

Original Article

TPU Accelerated Multi-Resolution and Occupancy based Performance Analysis of Embedded Real-Time Traffic Sign Detection on Raspberry Pi 5

Thotapalli Vinayakumar¹, Samineni Peddakrishna²

^{1,2}School of Electronics Engineering, VIT-AP University, Inavolu, Amaravathi, Andhra Pradesh, India.

²Corresponding Author : samineni.peddakrishna@vitap.ac.in

Received: 08 August 2026

Revised: 15 September 2026

Accepted: 24 September 2026

Published: 28 October 2026

Abstract - Embedded traffic-sign detection must balance accuracy against the compute, memory and power limits of the target platform. This study evaluates YOLOv8n on TT100K at six independently trained input resolutions (640×640, 512×512, 480×480, 320×320, 224×224 and 160×160) and on three platform configurations: an NVIDIA GeForce RTX 3050 laptop GPU, a Raspberry Pi 5 CPU and a Raspberry Pi 5 with a Google Coral USB Edge TPU. Validation accuracy decreased monotonically from 0.812 mAP@50 at 640×640 to 0.088 at 160×160. By contrast, measured Edge TPU throughput remained between 87.70 and 89.33 FPS, so downsampling produced no meaningful throughput gain on the evaluated configuration while substantially reducing accuracy. Normalized sign occupancy was then related to recall for all 7,471 annotations in the test subset. Recall increased from 20.50% below 0.005% occupancy to 98.17% above 0.10%, with a transition region between approximately 0.02% and 0.05% observed for this dataset and model configuration. The combined analysis provides a platform-specific basis for selecting input resolution and camera geometry.

Keywords - Traffic sign detection, YOLOv8n, Edge AI, Raspberry Pi 5, Google Coral Edge TPU, Multi-Resolution analysis, Occupancy-based evaluation.

1. Introduction

Traffic sign detection is an essential perception function in Intelligent Transportation Systems (ITS). Speed limits, warnings, prohibitions and mandatory instructions are all conveyed through signs, so a vehicle that misses them cannot drive lawfully or safely. For the same reason, sign detection is a required function in Advanced Driver-Assistance Systems (ADAS), where one missed sign can directly lead to a safety risk [1]. With the gradual move of driving automation from test vehicles to commercial products, sign detection is now treated as a basic part of the perception system.

Earlier methods for this task followed three main stages. The first detectors located candidate regions by colour, checked their shape against templates and classified them with manually designed descriptors. They worked on clear daytime images, but their accuracy dropped when lighting, viewing angle or background clutter changed. Machine-learning classifiers based on engineered features gave some improvement, but their accuracy still depended on how well the features were designed, and they did not transfer well between cities, seasons and cameras [2]. Convolutional networks removed this dependence because the features are learned from data. Among them, the single-stage YOLO

detectors reached accuracy close to two-stage detectors at a much lower inference cost [3].

Most recent traffic-sign detectors are built on this single-stage design and try either to reduce the size of the network or to improve its multi-scale features. Wang et al. [4] strengthened the semantic features of the detector so that localization remains accurate under different driving conditions. Chen et al. [5] proposed YOLO-TS, which combines optimized receptive fields with anchor-free prediction. Wang et al. [6] modified the YOLOv8 backbone for crowded traffic scenes, and Ji et al. [7] reported that adding multi-scale fusion to a lightweight YOLOv8 model improves accuracy with little loss in speed.

These improvements do not remove the basic difficulty. In the TT100K test subset used here, the median sign covers only 0.0326% of the image area, and rain, glare, motion blur, partial occlusion and cluttered backgrounds further reduce the useful information in those few pixels. Li et al. [8] used partial convolutions to keep fine local detail while suppressing background response; Zhang et al. [9] proposed a granular perception feature pyramid for cross-scale interaction; and Tian and Zhang [10] fused heterogeneous features with guided



attention to separate signs from cluttered surroundings. All three methods change the network itself. None of them reports how the same network behaves after it is quantized, compiled for an accelerator and run on a single-board computer, which is the situation studied in this paper.

Edge AI research approaches the problem from the deployment side. Kamath et al. [11] documented the data preparation and training workflow needed before an object detector can be hosted on a Raspberry Pi. Minott et al. [12] compared NVIDIA Jetson boards with a Raspberry Pi 5 carrying a Coral TPU and observed a large speed-up over CPU-only execution. Kurniawan et al. [13] found that throughput on heterogeneous edge boards depends jointly on compute capability and memory bandwidth rather than on network size alone, and Alqahtani et al. [14] framed practical edge deployment as an explicit accuracy-versus-cost decision.

These two groups of studies are rarely combined. Architecture studies usually report Precision, Recall, mAP and sometimes frame rate at one fixed input size, whereas deployment studies benchmark complete hardware-software configurations without resolving performance by object size. As a result, input resolution, execution platform and normalized sign occupancy have rarely been evaluated together. What is missing is a controlled measurement that varies input resolution while reporting deployment throughput and scale-resolved reliability for the same trained models. That is the research gap addressed in this study.

To address this gap, YOLOv8n was trained six times on the TT100K benchmark [15], once for each input resolution from 160×160 to 640×640, with all other settings kept the same. Each model was then run on an NVIDIA GeForce RTX 3050 laptop GPU, on a Raspberry Pi 5 (RPi 5) CPU, and on an RPi 5 fitted with a Google Coral USB Edge TPU (referred to below as RPi 5 + Edge TPU). In addition to the usual detection metrics, every annotated sign in the test subset was given a normalized occupancy value, and recall was computed for each occupancy band. This replaces the general statement that small signs are harder to detect with a measured curve that shows where the change occurs. Table 1 compares this scope with representative earlier studies: those works optimize the architecture at a single resolution and seldom report embedded throughput, whereas the present study measures resolution, hardware and object scale together for the same trained models. This combined treatment, and the occupancy transition it reveals, forms the novelty of the work.

The main contributions of the paper are listed below.

- Six YOLOv8n detectors, trained and evaluated at six input resolutions, quantify the accuracy loss caused by downsampling and the corresponding throughput change on each evaluated platform configuration.
- The same six models, quantized to INT8 and compiled for the Edge TPU, are benchmarked on a desktop GPU, the

RPi 5 CPU and the RPi 5 + Edge TPU configuration.

- An occupancy-based evaluation procedure relates normalized traffic-sign area to recall, exposing scale-dependent behaviour that aggregate dataset-level metrics conceal.
- An empirical occupancy transition band is identified for the evaluated dataset, detector and settings, providing a measured basis for deployment design without presenting the band as a universal threshold.

The rest of the paper is arranged as follows. Section 2 reviews detector architectures, small-object techniques and edge-deployment benchmarks and places the present study among them. Section 3 covers the dataset, the six training runs, the export and quantization pipeline, the hardware and software environment and the occupancy procedure. Section 4 reports accuracy across resolutions, throughput across platforms, the resulting trade-off and the occupancy analysis. Section 5 gives the conclusions and the limits of what was measured.

2. Related Work

Over about ten years, traffic sign detection has changed from hand-designed image-processing pipelines to learned detectors that can run in real time on busy road scenes. The first systems relied on the features that make signs easy for drivers to see: strong colour, regular shape and high contrast with the background.

Yıldız and Dizdaroğlu [16] combined colour thresholding with shape analysis and obtained accurate detection on controlled video, but they noted that performance fell when the lighting changed, when signs were partly hidden or when the background was complex. Timofte et al. [17] handled viewpoint changes by combining appearance learning with three-dimensional localization from several camera views. This improved detection over a wide range of angles, although the pipeline became much heavier. On the hardware available at that time, neither method was fast enough for continuous use in a vehicle.

Convolutional networks improved accuracy considerably because they learn hierarchical features directly instead of using hand-designed descriptors. Vennelakanti et al. [18] combined several convolutional models in an ensemble to obtain more stable results across different traffic scenes. Zhang et al. [19] designed a cascaded R-CNN with multi-scale attention and a sampling scheme that accounts for class imbalance; it detected small signs more accurately and produced fewer false positives. However, two-stage detectors of this type need much more computation, and their latency is too high for embedded processors. With one-stage detectors, the aim shifted to obtaining the highest accuracy within a fixed computational budget, and much of the later traffic-sign work modifies the YOLO family for this purpose. Zhang et al. [20] redesigned the feature extraction and localization of YOLOv8

in YOLO-BS to handle large variations in sign scale and appearance. Meng and Shi [21] modified YOLOv7 to reuse features more and to obtain richer semantics, which improved small-sign accuracy without increasing the number of parameters much. These studies show that targeted changes to the backbone and the detection head can raise accuracy at a small extra cost. In all of them, however, the input resolution is fixed at one value, and the detector is tested on a workstation GPU, so the effect of quantization and the behaviour on embedded hardware are not reported.

