Real-Time Object Detection with Quantized YOLOv11 and YOLOv8 on Raspberry Pi 5 for Low-Speed ADAS

Furkan Şimşek[1]

[1]Independent Researcher, Trabzon, Türkiye.

Contributing authors: me@furkan-simsek.tr;

his study investigates the object detection performance differences between YOLOv11 and YOLOv8 architectures for Advanced Driver Assistance Systems (ADAS) under CPU-only constraints. Speed and accuracy evaluations were conducted on a Raspberry Pi 5 (8 GB), with the CPU governor configured in performance mode to ensure stable benchmarking conditions. Since hardware accelerators such as GPUs or NPUs are not universally available in all vehicle platforms, this work focuses exclusively on CPU-based inference, which remains critical for practical ADAS deployment. Four models were evaluated: YOLOv11n, YOLOv11s, YOLOv8n, and YOLOv8s. Additionally, model optimization was performed using ONNX with INT8 post-training quantization to improve inference efficiency. Experimental results on the KITTI dataset demonstrate that the quantized YOLOv11n model achieved approximately 13 FPS with an average latency of 76.78 ms, whereas the YOLOv11s model achieved around 7 FPS with a latency of 152.20 ms. This corresponds to an approximately 46% latency reduction for the nano model compared to the small variant. The findings indicate that nano-scale YOLO models provide a more favorable speed–accuracy trade-off for real-time, CPU-based ADAS applications on low-power embedded platforms.

1 Introduction

Advanced Driver Assistance Systems (ADAS) are pivotal in reducing driver error and enhancing traffic safety using technologies like LiDAR, radar, and computer vision. There are six levels of ADAS. Deep learning serves as the core decision-making mechanism within these systems. However, deploying state-of-the-art models on edge devices presents a significant challenge due to computational constraints. There are levels of ADAS, which are listed on Table 1 [2]. This study reviews Levels 0, 1 and 2.

Table 1: Levels of ADAS

Automation Level	Description	Capabilities
Level 0	No Driving Automation	Driver performs all driving tasks
Level 1	Driver Assistance	Automated system supports driver
Level 2	Partial Driving Automation	Driver must stay alert
Level 3	Conditional Driving Assistance	Detects environment
Level 4	High Driving Automation	Self-Driving in certain areas
Level 5	Full Driving Automation	No human attention needed

This study utilizes a fine-tuning methodology to adapt pre-trained models to traffic-specific scenarios using KITTI and BDD100K datasets. Furthermore, to address the hardware limitations of the Raspberry Pi 5, This study employs Open Neural Network Exchange (ONNX) conversion and INT8 quantization. These techniques are known to enhance cross-platform compatibility and reduce memory bandwidth pressure on CPU-based systems. Accuracy is not treated as a primary optimization objective, as the study focuses on latency-critical CPU-only deployment scenarios.

2 Related Work

The evolution of ADAS architectures has increasingly focused on optimizing deep learning models for real-time performance on devices lacking dedicated accelerators (GPUs/NPUs).

Ji et al. applied architectural optimizations such as batch normalization and residual connections to improve efficiency. Similarly, Kwasniewska et al. highlighted that combining lightweight architectures (e.g., MobileNet) with reduced numerical precision (quantization) significantly lowers memory usage and computational load [6].

In a comparative study, Kristiani et al. (2020) evaluated Inception V3 and MobileNet on Raspberry Pi 4 and Intel Neural Compute Stick 2. They reported that MobileNet achieved 24 FPS with 71.29% accuracy, outperforming Inception V3 [7].

Building on this hardware progression, recent benchmarks on the Raspberry Pi 5 demonstrate that the YOLOv8n (Nano) model, when optimized with NCNN or TFLite (INT8 quantization), can achieve inference speeds of approximately 30–40 FPS at 320×320 resolution, effectively enabling real-time detection on CPU-only edge configurations [5].

This study extends such benchmarks to the newer Raspberry Pi 5, offering a direct quantitative comparison between YOLOv8 and YOLOv11 architectures under quantized conditions [13].

3 Methodology

3.1 Model and Architecture Selection

In this work, we evaluated two variants of the YOLOv11 architecture (Nano and Small) and YOLOv8 architecture. The YOLO family was selected for its single-stage design, which offers a favorable trade-off between inference speed and accuracy. YOLOv8 was selected for its performance on the related works. It is usually used for Edge-Devices based AI/ML applications. This is because this study also does benchmark for YOLOv8 and YOLOv11 architectures.

