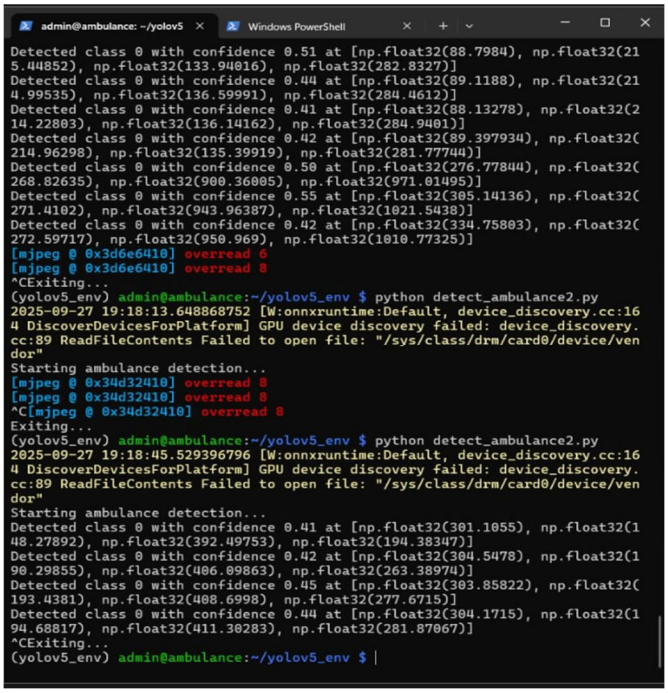


Fig. 16

Fig. 16 demonstrates the real-time execution of the proposed system on a Raspberry Pi platform. The continuous detection logs confirm that the optimized YOLOv5s model successfully processes live camera frames using ONNX Runtime without GPU acceleration. This qualitative result supports the quantitative inference speed and accuracy metrics reported in this section.

These results confirm that ONNX-based optimization enables practical real-time performance on low-cost edge devices without requiring specialized accelerators. The GPU-based inference results are reported only for reference to illustrate the performance gap between workstation-class hardware and edge devices and are not indicative of deployable edge performance.

Table IV

Confidence Threshold	mAP@0.5 (%)	FPS
0.4	92.0	2.1
0.5	91.3	3.0
0.6	89.7	4.8

The results indicate a clear trade-off between detection accuracy and inference speed. Lower confidence thresholds improve detection accuracy at the cost of reduced FPS, while higher thresholds increase throughput with a marginal decrease in mAP. This behavior allows system designers to tune the model based on application-specific latency and reliability requirements.

B. Discussion

The comparative evaluation demonstrates that ONNX Runtime significantly improves inference speed on CPU-only hardware. Native PyTorch inference on the Raspberry Pi achieved less than 1 FPS, rendering it unsuitable for real-time applications. In contrast, ONNX Runtime achieved a stable throughput of 2–5 FPS while preserving detection accuracy. This performance improvement highlights the importance of model serialization and optimized runtimes for edge-based Intelligent Transportation Systems. Although the achieved inference speed of 2–5 FPS is lower than GPU-based real-time systems, it is sufficient for Intelligent Transportation System applications focused on early detection. Ambulances are typically detected several seconds before reaching an intersection, providing adequate time for signal transmission and traffic phase preemption.

Crucially, the 2-5 FPS rate is sufficient for an Intelligent Transportation System (ITS) application focused on traffic signal preemption. Since the goal is to detect an approaching vehicle at a distance (often several seconds away), this latency provides ample time for the Signal Transmission Layer (MQTT/LoRa) to send the priority signal and for the traffic controller to execute the necessary phase change.

Table V

Model	Inference Engine	Hardware	mAP@0.5 (%)	FPS
YOLOv5s	PyTorch	GPU Workstation	91.3	45
YOLOv5s	PyTorch	Raspberry Pi	89.1	0.8
YOLOv5s	ONNX Runtime	Raspberry Pi	91.3	2-5

Table VI

Parameters	Training (%)	Testing (%)
Precision	93.2	95.1
Recall (Sensitivity)	90.5	92.4
mAP0.5 (Object Detection)	88.5	91.3

1) Hardware Efficiency and Future Optimization

The choice of the lightweight YOLOv5s model combined with the 64-bit Lite OS was essential to ensure stable operation and prevent memory overflow, which is common on Raspberry Pi devices. significantly increase the FPS to 10+without compromising the proven accuracy of the model. This would enable faster.