A distant sign may keep only a few pixels after downsampling, and therefore, a second group of studies focuses on multi-scale representation. Liu et al. [22] cascaded and recursively fused shallow and deep features to improve the localization of small signs. ES-YOLO [23] added edge and shape information to feature extraction to obtain clearer boundaries in difficult road conditions. Chen et al. [24] adapted a lightweight YOLOv8s for embedded autonomous-driving systems while keeping real-time inference. Dewi et al. [25] used spatial pyramid pooling to widen the range of receptive fields so that very small signs can still be detected, and Deng et al. [26] extended the feature pyramid to improve cross-scale interaction for small objects in general. Manzari et

al. [27] applied transformer-based feature learning, using long-range context to separate signs from similar-looking background structures. Qiu et al. [28] proposed DP-YOLO, which uses decoupled prediction heads to detect small objects more efficiently.

Further progress in feature representation has come from multi-scale learning and attention mechanisms. Using large-scale traffic-sign data, Tabernik and Skočaj [29] showed that stronger feature aggregation clearly improves localization in complex scenes. HMS-YOLO [30] extracts features hierarchically at several scales and separates objects by size before enriching their semantics. PST-YOLO [31] combines progressive scale fusion with polynomial feature enhancement so that small signs remain detectable in poor outdoor conditions. Context-Aware Block Net [32] refines features adaptively to separate signs from the background, and Dynamic Head [33] brings scale, spatial and task attention into a single detection head so that feature selection can adapt to each scale. According to these studies, better feature representation is the main way to improve small-sign detection. Still, none of them relates the reported accuracy to the hardware on which the detector will finally run, or to the object sizes at which that accuracy is achieved.

Table 1. Comparison of existing studies and proposed work

Ref	Method	Small object	Edge deploy.	Multi-res.	Occupancy	Main limitation
[20]	YOLO-BS	Yes	No	No	No	Focused on detector architecture improvement.
[21]	Improved YOLOv7	Yes	No	No	No	Architecture-oriented evaluation
[23]	ES-YOLO	Yes	No	No	No	Hardware performance not analyzed.
[24]	Lightweight YOLOv8s	Yes	No	No	No	Resolution effect not analyzed
[25]	Spatial Pyramid Pooling	Yes	No	No	No	Object occupancy is not considered.
[32]	Context-Aware Block Net (CAB Net)	Yes	No	No	No	No embedded deployment
[12]	Edge Platform Comparison	No	Yes	No	No	No detector behavior analysis
[36]	Edge Benchmark	No	Yes	No	No	Object-scale effect not investigated
Proposed	YOLOv8n + Multi-Resolution Occupancy Analysis	Yes	Yes	Yes	Yes	Comprehensive evaluation across heterogeneous platforms

A separate literature examines that hardware directly. Benchmarks consistently find that deployed performance is governed by compute capability, memory bandwidth and runtime support rather than by network architecture alone.

Comparative evaluation of several deep models across edge devices showed the decisive trade-off to be accuracy against inference speed on heterogeneous boards [14].

Santos et al. [34] measured YOLOv7 on edge hardware and confirmed that one-stage detectors run substantially faster than two-stage alternatives at competitive accuracy.

Amanatidis et al. [35] distributed detection work across cooperating edge nodes to improve utilization and responsiveness.

More recently, Le et al. [36] benchmarked deep models for edge robotics and argued that hardware-aware optimization is indispensable under tight compute budgets.

Minott et al. [12] quantified specifically the benefit of attaching a Coral Edge TPU to a Raspberry Pi 5, which is the configuration adopted in the present study. What is missing in both groups is a study that links them. Architecture-centred papers optimize networks and report accuracy; deployment-centred papers profile hardware and report latency.

Consequently, there is little evidence on how input resolution, the choice of edge accelerator and the physical size of the sign in the image act together on detector reliability, and the relationship between normalized sign occupancy and lightweight-detector performance has scarcely been quantified at all.

Answering that question requires a single evaluation framework in which accuracy, deployment efficiency and object scale are measured for the same trained models. That is the approach taken in this work, and Table 1 summarizes how its scope differs from the studies reviewed above.

3. Methodology

This section describes how the effect of input image resolution on traffic-sign detection accuracy and real-time deployment performance was measured.

As shown in Figure 1, the procedure has five stages: (i) dataset preparation, (ii) training and model selection for YOLOv8n at six input resolutions, (iii) TensorFlow Lite conversion and Edge TPU compilation, (iv) deployment on three computing platforms, and (v) evaluation with standard detection metrics and an occupancy-based reliability analysis. Aggregate metrics alone show how well the detector performs overall. The occupancy analysis adds the object size at which its behaviour changes, and this size decides whether a given

camera and input resolution will work at a required detection distance.

The training hyperparameters, the experimental environment and the inference settings are listed separately in Table 3, Table 5 and Table 6.

3.1. TT100K Dataset Description

TT100K was used for all experiments. It is a traffic-sign benchmark made from high-resolution street-level panoramas recorded under different weather and lighting conditions.

The dataset includes regulatory, warning and prohibitory signs with large variation in object scale, viewpoint and scene complexity, so it is suitable for studying detection accuracy and deployment behaviour together.

The annotations were converted to the Ultralytics YOLO format, where every image has a text file with the class index and the normalized bounding-box coordinates of each object. A total of 50 traffic-sign classes were used.

The images were split into training, validation and test subsets in the proportion 70/20/10, and Table 2 gives the number of images and traffic-sign instances in each subset. The same fixed subsets were used for all six input resolutions.

Table 2. TT100K dataset composition used in this study

Split	Images	Traffic-sign instances	Share of dataset (%)
Training	16,916	51,483	70
Validation	4,833	14,885	20
Test	2,417	7,471	10
Total	24,166	73,839	100

3.2. Multi-Resolution YOLOv8n Training and Model Selection

YOLOv8n was used as the detector throughout. Six models were trained independently, one per input resolution, so that input resolution is the only variable distinguishing them.

Each model started from the official YOLOv8n COCO-pretrained weights and was fine-tuned on the TT100K training split at its assigned resolution: 640×640, 512×512, 480×480, 320×320, 224×224 or 160×160 pixels.

Optimizer, learning-rate schedule, augmentation policy, epoch count and batch size were held identical across the six runs; only the input size differed. Table 3 lists the training configuration.

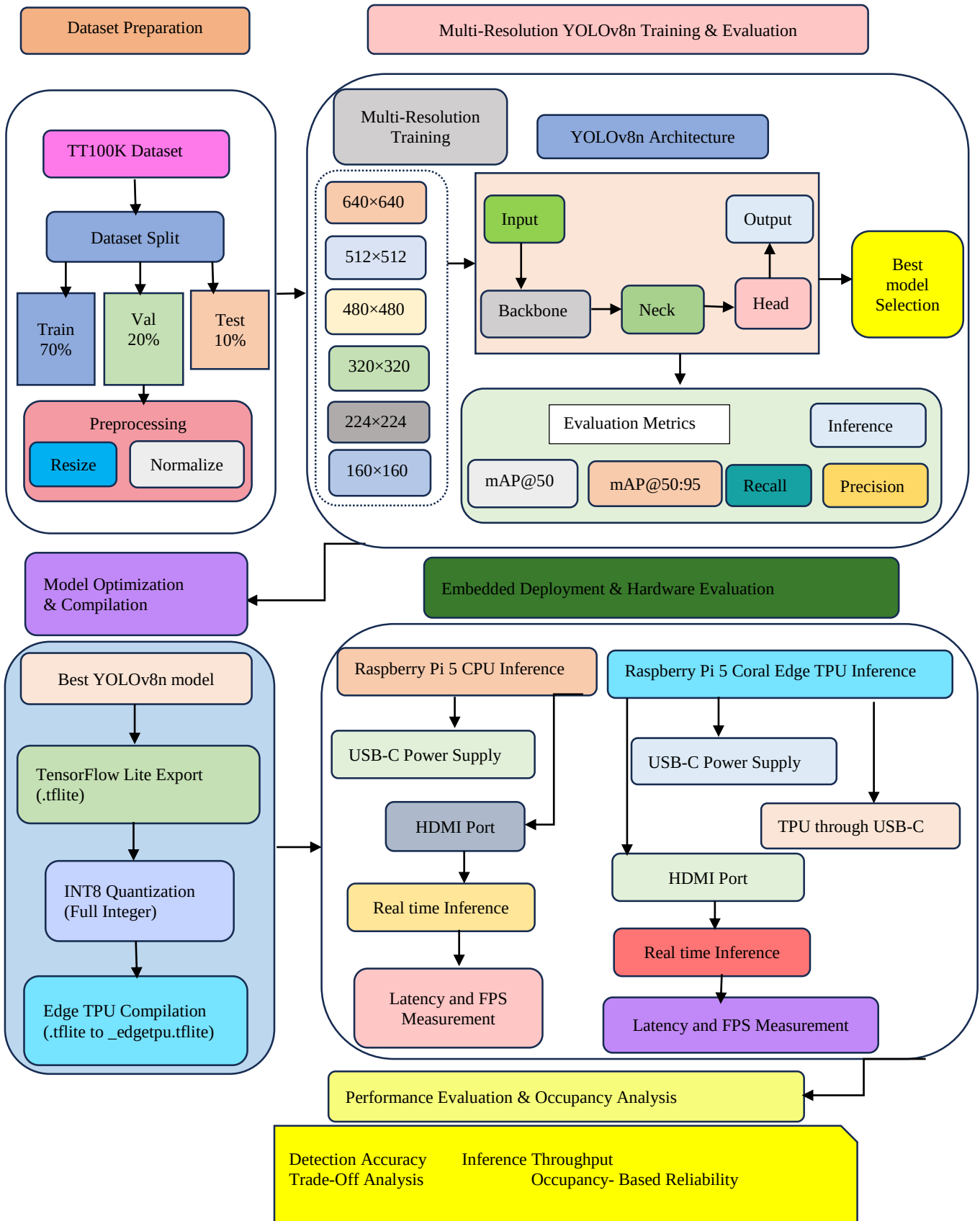


Fig. 1 Overall workflow of the proposed multi-resolution edge deployment framework