3.2 Datasets and Preprocessing

To ensure a balanced evaluation within reasonable training times, utilized reduced versions of the KITTI and BDD100K datasets [14]. The KITTI dataset contains approximately 7,000 images for training and validation, while the BDD100K dataset contains approximately 10,000 images for training, validation, and testing. All images were resized to 640×640 pixels, and preprocessing followed the standard YOLO and COCO pipeline, including mosaic augmentation. Due to the computational constraints of the local training hardware (RTX 3050 Laptop GPU), stratified sampling to preserve diversity of the BDD100K dataset. This approach allowed for feasible training durations while aiming to preserve the statistical diversity of the original data.

3.3 Training Configuration

Training was conducted on an NVIDIA RTX 3050 Laptop GPU. The hyperparameters were configured as follows:

- **Input Size:** 640×640 (KITTI and BDD100K)
- **Epochs:** 50 (KITTI), 100 (BDD100K)
- **Batch Size:** 16
- **Optimizer:** AdamW

3.4 Optimization Strategy

To enable efficient execution on Raspberry Pi 5, a two-stage optimization pipeline was applied [4]:

1. **ONNX Conversion:** PyTorch models were converted to ONNX format (Opset 12) to ensure compatibility with CPU-based runtime engines.
2. **INT8 Quantization:** Using the Quantize-DeQuantize (QDQ) format, FP32 weights were quantized to INT8 with 30% pruning applied post-training. Calibration was performed using 300 random images from the validation set to minimize quantization error [8].

3.5 Experimental Setup

All inference experiments were conducted using ONNX Runtime with a fixed number of CPU threads set to the number of available Cortex-A76 cores. Default ONNX

Runtime optimization flags were enabled, and dynamic frequency scaling was disabled by enforcing the Linux CPU governor in performance mode. The test set consisted of three independent videos (1280 × 720 resolution, 10 FPS) recorded on the Trabzon Coastal Road. Each test was repeated 5 times, with the first 20 frames discarded as warm-up iterations. However, the accuracy tests (mAP50 and mAP50-95) are conducted on their own validation datasets.

3.6 Test Metrics

To comprehensively evaluate the suitability of YOLO-based object detection models for real-time ADAS deployment on resource-constrained hardware, multiple performance metrics were considered. In addition to conventional throughput measurements, latency-oriented metrics were emphasized, as real-time driving assistance systems are highly sensitive to worst-case delays rather than average performance alone.

End-to-end (E2E) latency was adopted as the primary system-level metric, as it captures the total processing delay from input frame acquisition to final output generation, including preprocessing, neural network inference, and post-processing stages. This metric directly reflects the responsiveness perceived by an ADAS pipeline in practical driving scenarios. In contrast, inference time isolates the pure forward-pass execution cost of the neural network, enabling a more granular comparison of model architectures independent of I/O and post-processing overheads.

Throughput was quantified using Frames Per Second (FPS), which provides an intuitive measure of real-time capability and allows direct comparison with commonly accepted ADAS performance thresholds. Furthermore, the 99th-percentile (p99) latency was included to characterize tail-latency behavior under worst-case conditions. This metric is particularly critical for safety-related systems, where sporadic latency spikes can lead to delayed perception and potentially hazardous decision-making.

The definitions and mathematical formulations of all evaluation metrics are summarized in Table 2. Collectively, these metrics provide a balanced and rigorous basis for comparing accuracy–latency trade-offs and assessing the practical feasibility of CPU-based object detection for low-speed ADAS applications.

Table 2: Definitions and formulas of evaluation metrics used in this study.

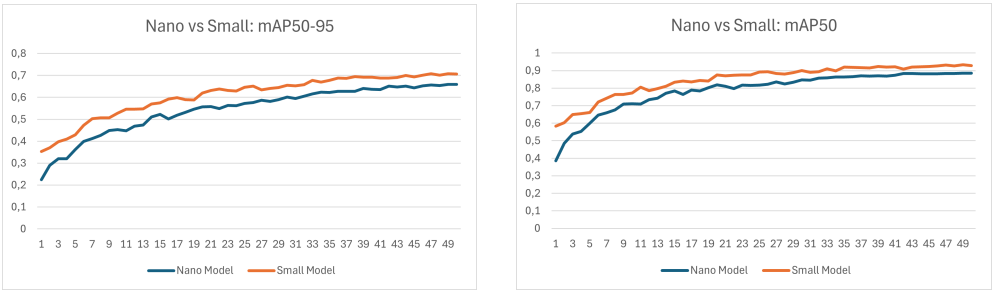
Metric	Description / Formula
E2E Latency (ms)	End-to-end latency measures the total time elapsed from input frame acquisition to final output generation, including preprocessing, inference, and post-processing stages. It is computed as: $$T_{\text{E2E}} = T_{\text{pre}} + T_{\text{inf}} + T_{\text{post}}$$
Inference Time (ms)	Inference time represents the pure forward-pass execution time of the neural network, excluding data loading and post-processing. It is measured as the average per-frame inference duration: $$T_{\text{inf}} = \frac{1}{N}\sum_{i=1}^{N} t_i$$
Effective FPS	Frames Per Second (FPS) indicates the number of frames processed per second and is inversely proportional to the average inference time: $$\text{FPS} = \frac{1000}{T_{\text{E2E}}}$$
p99 Latency (ms)	The 99th-percentile latency represents the upper-bound inference delay under worst-case conditions. It is defined as the latency value below which 99% of the measured inference times fall, highlighting tail-latency behavior critical for real-time systems.

4 Results

4.1 KITTI Dataset Performance

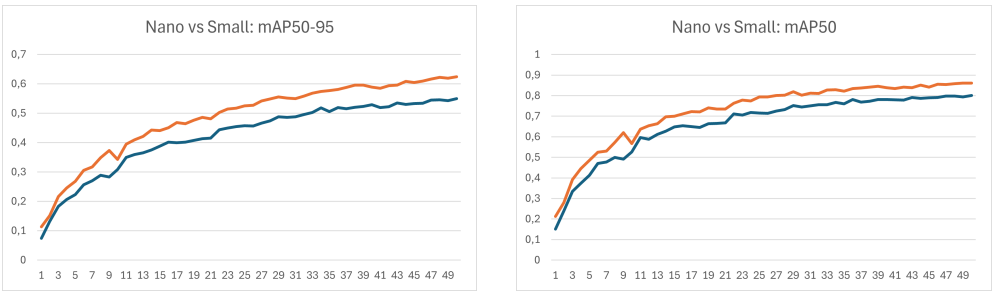
Training metrics for YOLOv11 (Fig. 1) indicated that the Small model achieved higher convergence accuracy. However, for edge deployment scenarios, the trade-off between inference speed and accuracy is critical. For the YOLOv8 training metrics are showed in Fig 2

Table 3 presents the quantitative impact of INT8 quantization on the KITTI dataset. The Nano model achieved a ∼146% increase in inference speed (from 5.29 to 13.02 FPS).



(a) mAP50-95 (b) mAP50

Fig. 1: Comparative training metrics for YOLOv11 on the KITTI dataset. The Small model (orange) demonstrates higher accuracy but higher computational cost.



(a) mAP50-95 (b) mAP50

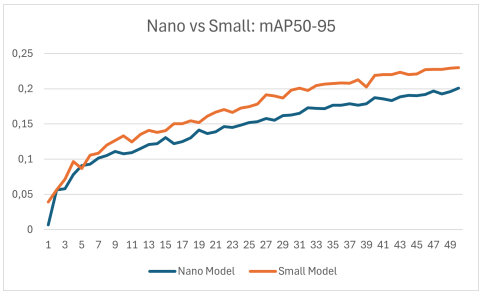
Fig. 2: Comparative training metrics for YOLOv8 on the KITTI dataset. The Small model (orange) demonstrates higher accuracy but higher computational cost.

Table 3: Inference Performance and Accuracy Trade-off (KITTI Dataset)

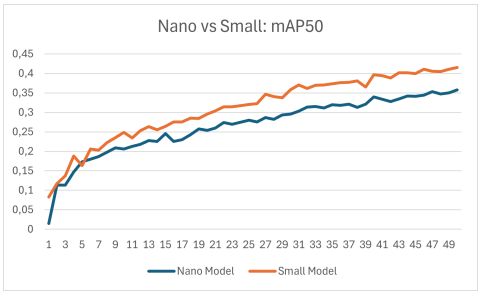
Model	Format	E2E Latency	Effective FPS	Inference Latency	p99 Latency
v11 Nano	FP32	88.4	5.29	189.17	257.81
	INT8	76.78	13.02	76.68	142.78
v11 Small	FP32	93.1	5.27	189.81	229.89
	INT8	152.20	6.57	152.9	275.00
v8 Nano	FP32	391.68	2.55	391.57	506.99
	INT8	103.52	9.65	103.42	216.53
v8 Small	FP32	392.20	2.54	392.09	509.36
	INT8	93.04	10.74	113.33	177.45

4.2 BDD100K Dataset Performance

Similar performance trends were observed with the BDD100K dataset (Table 4). The Nano model with INT8 quantization provided the highest throughput, establishing it as the most viable candidate for real-time applications on the Raspberry Pi 5. Fig 3 shows mAP50/50-95 values for YOLOv11-based models.

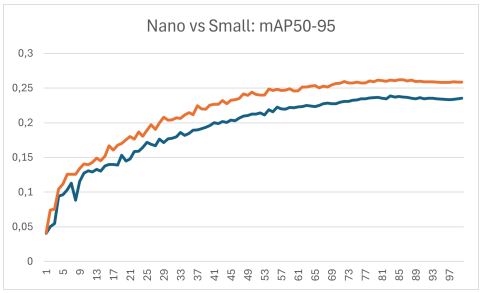


(a) mAP50-95

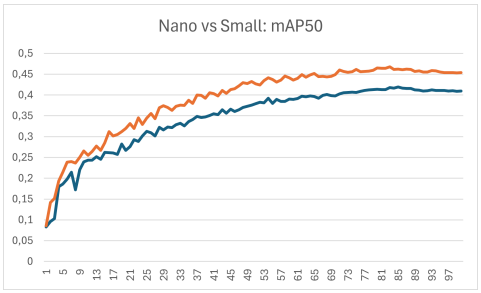


(b) mAP50

Fig. 3: Comparative training metrics for YOLOv11 on the BDD100K dataset. The Small model (orange) demonstrates higher accuracy but higher computational cost.



(a) mAP50-95



(b) mAP50

Fig. 4: Comparative training metrics for YOLOv8 on the BDD100K dataset. The Small model (orange) demonstrates higher accuracy but higher computational cost.

On the YOLOv8 training same hyperparameters were used. Similar performance were observed with the YOLOv8 training on the BDD100K dataset (Table 4). Fig 4 shows mAP50 and mAP50-95 values for whole training.

Table 4: Inference Performance (BDD100K Dataset)

Model	Format	E2E Latency	Effective FPS	Inference Latency	p99 Latency
v11 Nano	FP32	195.38 ms	5.12	195.27 ms	383.54 ms
	INT8	77.03 ms	12.98	79.92 ms	143.47 ms
v11 Small	FP32	433.14 ms	2.31	433.03 ms	718.56 ms
	INT8	147.89 ms	6.76	147.78 ms	263.83 ms
v8 Nano	FP32	179.31 ms	5.57	179.20 ms	262.16 ms
	INT8	49.67 ms	20.31	49.57 ms	94.93 ms
v8 Small	FP32	407.29 ms	2.45	407.18 ms	532.76 ms
	INT8	92.96 ms	10.75	92.86 ms	171.86 ms

5 Discussion

5.1 KITTI Dataset Performance

The Nano models outperformed their Small counterparts in terms of latency and throughput under CPU-only execution. While Small variants achieve higher mAP values, this study prioritizes latency-oriented performance. From this perspective, Nano-scale models provide a more favorable trade-off for real-time, CPU-based ADAS scenarios. INT8 quantization significantly alleviates this limitation by reducing model size and memory bandwidth requirements, resulting in substantial FPS gains and improved p99 latency. In particular, the YOLOv11 Nano INT8 model demonstrates a more favorable speed–accuracy trade-off compared to YOLOv8-based alternatives, making it a suitable candidate for low-speed ADAS perception tasks in resource-constrained environments.

5.2 BDD100K Dataset Performance

YOLO-based deep learning models were employed in this study, including YOLOv11n, YOLOv11s, YOLOv8n, and YOLOv8s. The inclusion of multiple models was motivated by the need to compare Nano and Small variants with respect to architectural differences and performance characteristics. In particular, although the YOLOv11 architecture is newer than YOLOv8, notable differences were observed in terms of inference speed.

This study primarily focused on speed-related metrics, including end-to-end (E2E) latency and inference latency, as these metrics provide a clear basis for performance analysis. After training, all models were converted to the ONNX format with 30% pruning and INT8 post-training quantization.

The YOLOv11 Nano model in FP32 format achieved an E2E latency of 195.38 ms with 5.12 FPS, whereas its INT8-quantized counterpart reduced the E2E latency to 77.03 ms and increased throughput to 12.98 FPS. In contrast, the YOLOv11 Small model exhibited an E2E latency of 433.14 ms with 2.31 FPS in FP32 format, which improved to 147.89 ms and 6.76 FPS after INT8 quantization.

These results indicate that the INT8-quantized Nano model outperforms the Small model in terms of speed, despite the Small model achieving higher accuracy in mAP-based evaluations.

6 Conclusion

This study presented a comprehensive benchmark of YOLOv11 and YOLOv8 on Raspberry Pi 5 [1]. demonstrating that converting models to ONNX format with INT8 quantization yields a 146% increase in FPS Accuracy metrics were not treated as primary optimization objectives; instead, they were used to verify that quantization did not result in catastrophic degradation [10]. While pure CPU inference is limited to ∼13 FPS, this performance is sufficient for low-speed ADAS functions [9, 11]. Future work will explore the integration of Neural Processing Units (NPUs) to achieve 30+ FPS [3, 12].

Declarations

- **Funding:** This research received no external funding.
- **Conflict of Interest:** The author declares no conflict of interest.
- **Data Availability:** The datasets used (KITTI, BDD100K) are publicly available.

References

[1] Chaman, M., Hadjoudja, A., Laamari, H., et al.: A real-time vehicle detection system for adas in autonomous vehicles using yolov11 deep neural network on embedded edge platforms. Engineering, Technology & Applied Science Research **15**(1), 28077–28082 (2025). DOI 10.48084/etasr.12138

[2] Dhatrika, S.K., Reddy, D.R., Reddy, N.K.: Real-time object recognition for advanced driver-assistance systems (adas) using deep learning on edge devices. Procedia Computer Science **252**, 25–42 (2025). DOI 10.1016/j.procs.2024.12.004

[3] Hussain, K., Moreira, C., Pereira, J., Jardim, S., Jorge, J.: A comprehensive literature review on modular approaches to autonomous driving: Deep learning for road and racing scenarios. Smart Cities **8**(3), 79 (2025)

[4] Jajal, P., Jiang, W., Tewari, A., et al.: Interoperability in deep learning: A user survey and failure analysis of onnx model converters. In: Proceedings of the 33rd ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA '24), pp. 13–23 (2024)

[5] Jocher, G., Chaurasia, A.: Ultralytics yolov8. https://github.com/ultralytics/ultralytics (2023). Version 8.0.0

[6] Kwasniewska, A., Das, A., Szankin, M., et al.: Deep learning optimization for edge devices: Analysis of training quantization parameters. In: IECON 2019 - 45th Annual Conference of the IEEE Industrial Electronics Society, pp. 96–101 (2019). DOI 10.1109/IECON.2019.8927153

[7] Ma, N., Zhang, X., Zheng, H.T., Sun, J.: Shufflenet v2: Practical guidelines for efficient cnn architecture design. In: Proceedings of the European conference on computer vision (ECCV), pp. 116–131 (2018)

[8] Martínez, H., Castelló, A., Igual, F.D., Quintana-Ortí, E.S.: The cambrian explosion of mixed-precision matrix multiplication for quantized deep learning inference. Future Generation Computer Systems **177**, 108231 (2025)
[9] Neelam Jaikishore, C., Podaturpet Arunkumar, G., et al.: Implementation of deep learning algorithm on a custom dataset for advanced driver assistance systems applications. Applied Sciences **12**(18), 8927 (2022). DOI 10.3390/app12188927
[10] Ratul, I.J., Zhou, Y., Yang, K.: Accelerating deep learning inference: A comparative analysis of modern acceleration frameworks. Electronics **14**(15), 2977 (2025). DOI 10.3390/electronics14152977
[11] Salakapuri, R., Navuri, N.K., Vobbilineni, T., et al.: Integrated deep learning framework for driver distraction detection and real-time road object recognition in advanced driver assistance systems. Scientific Reports **15**, 25125 (2025). DOI 10.1038/s41598-025-08475-4
[12] Shah, I.A., Li, J., George, R., Brophy, T., Ward, E., Glavin, M., Jones, E., Deegan, B.: Hyperspectral sensors and autonomous driving: Technologies, limitations, and opportunities. IEEE Open Journal of Vehicular Technology **6**, 120–135 (2025). DOI 10.1109/OJVT.2025.3636075
[13] Tian, Y., Xu, W., Yang, B., et al.: Development and evolution of yolo in object detection: A survey. Neurocomputing **669**, 132436 (2025). DOI 10.1016/j.neucom.2025.132436
[14] Yu, F., Chen, H., Wang, X., et al.: Bdd100k: A diverse driving dataset for heterogeneous multitask learning. arXiv preprint arXiv:1805.04687 (2020)