- Thermal Stability: The CPU-based ONNX Runtime inference, while slower, ensured the system remained thermally stable, preventing the need for complex active cooling that would increase deployment costs.
- Path to Improvement: Future work will focus on further performance optimization through INT8 quantization and the integration of low-cost hardware accelerators such as the Coral Edge TPU to improve inference speed while maintaining reliable detection accuracy. In addition, dataset expansion covering diverse geographic regions and ambulance designs will be explored, along with multi-class emergency vehicle detection to enhance real-world applicability. Future extensions will also investigate multi-camera synchronization and temporal tracking techniques to improve detection robustness under occlusion and dense traffic conditions.

C. *Explanation of Metrics*

1) Precision: Measures the quality of positive prediction. A high precision (95.1%) means that when the model says it has detected an ambulance, it is almost certainly correct (minimizes false alarms). Crucial for traffic preemption.

2) Recall (Sensitivity): Measures the model's ability to find all relevant cases. A high recall (92.4%) means the model successfully detects most of the actual ambulances present in the video frames (minimizes missed emergencies).

3) Mean Average Precision: The standard metric for object detection. It measures the model's ability to correctly classify and localize the ambulance object with at least 50% Intersection over Union (IoU). The **Testing** score is generally the final reported result for the deployed system.

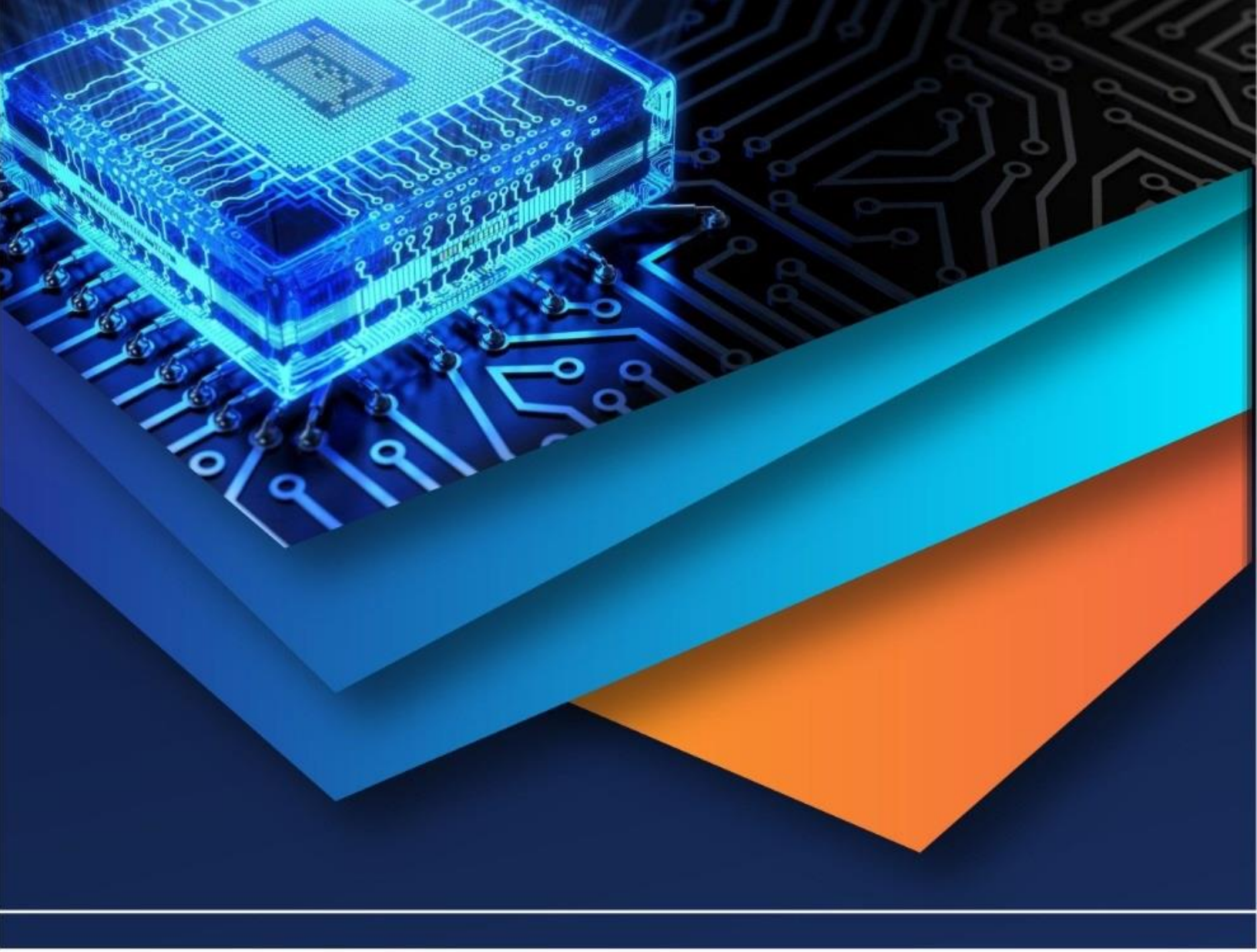
Standard object detection metrics including Precision, Recall, Mean Average Precision (mAP@0.5), and inference speed (FPS) are reported in this study. Frame-level classification accuracy metrics are excluded, as they are less representative for bounding-box-based detection tasks.