Table 3. Training configuration and hyperparameters

Parameter	Value
Detector	YOLOv8n (Ultralytics implementation)
Weight initialization	Official COCO-pretrained yolov8n checkpoint
Input resolutions trained	640, 512, 480, 320, 224, 160 (six independent runs)
Epochs	100
Optimizer	SGD (Ultralytics automatic selection, optimizer = auto), momentum 0.9
Initial learning rate (lr0)	0.01
Batch size	16
Model-selection criterion	Highest validation mAP@50:95 (Ultralytics best.pt)
Training hardware	NVIDIA GeForce RTX 3050 6 GB Laptop GPU

All six runs used 100 epochs and a batch size of 16. The optimizer was left at the Ultralytics automatic setting, which, for a training set of this size, selects Stochastic Gradient Descent (SGD) with an initial learning rate of 0.01 and a momentum of 0.9.

Precision, Recall, mAP@50 and mAP@50:95 were computed on the validation subset after every epoch. For each resolution, the checkpoint with the highest validation mAP@50:95 (the fitness criterion of Ultralytics 8.3.204) was saved as best.pt, and only this checkpoint was used for export, quantization and deployment.

3.3. Model Export, Quantization, and Edge Deployment

After training, the selected YOLOv8n checkpoint for each input resolution was optimized for execution on the Google Coral Edge TPU. The Edge TPU cannot execute PyTorch models directly, so a three-stage conversion pipeline was used to bring each trained detector into a hardware-compatible form. For each resolution, the selected PyTorch checkpoint was exported with the Ultralytics export function and converted into a full-integer INT8 TensorFlow Lite model.

A representative calibration set taken from the TT100K training subset was used during this conversion. After this step, both weights and activations are stored as 8-bit integers, as required by the Edge TPU.

Each INT8 TensorFlow Lite model was then compiled with the Edge TPU Compiler (version 16.0.384591198). For all six resolutions, the compiler report shows that all 257 operations of the model were mapped to the Edge TPU and none were left for the host CPU. Only the compiled

_edgetpu.tflite models were used for the accelerated measurements. The compiled Edge TPU models were run on the RPi 5 with the Coral USB Accelerator connected, and the corresponding INT8 TensorFlow Lite models were run on the same board without the accelerator. The two runtime configurations are listed in Table 4.

Table 4. Runtime configurations for embedded deployment

Configuration	Runtime
CPU-Only	Raspberry Pi 5 + INT8 TFLite Interpreter
Edge TPU	Raspberry Pi 5 + Coral USB Accelerator + PyCoral Runtime

On the RPi 5 CPU and RPi 5 + Edge TPU configurations, the models came from the same selected checkpoints and were timed with the same script and the same confidence and non-maximum-suppression thresholds. The comparison, therefore, describes complete platform configurations, including model format and runtime, and not the hardware alone.

3.4. Experimental Platform and Performance Evaluation

Three platforms representing different computing conditions were used. An NVIDIA GeForce RTX 3050 laptop GPU was used for all training and gave the upper reference for throughput. The RPi 5 ran the INT8 TensorFlow Lite models on its CPU, representing embedded inference without acceleration. The same RPi 5 with the Coral USB Edge TPU connected ran the compiled Edge TPU models, representing accelerated embedded inference. Table 5 lists the hardware, operating systems and software versions of the three platforms.

Table 5. Experimental environment: Hardware platforms and software / runtime versions

Item	Desktop GPU	Raspberry Pi 5 (CPU)	Raspberry Pi 5 + Coral Edge TPU
Processor / Accelerator	NVIDIA GeForce RTX 3050 6 GB Laptop GPU	Broadcom BCM2712, quad-core Arm Cortex-A76 @ 2.4 GHz	BCM2712 host + Google Coral USB Edge TPU
Operating system	Windows 11 (64-bit)	Raspberry Pi OS (Debian 12 Bookworm, 64-bit)	Raspberry Pi OS (Debian 12 Bookworm, 64-bit)
Software/runtime versions	Python 3.13.7, PyTorch 2.8.0 (CUDA 12.6), Ultralytics 8.3.204	TensorFlow Lite runtime 2.14.0	TensorFlow Lite runtime 2.14.0, libedgetpu1-max 16.0; models compiled with Edge TPU Compiler 16.0.384591198
Deployed model format	PyTorch FP32 (.pt)	INT8 TFLite (.tflite)	INT8 Edge TPU (_edgetpu.tflite)



Fig. 2 Physical test setup: RPi 5 with the Coral USB Edge TPU running live inference

Figure 2 shows the physical setup, with the RPi 5 and the Coral USB Edge TPU running live inference. The embedded deployment results in Section 4.2 were measured on this setup.

All three platforms ran models derived from the same six selected checkpoints. The desktop GPU used the PyTorch FP32 model, the RPi 5 CPU used the INT8 TensorFlow Lite model, and the Edge TPU used the compiled INT8 Edge TPU model, each at its own training resolution.

The settings used for each type of measurement are given in Table 6. Detection accuracy is reported as Precision, Recall, mAP@50 and mAP@50:95. mAP@50 uses one intersection-over-union (IoU) threshold of 0.5, while mAP@50:95 is the average over IoU thresholds from 0.5 to 0.95 in steps of 0.05. mAP@50:95 is the stricter of the two, because a box that is only roughly in the right place still loses credit at the higher thresholds.

Throughput is reported in Frames Per Second (FPS). On the desktop GPU, FPS was calculated as 1000 divided by the sum of the per-image preprocessing, inference and post-processing times (in milliseconds) reported by Ultralytics over the 4,833 validation images. On the RPi 5 CPU and RPi 5 + Edge TPU configurations, latency was measured with the benchmarking script visible in Figure 2: after the model was loaded, 50 consecutive inferences were timed, and the mean latency was converted to FPS.

3.5. Occupancy-based Detection Analysis

Precision, recall and mean average precision report how well a detector performs on a dataset; they do not report at what physical object size that performance is obtained. Because traffic signs cover only a small share of the frame, particularly at the distances at which a vehicle needs to react, a second analysis was carried out in terms of object occupancy, so that detection reliability could be read directly against object size.

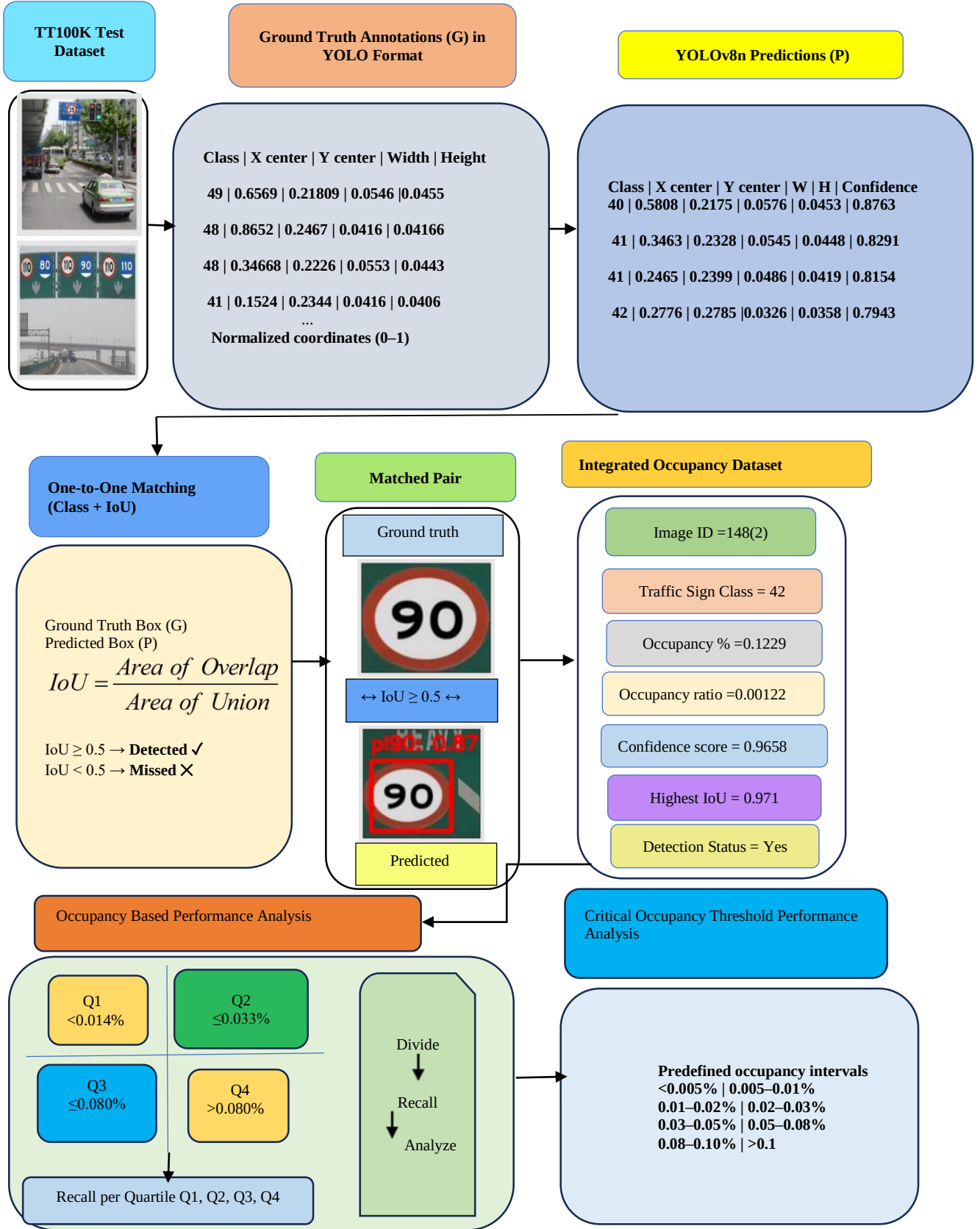


Fig. 3 Workflow of the proposed occupancy-based evaluation framework

Table 6. Inference and evaluation settings

Setting	Accuracy evaluation (validation subset)	Occupancy analysis (test subset)	RPi 5 CPU and RPi 5 + Edge TPU timing
Confidence threshold	0.001 (Ultralytics validation default)	0.25	0.05
NMS IoU threshold	0.70	0.70	0.65
Images	4,833 validation images	2,417 test images (7,471 signs)	Benchmarking script (Figure 2)
Prediction-to-annotation matching	Ultralytics mAP protocol	One-to-one, same class, IoU ≥ 0.50	Not applicable
Throughput measure	1000 / (pre-processing + inference + post-processing time per image, ms)	Not applicable	Mean latency over 50 consecutive runs after model loading

Occupancy was computed for every annotated traffic sign in the TT100K test split from the normalized ground-truth bounding-box dimensions as

$$\text{Occupancy (\%)} = w_n \times h_n \times 100 \quad (1)$$

where w_n and h_n are the bounding-box width and height divided by the image width and height, respectively. The TT100K labels are stored in normalized YOLO coordinates, so the occupancy from Equation (1) does not depend on the pixel size to which an image is resized. It is therefore a consistent measure of object size over the whole dataset and for all six input resolutions.

The selected detector was then applied to the full TT100K test subset to obtain the predicted boxes and their confidence scores. Predictions were paired with ground-truth annotations on a one-to-one basis; a pair was accepted only when both had the same class label and an IoU of 0.5 or more.

An annotated sign with an accepted pair was counted as detected, and one without a pair was counted as missed. Because each annotation can be paired only once, no sign is counted twice in the statistics.

Figure 3 summarizes the complete occupancy procedure. After matching, the computed occupancy values were joined with the detection outcomes into a single analysis file holding the image identifier, the traffic-sign class, the occupancy ratio and percentage, the detection outcome, the highest intersection-over-union achieved and the confidence score, which allows detection results to be grouped and examined by object scale.

Two complementary analyses were then carried out on that file. The first is distributional: descriptive statistics and a histogram characterize how traffic-sign sizes are distributed across the test split, quartile boundaries (Q1–Q4) are derived from that distribution, and recall is computed separately within each quartile to relate object scale to detection performance. The second analysis is interval-based. Occupancy values were partitioned into eight predefined

intervals spanning extremely small to comparatively large traffic signs, and recall was computed independently within each interval. This finer partition locates the occupancy range over which detection changes from unreliable to consistently reliable, which the four quartiles are too coarse to resolve. Unlike dataset-level aggregate metrics, this occupancy-based evaluation yields a physically interpretable account of detector behaviour and establishes a direct relationship between normalized traffic-sign scale and detection reliability under embedded deployment.

4. Experimental Results and Discussion

The results are presented in four parts. Detection accuracy is examined first for the six input resolutions using standard object-detection metrics. Inference throughput is then compared on the desktop GPU, the RPi 5 CPU and the RPi 5 + Edge TPU configuration.

Finally, the resolution–accuracy–throughput trade-off and the occupancy-based reliability analysis are presented to relate image resolution, computational efficiency and detection reliability.

4.1. Detection Performance at Multiple Input Resolutions

The effect of input resolution on accuracy was examined first for the six independently trained YOLOv8n models. Table 7 gives mAP@50 and mAP@50:95 of each selected checkpoint on the TT100K validation subset (4,833 images and 14,885 sign instances).

Table 7 shows that input resolution has a decisive effect. Both mAP@50 and mAP@50:95 decrease monotonically as the input is reduced, indicating a progressive loss of both localization and recognition capability.

Reducing the resolution from 640×640 to 512×512 lowered mAP@50 from 0.812 to 0.692, a relative decrease of approximately 14.8%. A further reduction to 480×480 produced 0.629 mAP@50, showing that even moderate downsampling reduced the detector's ability to preserve discriminative traffic-sign features.

Table 7. Detection performance across different input resolutions

Resolution	mAP@50	mAP@50:95
640	0.812	0.690
512	0.692	0.585
480	0.629	0.528
320	0.332	0.261
224	0.179	0.130
160	0.088	0.057

The decline became more pronounced at lower resolutions. At 320×320, mAP@50 was 0.332, approximately 59% below the 640×640 result. The 224×224 and 160×160 models achieved 0.179 and 0.088 mAP@50, respectively. mAP@50:95 followed the same monotonic pattern, indicating progressively poorer localization as the input resolution decreased.

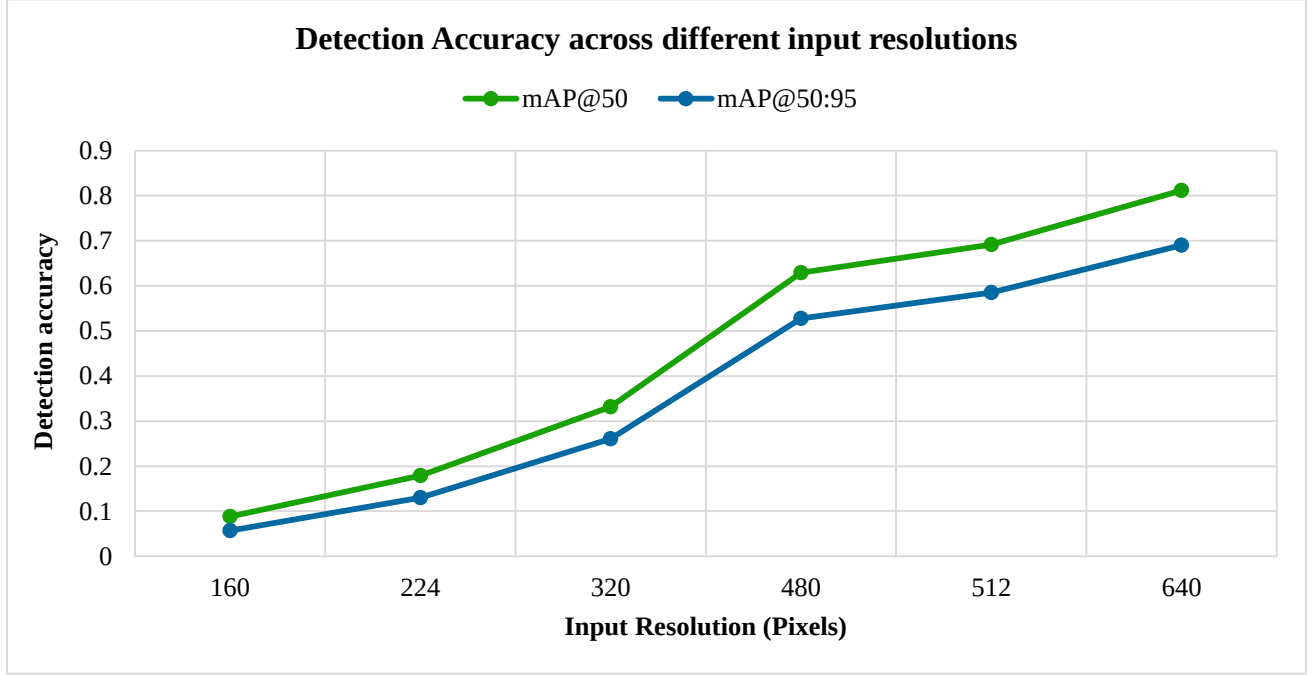
**Fig. 4 Detection accuracy across different input resolutions**

Figure 4 makes the pattern explicit: mAP@50 and mAP@50:95 both fall monotonically as the input size is reduced. The mechanism is straightforward. A sign spanning twenty pixels in a 640×640 input spans five at 160×160, and once the object is smaller than the effective stride of the deeper feature levels, the evidence needed to localize and classify it has already been discarded before the detection head receives it. These results show that input resolution materially affects detection accuracy in the evaluated deployment configurations. Selecting an operating resolution should therefore consider both localization accuracy and measured end-to-end throughput, as discussed in the next subsection.

4.2. Cross-Platform Inference Performance

Throughput was then measured on the three platforms: the NVIDIA GeForce RTX 3050 laptop GPU, the RPi 5 running the INT8 TensorFlow Lite model on its CPU, and the RPi 5 + Edge TPU configuration. The measurement procedure for each platform is described in Section 3.4 and Table 6. The resulting frame rates are listed in Table 8 and plotted in Figure 5. Table 8 shows three distinct throughput behaviours, one for each platform configuration. The desktop GPU scales as expected, with throughput rising as the input shrinks, from

163.93 FPS at 640×640 to 250.00 FPS at 160×160, reflecting the smaller tensors it has to process. Without hardware acceleration, the RPi 5 stays at about 24 FPS for all tested resolutions. Although the network requires less computation at smaller inputs, the frame rate hardly changes, which suggests that on this board, the CPU speed is limited by factors other than the network workload, such as the processor architecture and memory bandwidth.

Table 8. Multi-platform throughput (FPS) of YOLOv8n at different input resolutions

Resolution	GPU FPS	Raspberry Pi 5 (CPU)	Raspberry Pi 5 + Edge TPU
640	163.93	24.22	87.86
512	200.00	24.01	89.33
480	200.00	24.05	88.73
320	227.27	24.04	89.06
224	227.27	23.63	87.70
160	250.00	23.89	89.06

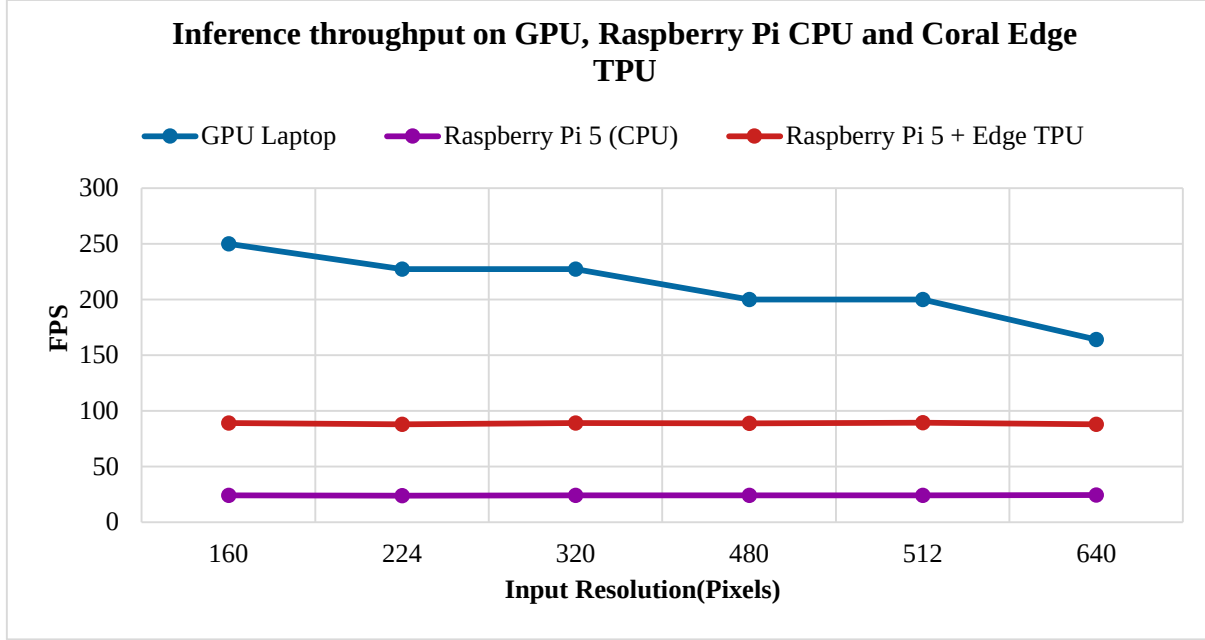


Fig. 5 Inference throughput on GPU, Raspberry Pi CPU and Coral Edge TPU

For the RPi 5 + Edge TPU configuration, the measured throughput stayed between 87.70 and 89.33 FPS for all six input resolutions. In other words, using a smaller input did not give any meaningful gain in speed with the measurement procedure used here.

The three configurations, therefore, behave differently. The GPU frame rate rises as the resolution is reduced, while the two RPi 5 configurations change very little. Since the platforms also use different model formats and runtimes, these results describe the tested systems as a whole and cannot be generalized to other accelerators or software stacks.

Among the three platforms tested, the platform configuration had a much larger effect on embedded inference speed than the input resolution. Adding the Coral USB Edge TPU to the RPi 5 increased throughput by a factor of about 3.7 over CPU-only execution and kept it between 87.70 and 89.33 FPS at every resolution. The desktop GPU was faster, as expected, but the RPi 5 + Edge TPU configuration achieved real-time throughput within the power and size limits of an embedded installation. These observations apply to the RPi 5 and Coral Edge TPU configurations evaluated here and are not claimed as a general property of edge accelerators.

4.3. Resolution-Accuracy-Throughput Trade-off Analysis

Input resolution is one of the few parameters that acts on both sides of an embedded vision budget at once, since it sets the computational cost of every forward pass and also determines how much of a small sign survives to reach the feature extractor. Lowering it is therefore not a free optimization, and the appropriate value has to be argued from measurements of both quantities rather than from either one

alone. The accuracy results of Section 4.1 and the throughput results of Section 4.2 were then examined together to choose an operating point for embedded deployment. Figure 6 plots mAP@50 against Edge TPU throughput for the six input resolutions, which shows the resolution-accuracy-throughput trade-off directly.

Figure 6 shows that a lower input resolution reduces accuracy considerably but brings almost no computational benefit to the target embedded platform. The best result was obtained at 640×640, with 0.812 mAP@50 and 0.690 mAP@50:95 at 87.86 FPS on the RPi 5 + Edge TPU configuration. Reducing the input to 512×512 increased throughput by only 1.47 FPS, from 87.86 to 89.33, while mAP@50 fell by about 14.8%, from 0.812 to 0.692. Each further reduction lost more accuracy, while throughput remained almost the same. Strong downsampling, therefore, gave little measured benefit on the RPi 5 + Edge TPU configuration.

The nearly constant throughput may be explained by fixed costs in accelerator execution, data transfer and host-side processing that do not shrink with the input size. On this platform, reducing the input resolution is not an effective way to gain speed, because accuracy drops sharply while throughput hardly changes. Of the six settings tested, 640×640 is the most suitable operating point: it gives the highest accuracy and still runs at about 88 FPS on the RPi 5 + Edge TPU configuration, which is well above the frame rates usually needed in intelligent transportation and driver-assistance systems. The conclusion holds for YOLOv8n on the tested hardware; with a larger detector or another accelerator, the balance may be different.

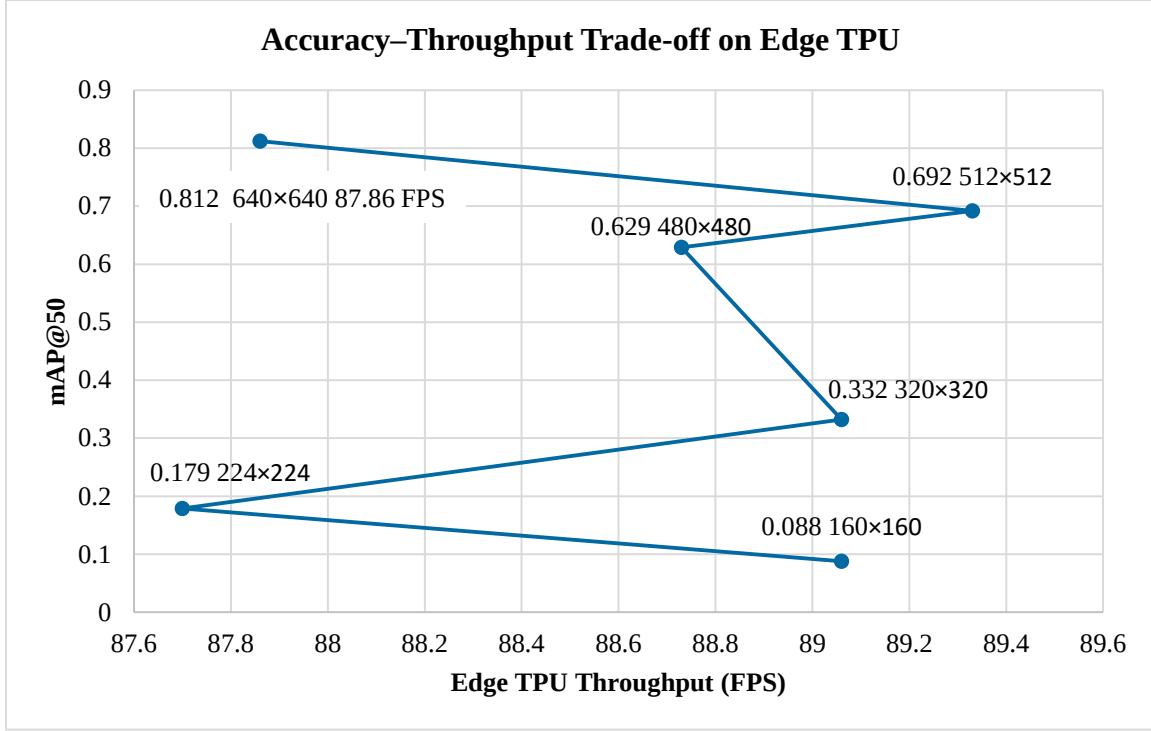


Fig. 6 Resolution-accuracy-throughput trade-off of YOLOv8n on the RPi 5 + Edge TPU configuration

4.4. Occupancy-based Detection Reliability Analysis

The trade-off analysis gives a suitable operating resolution, but it does not explain why accuracy falls so steeply at lower resolutions. An occupancy-based analysis was therefore carried out to measure how normalized traffic-sign occupancy is related to detector reliability.

4.4.1. Occupancy Distribution Analysis

Occupancy was calculated for every annotated sign in the TT100K test subset from its normalized bounding-box size. The distribution is shown in Figure 7, and Table 9 lists the descriptive statistics.

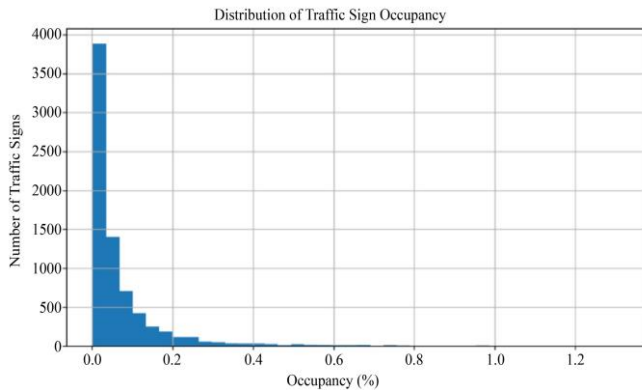


Fig. 7 Occupancy distribution histogram

Occupancy values span 0.0022% to 1.3134%, with a mean of 0.0725% and a median of 0.0326%. Half of all annotated signs, therefore, cover less than about one three-

thousandth of the frame. The histogram is strongly right-skewed: the great majority of objects fall in the lowest occupancy bins, and only a small tail occupies an appreciable share of the image.

Table 9. Descriptive statistics of normalized traffic-sign occupancy in the TT100K test split

Statistic	Value (%)
Number of Traffic Signs	7,471
Mean Occupancy	0.0725
Standard Deviation	0.1150
Minimum Occupancy	0.0022
First Quartile (Q1)	0.0139
Median Occupancy	0.0326
Third Quartile (Q3)	0.0802
Maximum Occupancy	1.3134

The quartile boundaries sharpen the picture. A quarter of the annotated signs cover less than 0.0139% of the image, and three-quarters cover less than 0.0802%. This follows from how the dataset was captured, namely vehicle-perspective imagery in which signs first become visible at a distance, and it mirrors the operating condition that matters most in practice, since a driver-assistance system gains little from recognizing a sign only once the vehicle has almost reached it. It also means that dataset-level averages on TT100K are dominated by small objects, which is precisely why a scale-resolved measure is needed. The next subsection provides one by computing recall separately within each occupancy quartile.

4.4.2. Quartile-based Detection Reliability Analysis

For the quartile analysis, the occupancy values were sorted and divided into four groups (Q1–Q4) of about 1,868 signs each. Recall was calculated separately for each group with the one-to-one pairing rule of Section 3.5; the values are listed in Table 10 and plotted in Figure 8.

Recall rises monotonically with occupancy across the four quartiles, as Table 10 and Figure 8 show. In the smallest quartile, it reaches only 34.31%, meaning that roughly two of every three signs in that group are missed. With occupancy below 0.0139% in this band, such objects survive downsampling as a handful of pixels, leaving the feature extractor very little to work with.

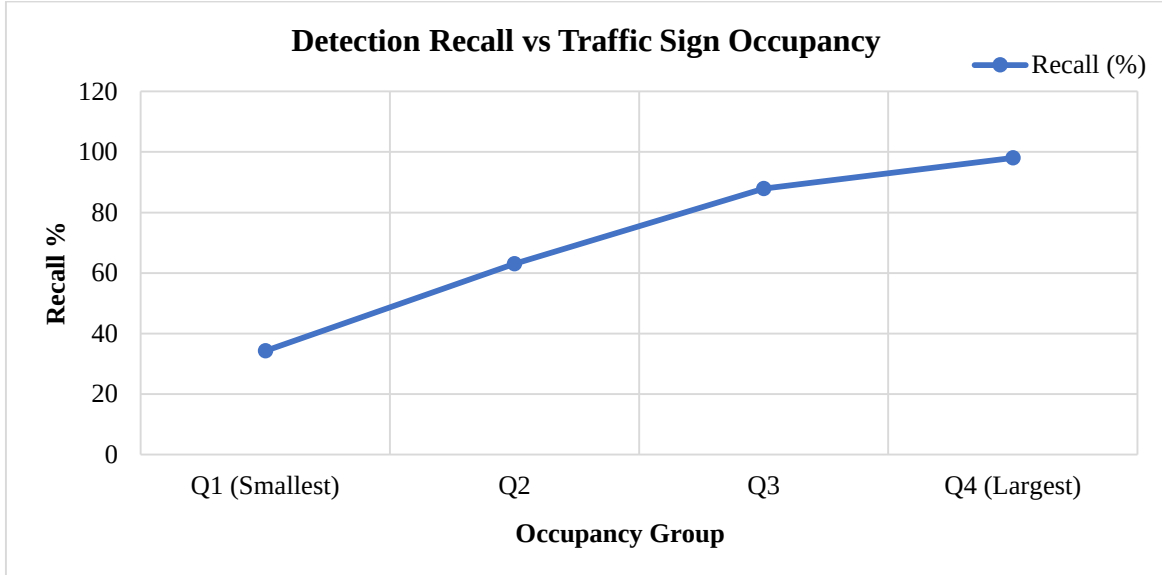


Fig. 8 Detection recall vs quartiles occupancy

Table 10. Recall performance across occupancy quartiles

Occupancy Group	Total Signs	Detected	Missed	Recall (%)
Q1 (Smallest)	1868	641	1227	34.31
Q2	1870	1179	691	63.05
Q3	1865	1640	225	87.94
Q4 (Largest)	1868	1831	37	98.02

Recall rises to 63.05% in Q2, almost 29 percentage points above Q1, and to 87.94% in Q3. A moderate increase in sign size thus produces a large gain in detection. In Q4, recall is 98.02%, which is close to its upper limit; once enough spatial detail is available, the remaining misses are caused mainly by factors other than size.

The steady rise from Q1 to Q4 shows that occupancy is one of the main factors controlling detection reliability. A single dataset-level score averages this effect out, whereas recall grouped by occupancy makes it visible. The quartiles confirm that occupancy matters, but four wide groups cannot show where the detector moves from unreliable to reliable operation. A finer interval analysis is used for this purpose in the next subsection.

4.4.3. Critical Occupancy Threshold Analysis

For the finer analysis, the occupancy range was divided into eight predefined intervals, from very small to

comparatively large signs, and recall was calculated for each interval. The results are listed in Table 11 and plotted in Figure 9.

The interval decomposition resolves the detector's behaviour more finely than the quartiles. Recall was 20.50% below 0.005% occupancy and 27.25% between 0.005% and 0.01%. It then increased to 50.85% in the 0.01%–0.02% interval, representing the largest adjacent increase in Table 11.

Table 11. Detection recall across traffic-sign occupancy ranges

Occupancy Range (%)	Total Signs	Recall (%)
<0.005	161	20.50
0.005–0.01	910	27.25
0.01–0.02	1591	50.85
0.02–0.03	901	67.15
0.03–0.05	1117	81.38
0.05–0.08	905	92.60
0.08–0.10	408	97.55
>0.10	1478	98.17

Recall reached 50.85% in the 0.01%–0.02% interval, 67.15% in the 0.02%–0.03% interval and 81.38% in the 0.03%–0.05% interval, and exceeded 92% in every interval above 0.05%. Above approximately 0.050% occupancy, the detector performs consistently well. Signs covering 0.050% to

0.080% of the image reach 92.60% recall, and larger objects exceed 97%, reaching 98.17% beyond 0.10%. Once sufficient spatial detail is present, further increases in object size yield only marginal gains.

Taken together, these intervals identify a transition region between approximately 0.02% and 0.05% of image area: below 0.02%, recall does not exceed 51%, within the region it rises from 67.15% to 81.38%, and above 0.05% it exceeds 92%. This region is an observation specific to YOLOv8n trained and evaluated on the TT100K test subset with the settings in Table 3 and Table 6; it is not a generally applicable threshold, and different detectors, datasets or training regimes may produce a different relationship.

The results show that detection accuracy in the evaluated configurations depends on input resolution, execution platform and sign occupancy. Among the six tested resolutions, 640×640 produced the highest accuracy while maintaining real-time throughput on the RPi 5 + Edge TPU configuration. The occupancy analysis identifies a transition region between approximately 0.02% and 0.05% for the evaluated dataset and model configuration.

Architecture-centred studies such as YOLO-BS [20] and ES-YOLO [23] report higher accuracy on their respective benchmarks, and the present study does not claim that unmodified YOLOv8n outperforms those models. Its contribution is deployment-side evidence obtained by varying

resolution, platform configuration and object occupancy within one evaluation. On the evaluated RPi 5 + Edge TPU configuration, reducing input resolution produced little measured throughput gain but a substantial accuracy loss. The occupancy analysis further showed that recall was strongly scale-dependent in the selected TT100K test subset. These findings complement, rather than supersede, architecture-focused results.

5. Conclusion

This paper evaluated YOLOv8n for embedded traffic-sign detection across six input resolutions, three computing platforms, and the full range of object scales present in TT100K. Rather than proposing a further architectural modification, it measured how resolution, execution hardware and object occupancy act together, since that combination determines whether a lightweight detector is usable once deployed.

On TT100K, reducing the input resolution lowered validation mAP@50 from 0.812 at 640×640 to 0.088 at 160×160, while the measured throughput of the RPi 5 + Edge TPU configuration stayed between 87.70 and 89.33 FPS. Of the configurations tested, 640×640 therefore gave the highest accuracy while keeping real-time throughput. On the test subset, recall increased from 20.50% below 0.005% occupancy to 98.17% above 0.10%, with a transition region between approximately 0.02% and 0.05% observed for this dataset and model configuration.

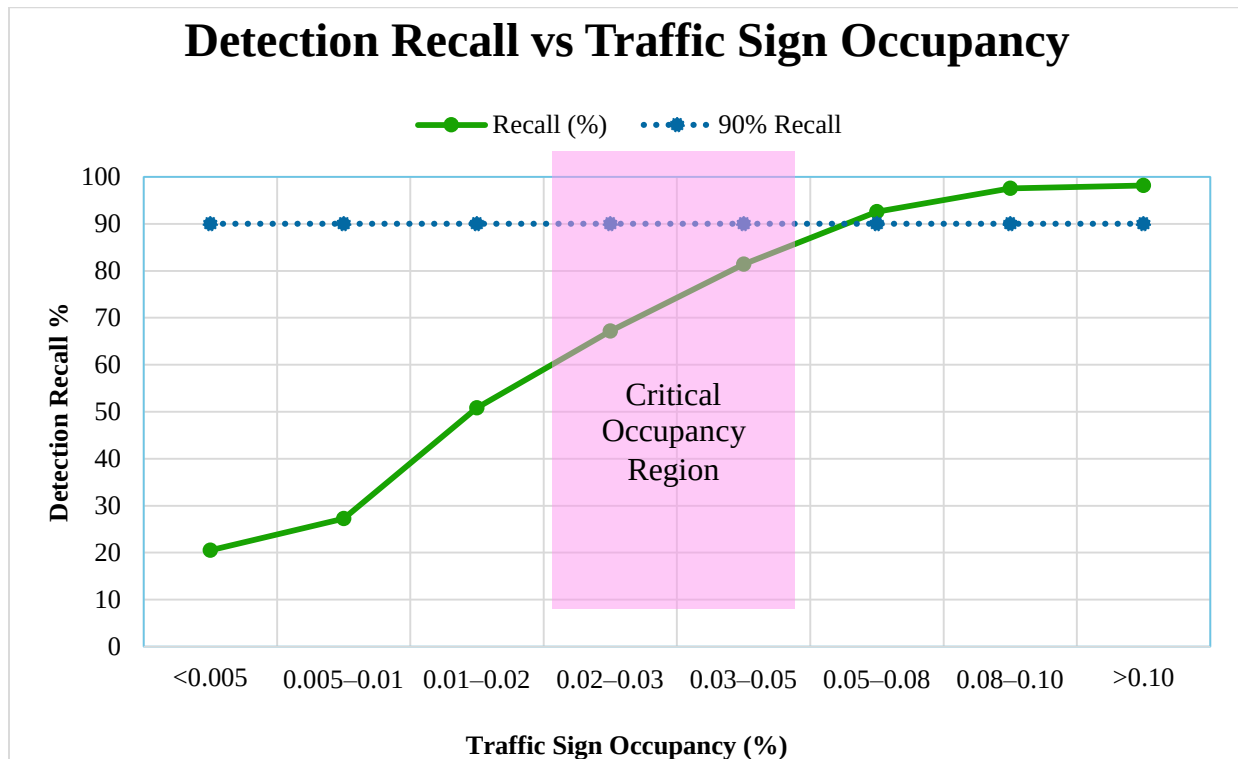


Fig. 9 Detection recall as a function of normalized traffic-sign occupancy

Deploying a lightweight traffic-sign detector is therefore a joint choice of input resolution, accelerator and expected object size, and not only a choice of network. The reported figures are specific to YOLOv8n, to the TT100K benchmark and to the RPi 5 and Coral Edge TPU configurations that were tested. The procedure itself is general and can be repeated with other detectors and boards to obtain the corresponding values.

Future work will apply the occupancy-based procedure to further traffic-sign datasets and to other lightweight architectures, including transformer-based detectors, and will examine adaptive-resolution schemes that raise the input size only when the scene is likely to contain distant signs.

Conflicts of Interest

The authors declare no conflict of interest.

Funding Statement

No external funding was received for this work. Internal support from VIT-AP University under the RGEMS scheme is acknowledged below.

Acknowledgments

The authors thank VIT-AP University for financial support under the RGEMS scheme (Project ID: RGEMS2021010).

References

- [1] Andreas Mogelmose, Mohan Manubhai Trivedi, and Thomas B. Moeslund, "Vision-based Traffic Sign Detection and Analysis for Intelligent Driver Assistance Systems: Perspectives and Survey," *IEEE Transactions on Intelligent Transportation Systems*, vol. 13, no. 4, pp. 1484-1497, 2012. [[CrossRef](#)] [[Google Scholar](#)] [[Publisher Link](#)]
- [2] Ayoub Ellahyani, Ilyas El Jaafari, and Said Charfi, "Traffic Sign Detection for Intelligent Transportation Systems: A Survey," *E3S Web of Conferences*, vol. 229, pp. 1-10, 2021. [[CrossRef](#)] [[Google Scholar](#)] [[Publisher Link](#)]
- [3] Joseph Redmon et al., "You Only Look Once: Unified, Real-Time Object Detection," *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Las Vegas, NV, USA, pp. 779-788, 2016. [[CrossRef](#)] [[Google Scholar](#)] [[Publisher Link](#)]
- [4] Junfan Wang et al., "Vehicle-Mounted Adaptive Traffic Sign Detector for Small-Sized Signs in Multiple Working Conditions," *IEEE Transactions on Intelligent Transportation Systems*, vol. 25, no. 1, pp. 710-724, 2024. [[CrossRef](#)] [[Google Scholar](#)] [[Publisher Link](#)]
- [5] Junzhou Chen et al., "YOLO-TS: Real-Time Traffic Sign Detection with Enhanced Accuracy Using Optimized Receptive Fields and Anchor-Free Fusion," *IEEE Transactions on Intelligent Transportation Systems*, vol. 26, no. 11, pp. 1995-2001, 2025. [[CrossRef](#)] [[Google Scholar](#)] [[Publisher Link](#)]
- [6] Gaihua Wang et al., "Traffic Sign Detection Method based on Improved YOLOv8," *Scientific Reports*, vol. 15, pp. 1-13, 2025. [[CrossRef](#)] [[Google Scholar](#)] [[Publisher Link](#)]
- [7] Bin Ji et al., "Improved YOLOv8 for Small Traffic Sign Detection under Complex Environmental Conditions," *Franklin Open*, vol. 8, pp. 1-9, 2024. [[CrossRef](#)] [[Google Scholar](#)] [[Publisher Link](#)]
- [8] Yuqi Li et al., "A Small-Target Traffic Sign Detection Algorithm based on Partial Conv and Atrous Spatial Pyramid," *IET Image Processing*, vol. 18, no. 14, pp. 4639-4652, 2024. [[CrossRef](#)] [[Google Scholar](#)] [[Publisher Link](#)]
- [9] Yan Zhang et al., "A Lightweight Granular Perception Feature Pyramid Network with Context-Awareness for Small Traffic Sign Detection," *Expert Systems with Applications*, vol. 317, 2026. [[CrossRef](#)] [[Google Scholar](#)] [[Publisher Link](#)]
- [10] Quanquan Tian, and Yang Zhang, "Synergising Heterogeneous Features and Guided Attention for Small Traffic Sign Detection in Complex Scenes," *Expert Systems*, vol. 43, no. 6, 2026. [[CrossRef](#)] [[Google Scholar](#)] [[Publisher Link](#)]
- [11] Vidya Kamath et al., "Exploratory Data Preparation and Model Training Process for Raspberry Pi-Based Object Detection Model Deployments," *IEEE Access*, vol. 12, pp. 45423-45441, 2024. [[CrossRef](#)] [[Google Scholar](#)] [[Publisher Link](#)]
- [12] David Minott, Salman Siddiqui, and Rami J. Haddad, "Benchmarking Edge AI Platforms: Performance Analysis of NVIDIA Jetson and Raspberry Pi 5 with Coral TPU," *SoutheastCon 2025*, Concord, NC, USA, pp. 1384-1389, 2025. [[CrossRef](#)] [[Google Scholar](#)] [[Publisher Link](#)]
- [13] Arief Kurniawan et al., "Performance Comparison of Edge Devices for Deep Learning-based Object Detection and Classification," *2025 International Seminar on Intelligent Technology and Its Applications (ISITIA)*, Surabaya, Indonesia, pp. 373-378, 2025. [[CrossRef](#)] [[Google Scholar](#)] [[Publisher Link](#)]
- [14] Daghash K. Alqahtani, Muhammad Aamir Cheema, and Adel N. Toosi, "Benchmarking Deep Learning Models for Object Detection on Edge Computing Devices," *22nd International Conference Service-Oriented Computing*, Tunis, Tunisia, pp. 142-150, 2024. [[CrossRef](#)] [[Google Scholar](#)] [[Publisher Link](#)]
- [15] Zhe Zhu et al., "Traffic-Sign Detection and Classification in the Wild," *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Las Vegas, NV, USA, pp. 2110-2118, 2016. [[CrossRef](#)] [[Google Scholar](#)] [[Publisher Link](#)]
- [16] Gülcan Yıldız, and Bekir Dizdaroğlu, "Traffic Sign Detection via Color and Shape-Based Approach," *2019 1st International Informatics and Software Engineering Conference (UBMYK)*, Ankara, Turkey, pp. 1-5, 2019. [[CrossRef](#)] [[Google Scholar](#)] [[Publisher Link](#)]
- [17] Radu Timofte, Karel Zimmermann, and Luc Van Gool, "Multi-View Traffic Sign Detection, Recognition, and 3D Localisation," *Machine Vision and Applications*, vol. 25, no. 3, pp. 633-647, 2014. [[CrossRef](#)] [[Google Scholar](#)] [[Publisher Link](#)]

- [18] Aashrith Vennelakanti et al., "Traffic Sign Detection and Recognition using a CNN Ensemble," *2019 IEEE International Conference on Consumer Electronics (ICCE)*, Las Vegas, NV, USA, pp. 1-4, 2019. [[CrossRef](#)] [[Google Scholar](#)] [[Publisher Link](#)]
- [19] Jianming Zhang et al., "A Cascaded R-CNN with Multiscale Attention and Imbalanced Samples for Traffic Sign Detection," *IEEE Access*, vol. 8, pp. 29742-29754, 2020. [[CrossRef](#)] [[Google Scholar](#)] [[Publisher Link](#)]
- [20] Zhang Hong, Liang Mingyin, and Wang Yufeng, "YOLO-BS: A Traffic Sign Detection Algorithm based on YOLOv8," *Scientific Reports*, vol. 15, pp. 1-11, 2025. [[CrossRef](#)] [[Google Scholar](#)] [[Publisher Link](#)]
- [21] Bo Meng, and Weida Shi, "Small Traffic Sign Recognition Method Based on Improved YOLOv7," *Scientific Reports*, vol. 15, no. 1, pp. 1-18, 2025. [[CrossRef](#)] [[Google Scholar](#)] [[Publisher Link](#)]
- [22] Xianzheng Liu et al., "A Feature Cascade and Recursive Fusion Architecture for Traffic Sign Detection in Vehicle Perception," *Scientific Reports*, vol. 15, no. 1, pp. 1-13, 2025. [[CrossRef](#)] [[Google Scholar](#)] [[Publisher Link](#)]
- [23] Weiguo Pan et al., "ES-YOLO: Edge and Shape Fusion-Based YOLO for Traffic Sign Detection," *Computers, Materials and Continua*, vol. 87, no. 1, pp. 1-19, 2026. [[CrossRef](#)] [[Google Scholar](#)] [[Publisher Link](#)]
- [24] Chen Xing, Haoran Sun, and Jiafu Yang, "A Lightweight Traffic Sign Detection Model based on Improved YOLOv8s for Edge Deployment in Autonomous Driving Systems Under Complex Environments," *World Electric Vehicle Journal*, vol. 16, no. 8, pp. 1-18, 2025. [[CrossRef](#)] [[Google Scholar](#)] [[Publisher Link](#)]
- [25] Christine Dewi et al., "Robust Detection Method for Improving Small Traffic Sign Recognition based on Spatial Pyramid Pooling," *Journal of Ambient Intelligence and Humanized Computing*, vol. 14, no. 7, pp. 8135-8152, 2021. [[CrossRef](#)] [[Google Scholar](#)] [[Published Link](#)]
- [26] Chunfang Deng et al., "Extended Feature Pyramid Network for Small Object Detection," *IEEE Transactions on Multimedia*, vol. 24, pp. 1968-1979, 2022. [[CrossRef](#)] [[Google Scholar](#)] [[Publisher Link](#)]
- [27] Omid Nejati Manzari, Amin Boudesh, and Shahriar B. Shokouhi, "Pyramid Transformer for Traffic Sign Detection," *2022 12th International Conference on Computer and Knowledge Engineering (ICCKE)*, Mashhad, Iran, Islamic Republic of, pp. 112-116, 2022. [[CrossRef](#)] [[Google Scholar](#)] [[Publisher Link](#)]
- [28] Ji Qiu et al., "DP-YOLO: A Lightweight Traffic Sign Detection Model for Small Object Detection," *Digital Signal Processing*, vol. 165, pp. 1-15, 2025. [[CrossRef](#)] [[Google Scholar](#)] [[Publisher Link](#)]
- [29] Domen Tabernik, and Danijel Skočaj, "Deep Learning for Large-Scale Traffic-Sign Detection and Recognition," *IEEE Transactions on Intelligent Transportation Systems*, vol. 21, no. 4, pp. 1427-1440, 2019. [[CrossRef](#)] [[Google Scholar](#)] [[Publisher Link](#)]
- [30] Jianfeng Han, Chao Wang, and Jinnan Du, "HMS-YOLO: Multi-Scale Traffic Sign Detection for Complex Scenarios," *Digital Signal Processing*, vol. 183, 2026. [[CrossRef](#)] [[Google Scholar](#)] [[Publisher Link](#)]
- [31] Yili Xu, and Xuanxuan Xiao, "PST-YOLO: A Lightweight, Robust Approach for Traffic Sign Detection via Polynomial-based and Progressive Scale Fusions," *Digital Signal Processing*, vol. 169, pp. 1-13, 2026. [[CrossRef](#)] [[Google Scholar](#)] [[Publisher Link](#)]
- [32] Lisha Cui et al., "Context-Aware Block Net for Small Object Detection," *IEEE Transactions on Cybernetics*, vol. 52, no. 4, pp. 2300-2313, 2022. [[CrossRef](#)] [[Google Scholar](#)] [[Publisher Link](#)]
- [33] Xiyang Dai et al., "Dynamic Head: Unifying Object Detection Heads with Attentions," *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, Nashville, TN, USA, pp. 7369-7378, 2021. [[CrossRef](#)] [[Google Scholar](#)] [[Publisher Link](#)]
- [34] Ricardo C. Camara de M. Santos, Mateus Coelho, and Ricardo Oliveira, "Real-Time Object Detection Performance Analysis using YOLOv7 on Edge Devices," *IEEE Latin America Transactions*, vol. 22, no. 10, pp. 799-805, 2024. [[CrossRef](#)] [[Google Scholar](#)] [[Publisher Link](#)]
- [35] Petros Amanatidis et al., "Cooperative Task Execution for Object Detection in Edge Computing: An Internet of Things Application," *Applied Sciences*, vol. 13, no. 8, pp. 1-19, 2023. [[CrossRef](#)] [[Google Scholar](#)] [[Publisher Link](#)]
- [36] Quang Nhat Nguyen Le, Mamen Thomas Chembakasseril, and Ronny Hartanto, "Benchmark Analysis of Deep Learning Algorithms for Edge-Based Robotic Applications," *Array*, vol. 30, pp. 1-12, 2026. [[CrossRef](#)] [[Google Scholar](#)] [[Publisher Link](#)]