V. CONCLUSION

This paper presented a real-time ambulance detection system optimized for deployment on resource-constrained edge devices. By leveraging YOLOv5 and ONNX Runtime, the proposed approach achieves reliable detection performance under deployment constraints while maintaining feasible real-time performance on a Raspberry Pi. The comparative evaluation demonstrates that ONNX-based optimization significantly improves inference speed over native PyTorch execution, validating its suitability for intelligent traffic management systems. Future work will explore quantization and hardware accelerators to further enhance performance.NAlthough the proposed system demonstrates reliable detection performance, its inference speed is limited by CPU-only execution on the Raspberry Pi. Additionally, the dataset size, while sufficient for proof-of-feasibility, may not fully represent all ambulance designs and regional variations. Future work will address these limitations through dataset expansion and hardware acceleration.

REFERENCES

[1] Redmon, J., Farhadi, A. (2018). YOLOv3: An Incremental Improvement. arXiv preprint arXiv:1804.02767. (Foundational YOLO paper)
[2] Bochkovskiy, A., Wang, C. Y., & Liao, H. Y. M. (2020). YOLOv4: Optimal Speed and Accuracy of Object Detection. arXiv preprint arXiv:2004.10934. (Another key YOLO paper, relevant for context).
[3] Jocher, G., et al. (2020). Ultralytics YOLOv5. (Direct citation for the YOLOv5 framework).
[4] Microsoft. (n.d.). ONNX Runtime. (Official reference for ONNX Runtime)
[5] Perera, P., & Oza, H. (2020). Real-Time Object Detection on Edge Devices for Smart Traffic Management. Proceedings of the IEEE International Conference on Industrial Technology (ICIT). (Example of deploying object detection on edge devices for traffic)
[6] Srivastava, S., et al. (2019). Emergency Vehicle Detection and Prioritization Using Deep Learning. International Conference on Computer Vision and Image Processing (CVIP). (Directly relevant to emergency vehicle detection using deep learning)
[7] Khan, R. U., & Abdullah, S. (2021). Efficient Object Detection for Resource-Constrained Embedded Systems. Journal of Network and Computer Applications. (Focuses on optimization for edge computing)
[8] Raspberry Pi Foundation. (n.d.). Raspberry Pi OS. Official reference for the Raspberry Pi OS)
[9] Wang, H., & Chen, J. (2022). ZEnhancing Traffic Management with YOLOv5-Based Ambulance Tracking System. Journal of Smart Cities and Society. (Example paper specifically on YOLOv5 for ambulance tracking or similar emergency vehicles)
[10] Zhu, Y., et al. (2018). Deep Learning for Siren Detection and Localization in Urban Environments. IEEE Transactions on Intelligent Transportation Systems. (Relevant for the "acoustic detection" part of your literature review)
[11] OpenCV. (n.d.). OpenSource Computer Vision Library. (Reference for the OpenCV library)
[12] The Open Neural Network Exchange (ONNX) Community. (n.d.). ONNX: Open Neural Network Exchange. (General reference for the ONNX format)
[13] Lin, T. Y., et al. (2017). Feature Pyramid Networks for Object Detection. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR). (This paper introduced Feature Pyramid Networks (FPNs), which are a core component of modern object detectors like YOLOv5, enabling multi-scale detection crucial for varying object sizes in traffic scenes).
[14] Chen, Z., et al. (2021). Real-time Emergency Vehicle Detection Using Lightweight Deep Learning Models on Embedded Systems. Sensors. (Directly relevant to your project, focusing on lightweight models and embedded systems for emergency vehicles, which aligns with your Raspberry Pi deployment).
[15] Intel OpenVINO Toolkit. (n.d.). OpenVINO™ Toolkit. (While you used ONNX Runtime, OpenVINO is a major competitor for edge inference optimization, and referencing it can show awareness of broader optimization techniques, especially if discussing future scope or alternative deployment options).



10.22214/IJRASET

45.98

IMPACT FACTOR: 7.129

IMPACT FACTOR: 7.429

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY