TELKOMNIKA Telecommunication Computing Electronics and Control
Vol. 24, No. 1, February 2026, pp. 126~141
ISSN: 1693-6930, DOI: 10.12928/TELKOMNIKA.v24i1.27448

Performance enhancement of embedded object detection via neural hardware acceleration

Alwin Hartono Limaran, Agung Wicaksono, Patah Herwanto
Department of Informatics Engineering, Faculty of Information Technology, STMIK Indonesia Mandiri, Bandung, Indonesia

Article Info

Article history:

Received Aug 4, 2025
Revised Oct 24 2025
Accepted Dec 8, 2025

Keywords:

Energy efficiency
Low-latency image inference
Orange Pi 3B
RK3566 NPU
YOLO11

ABSTRACT

This paper presents the first benchmarking of you only look once version 11 (YOLO11) on the Rockchip RK3566 neural processing unit (NPU) within the Orange Pi 3B platform. Performance was compared between the quad-core ARM Cortex-A55 CPU and the integrated NPU using the COCO2017 dataset, evaluating latency, energy, and accuracy. NPU acceleration achieved >80% latency reduction and ≈ 94% lower per-inference energy consumption, with speedup of up to 16.7× while maintaining accuracy within 0.03 mean average precision (mAP) of the baseline. Average power remained nearly constant (3.60 W central processing unit (CPU) vs. 3.59 W NPU), indicating that the efficiency gains stem from reduced inference time rather than lower wattage. Limitations included unstable INT8 quantization due to unsupported operators and calibration-range mismatch, as well as minor CPU-side overhead in preprocessing and non-maximum suppression. The findings confirm that the RK3566 NPU delivers substantial efficiency gains without accuracy loss, enabling compact and low-cost platforms to sustain modern object-detection workloads. This demonstrates that affordable NPUs can provide reliable, real-time artificial intelligence (AI) inference for embedded vision, internet of things (IoT), and robotics applications.

Corresponding Author:

Alwin Hartono Limaran
Department of Informatics Engineering, Faculty of Information Technology
STMIK Indonesia Mandiri
Belitung Street No. 7, Merdeka, Sumur Bandung, Bandung, West Java, Indonesia
Email: alwin.hartono@protonmail.com

1. INTRODUCTION

Object detection has become a central component of modern computer vision, powering applications in surveillance, robotics, autonomous vehicles, and the internet of things (IoT) [1]. Among the available approaches, the you only look once (YOLO) algorithm [2] is widely used because it balances accuracy with computational efficiency. Unlike region proposal-based methods that operate in multiple stages, YOLO processes an entire image in a single pass [3], [4], simultaneously predicting bounding boxes, confidence scores, and class labels. This design enables low-latency inference, even on resource-limited platforms.

YOLO11 extends this line of development with several improvements. It introduces faster inference optimized for low-power devices, integrates global scene awareness for more precise bounding box predictions [5], [6], and offers built-in support for embedded acceleration on neural processing units (NPUs) [7], [8]. These features position YOLO11 as a strong candidate for deployment in edge environments where latency and energy efficiency are critical. The present study restricted its scope to low-latency, per-image inference on static images and isolated compute-backend effects (central processing unit (CPU) vs. RK3566 NPU) on per-image latency and energy at matched accuracy; sequential or temporal processing was excluded.

Previous studies have surveyed YOLO versions and demonstrated the potential of earlier models on affordable platforms such as the Raspberry Pi [9], [10] and NVIDIA Jetson Nano [11]. Ali and Zhang [12] provided a comparative review of YOLO versions 1 through 11, whereas Zagitov *et al.* [13] benchmarked YOLOv5–v8 on embedded boards, showing the importance of hardware acceleration and quantization for sustaining inference performance. However, to the best of our knowledge, YOLO11 has not yet been systematically evaluated on the Rockchip RK3566 NPU within a cost-effective single-board computer such as the Orange Pi 3B.

This is the first study to benchmark YOLO11 on the Orange Pi 3B (RK3566 NPU) and to compare CPU-based and NPU-based executions under representative real-world scenarios. We evaluated end-to-end latency, including preprocessing, inference, and postprocessing, along with per-inference energy and model accuracy. NPU acceleration reduced inference latency by 82–94% and per-inference energy by 80–94%, while accuracy remained within 0.026–0.030 mean average precision (mAP) of the Ultralytics baseline [14]. These findings confirm that compact, affordable platforms can sustain modern object detection workloads without substantial accuracy loss, supporting cost-effective deployment in surveillance, robotics, and IoT applications [15].

2. METHOD

YOLO11 incorporates several new modules, including cross stage partial with kernel size 2 (C3K2), spatial pyramid pooling-fast (SPPF) [16], [17], and the convolutional block with parallel spatial attention (C2PSA). These components streamline computation and accelerate detection without requiring high-end processors. Building on these improvements, the present study examines YOLO11's performance when paired with an Orange Pi 3B Figure 1 and the Rockchip RK3566 NPU, with particular emphasis on speed and energy consumption. The study also introduces a dedicated YOLO11 port for the RK3566 platform, demonstrating that compact, affordable hardware can still deliver practical, low-latency image inference performance at the edge.

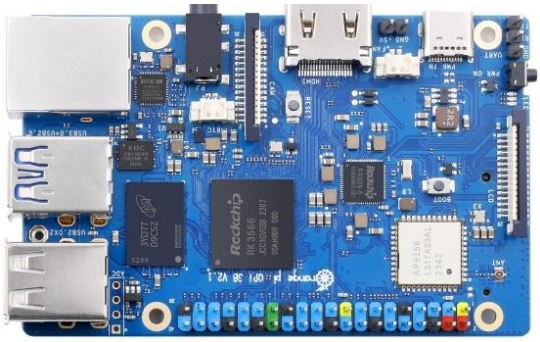

Figure 1. Orange Pi 3B with Rockchip NPU RK3566

As shown in Figure 2, these new modules were integrated into the YOLO11 pipeline to refine feature extraction, improve sensitivity across object scales, and enhance spatial attention. The C3K2 block replaces heavier layers with lightweight 3×3 convolutions, enabling the model to maintain detection accuracy while reducing computational load and accelerating the overall process.

The SPPF module streamlines the pooling stage to capture information from multiple scales more efficiently, thereby improving the detection of both small and large objects. In parallel, the C2PSA block applies dual spatial attention paths that highlight the most informative regions, enabling the network to refine object boundaries and improve localization accuracy.

In addition to these modules, YOLO11 updates its backbone with lighter blocks, such as C3K2 and a cross-stage partial block with a 3×3 kernel (C3K). Compared with the earlier coarse-to-fine (C2F) design, these structures reduce redundant operations while preserving feature richness, giving the network a leaner path for training and inference. To emphasize this refinement, Figure 3 directly compares the C2F block with the newer C3K2/C3K structures, illustrating how efficiency is improved without sacrificing representational power.

An NPU is a specialized accelerator for deep-learning workloads [18]-[20], designed to perform large matrix operations more efficiently than CPUs or graphics processing units (GPUs). Its efficiency derives from parallel dataflow architectures, on-chip memory, and low-power circuitry, making it suitable for energy-constrained edge devices such as smartphones and single-board computers.

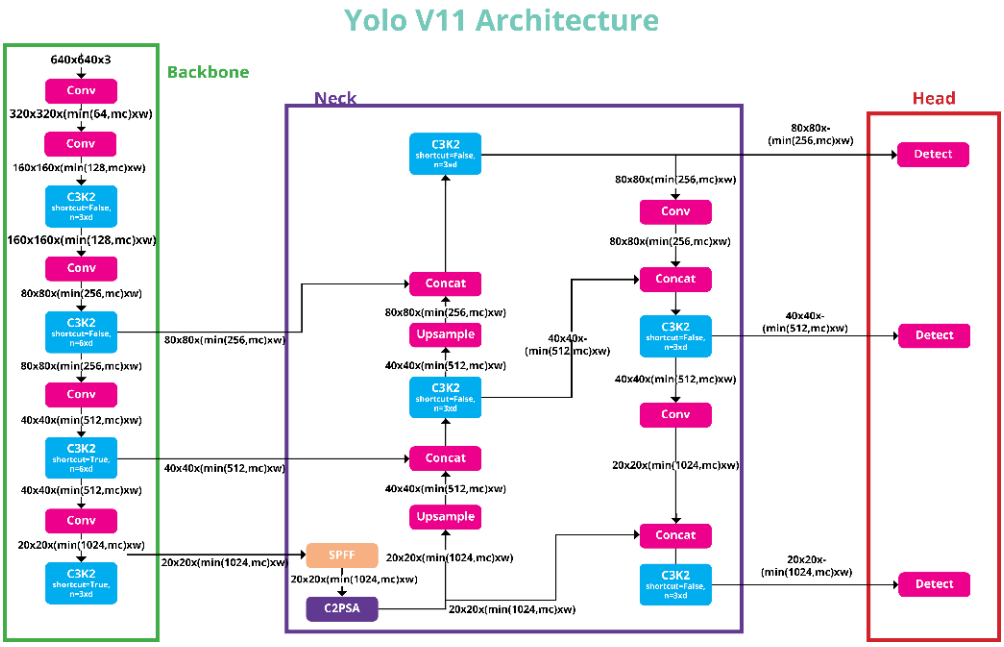

Figure 2. YOLO11 architecture

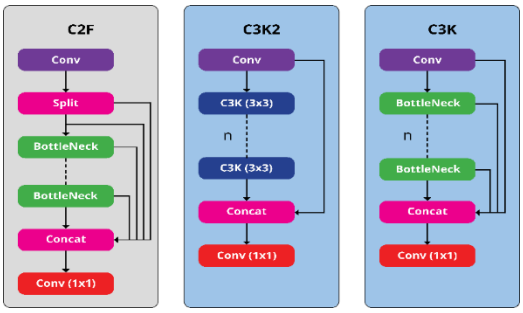

Figure 3. Comparison of C2F and C3K2 blocks

This study evaluated YOLO11 performance on the Orange Pi 3B by comparing CPU processing with NPU acceleration from the RK3566 chip. The analysis considered three stages of detection. In preprocessing, raw images were resized, normalized, and reformatted to meet input requirements. Inference then executed the trained YOLO11 network, scanned each image, extracted features, and predicted bounding boxes with class labels, a process dominated by large matrix multiplications constituting most of the computational load [21]. Finally, postprocessing applied non-maximum suppression (NMS) to filter overlaps and generate final object classes and confidence scores.

2.1. Hardware platform

The Orange Pi 3B single-board computer is built on the Rockchip RK3566 system-on-chip (SoC) [22], a low-power ARM-based platform designed for embedded and artificial intelligence of things (AIoT) applications. The SoC integrates a quad-core ARM Cortex-A55 CPU [23], a Mali-G52 GPU, and an RK3566 NPU officially rated at 0.8–1 tera operations per second (TOPS) [24], [25], depending on arithmetic precision. In practice, this corresponds to approximately 0.8 TOPS for 16-bit integer (INT16) precision [26] and 1 TOPS for 8-bit integer (INT8) precision [27]. However, such peak ratings are theoretical and do not always reflect real performance, particularly under the floating-point (FP32) [28] workloads used in this study. The board includes LPDDR4 memory and a 256 GB non-volatile memory express (NVMe) M.2 solid-state drive (SSD) as primary storage. These specifications offer sufficient bandwidth and capacity for modern object detection models while maintaining a compact design and low power consumption, making the platform suitable for edge artificial intelligence (AI) experiments.

2.2. Software environment

All experiments were conducted on Orange Pi OS 1.0.8 (Ubuntu Jammy) with the Linux 5.10.160-Rockchip-RK356X kernel, running on a 64-bit ARM architecture. The workflow used Python 3.10.12, with PyTorch 2.8.0 and TorchVision 0.23.0 as the primary deep-learning frameworks. YOLO11 was deployed using the Ultralytics library [29] (version 8.3.202). The pycocotools package (version 2.0.10) ensured compliance with the COCO evaluation protocol. Supporting libraries included NumPy 2.2.6, SciPy 1.15.3, Pillow 11.3.0, and OpenCV 4.12.0 for preprocessing and numerical computation. Deployment on the RK3566 NPU used Rockchip's rockchip neural network (RKNN)-Toolkit Lite2 (version 1.6.0), installed from the official aarch64 wheel, supporting pretrained model import from PyTorch, TensorFlow, and open neural network exchange (ONNX) format.

2.3. Model conversion workflow

YOLO11 models were obtained as official pretrained weights from Ultralytics, trained on the COCO dataset. Conversion used the Ultralytics export utility with the command "yolo export model=model_name.pt format=rknn name=rk3566," which first generated an ONNX model before producing the RKNN format. The command exported the PyTorch model to ONNX and then converted it to RKNN using RKNN-Toolkit Lite2. This intermediate step was automated within the Ultralytics library, enabling execution as a single command.

All models were maintained at FP32 precision without pruning or quantization to preserve baseline accuracy. Detection outputs were refined using class-agnostic NMS at an IoU threshold of 0.45. The experiments were run with a batch size of 1, and the maximum detections (max_det) were limited by the 8 GB system RAM.

Although the RK3566 NPU supports INT8 and INT16 inference, several INT8 conversion trials on COCO Val2017 (500–5000 images) produced invalid results due to quantization noise. Pruning was also excluded to avoid confounding effects. FP32 precision was therefore maintained throughout to ensure that the measured CPU–NPU differences reflected the hardware efficiency alone.

2.4. Dataset

The Microsoft COCO 2017 dataset was used to evaluate detection accuracy. It defines 80 object categories and provides 118,287 training images (train2017), 5,000 validation images (val2017), and 40,670 unlabeled test images divided into test-dev and test-challenge [30], [31]. The validation set was used for mAP evaluation, consistent with the Ultralytics YOLO11 benchmark. Accuracy was computed using pycocotools and the COCO protocol at a confidence threshold of 0.001. For Orange Pi 3B inference, a 0.25 threshold filtered low-confidence predictions. Latency and energy benchmarking were conducted on five random Internet images representing diverse visual conditions. This ensured accuracy aligned with the official benchmark, whereas hardware performance was assessed independently of large-scale dataset processing.

2.5. Processing pipeline

The workflow was measured in three stages Figure 4: preprocessing (resize, normalization, and tensor reformatting), inference, and postprocessing (NMS). In the CPU baseline, all stages ran on the CPU. In the NPU configuration, inference ran on the RK3566 NPU (device), whereas preprocessing and postprocessing ran on the CPU (host); host→NPU and NPU→host tensor transfers were included in the latency and energy totals. Stage times were summed for end-to-end latency, and power draw was logged continuously with a digital DC ammeter to compute energy per inference. Static, single-image inputs were used in the evaluations to isolate the CPU vs. NPU effects at matched accuracy.

2.6. Evaluation metrics

YOLO11 on the Orange Pi 3B was evaluated using three indicators: processing time, power consumption, and detection accuracy. Processing time was obtained from millisecond-level timestamps for preprocessing, inference, and postprocessing, with the total as their sum. Power consumption was measured inline on the 5 V USB-C supply using a HiDance type-C power meter (DC 4.5–50 V, 0–6 A continuous). Voltage and current were logged in real time, and instantaneous power was calculated as their product. Per-image mean power was derived by averaging readings across each inference interval. Detection accuracy was reported as mAP@[0.5:0.95] under the COCO protocol, with mAP at a 0.001 confidence threshold (as in Ultralytics YOLO) and deployment inference at 0.25.

Additional indicators included throughput (images per second at batch size 1), energy per image (average power × inference duration), performance per watt, and CPU–NPU speedup and energy-reduction ratios. The results are reported as per-image averages, with distributions provided where relevant.

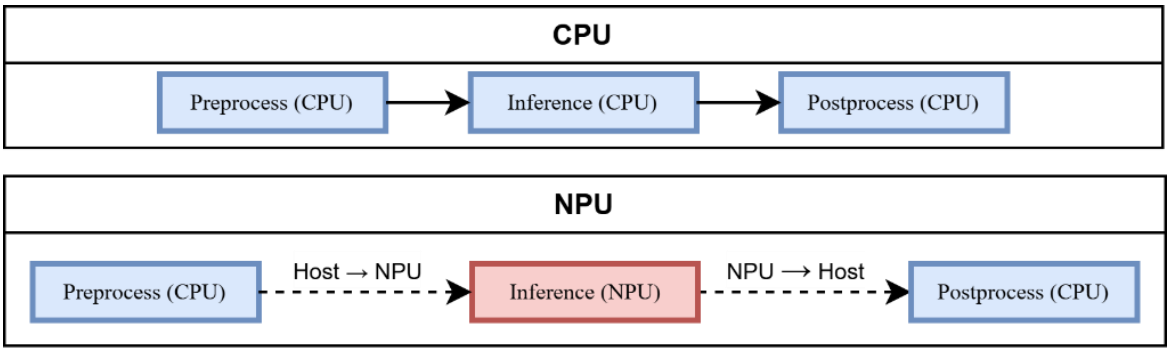

Figure 4. CPU vs. NPU processing pipeline

2.7. Experimental scenarios

Five practical scenarios were developed to evaluate how YOLO11 adapts to different environmental conditions on the Orange Pi 3B. The representative frames shown in Figures 5(a)–(e) capture variations, including diverse lighting conditions, motion levels, and object sizes.

These scenarios were specifically designed to probe YOLO11's performance under various real-world conditions.

- Classroom (Image 1; Figure 5(a) - quiet classroom with eight standing subjects under uniform lighting; it evaluates the stability of multi-person detection on a single frame.
- Urban street crossing (Image 2; Figure 5(b) - busy crosswalk with pedestrians and cyclists; it tests detection under crowding and occlusion.
- Warehouse operations (Image 3; Figure 5(c) - loading-dock CCTV with aisles and clutter; it assesses robustness against occlusion and visually similar classes.
- Parking lot monitoring (Image 4; Figure 5(d) - mid-afternoon lot with vehicles and pedestrians; strong shadows and high contrast challenge the separation of adjacent objects.
- Nighttime street scene (Image 5; Figure 5(e) - dim urban street with lamps and headlights; it probes low light, glare, and noise effects on detection.

(a) (b) (c) (d) (e)

Figure 5. Benchmark input scenes used for testing YOLO11 performance across diverse environments: (a) classroom setting, (b) urban street crossing, (c) warehouse operations, (d) parking lot monitoring, and (e) nighttime street scene

These scenarios provided complementary benchmarks for comparing CPU-only and NPU-accelerated inference. The measurements included preprocessing, inference, postprocessing, and total per-image times. Five scenarios captured real-world conditions: (i) lighting from bright outdoor to low-light nighttime, (ii) motion from static indoor to crowded streets, and (iii) object sizes and densities, including pedestrians, vehicles, and warehouse goods. Source images also varied in file size (hundreds of kilobytes to several megabytes) and resolution (72–300 DPI), enhancing input diversity.

Together, these selections ensured that the evaluation reflected realistic, challenging deployment without additional datasets. The methodology also outlined the Orange Pi 3B hardware, COCO 2017 dataset with local validation subsets, model conversion, parameter settings, and performance metrics. Accuracy and efficiency were reported through mAP scores, processing times, and power measurements for CPU and NPU runs. With the procedures established, the next section presents the experimental results and discusses YOLO11 performance under embedded edge hardware constraints.

3. RESULTS AND DISCUSSION

This section compares YOLO11 on the CPU and on the RK3566 NPU under matched-accuracy, static-image conditions. We evaluate per-image latency—decomposed into preprocessing, inference, and postprocessing and energy, and quantify efficiency as percentage change. Table 1 consolidates the results by juxtaposing CPU Table 1(a) and NPU Table 1(b) wall-clock times and reporting acceleration as percentage reduction.

Table 1. Visual comparison of YOLO11 detection performance across two platforms

(a) CPU-based inference using PyTorch

Model	Picture	Preprocess (ms)	Inference (ms)	Postprocess (ms)	Total time (ms)	Detection result
YOLO11n	Image 1	42.10	1,597.20	19.80	1,659.10	8 persons
YOLO11n	Image 2	44.30	1,622.50	10.60	1,677.40	12 persons, 3 cars, and 3 traffic lights
YOLO11n	Image 3	28.40	1,428.10	9.80	1,466.30	4 persons and 2 trucks
YOLO11n	Image 4	27.20	1,431.80	9.80	1,468.80	19 cars and 7 trucks
YOLO11n	Image 5	18.40	1,635.20	10.50	1,664.10	6 persons and 4 cars
YOLO11s	Image 1	36.40	5,122.20	10.20	5,168.80	8 persons
YOLO11s	Image 2	41.30	5,110.20	10.60	5,162.10	13 persons, 2 cars, 2 traffic lights, 3 handbags, 1 tie, and 1 cup
YOLO11s	Image 3	25.90	4,484.80	9.90	4,520.60	8 persons and 1 truck
YOLO11s	Image 4	25.00	4,445.00	10.30	4,480.30	1 person, 30 cars, and 5 trucks
YOLO11s	Image 5	16.00	5,130.80	10.20	5,157.00	7 persons, 2 cars, and 1 handbag
YOLO11m	Image 1	36.20	16,952.70	10.60	16,999.50	8 persons and 1 tie
YOLO11m	Image 2	40.80	16,868.20	10.90	16,919.90	15 persons, 3 cars, 3 traffic lights, and 5 handbags
YOLO11m	Image 3	25.00	14,625.40	9.80	14,660.20	5 persons
YOLO11m	Image 4	25.30	14,771.40	10.80	14,807.50	1 person, 26 cars, and 5 trucks
YOLO11m	Image 5	16.70	16,885.90	10.80	16,913.40	7 persons, 2 cars, 1 backpack, and 1 baseball glove
YOLO11l	Image 1	35.50	21,201.70	10.30	21,247.50	8 persons and 1 tie
YOLO11l	Image 2	39.20	21,326.40	11.10	21,376.70	16 persons, 3 cars, 3 traffic lights, and 5 handbags
YOLO11l	Image 3	25.50	18,500.10	9.90	18,535.50	6 persons and 1 backpack
YOLO11l	Image 4	25.60	18,406.10	10.40	18,442.10	1 person, 25 cars, 6 trucks, and 2 traffic lights
YOLO11l	Image 5	17.00	21,257.10	10.40	21,284.50	7 persons and 2 cars
YOLO11x	Image 1	35.80	42,672.20	10.30	42,718.30	8 persons and 1 tie
YOLO11x	Image 2	36.60	42,701.10	10.80	42,748.50	14 persons, 3 cars, 1 traffic light. 4 handbags, and 1 cup
YOLO11x	Image 3	25.60	36,741.20	9.70	36,776.50	6 persons, 1 truck, and 1 dining table
YOLO11x	Image 4	24.90	36,757.60	10.00	36,792.50	1 person, 21 cars, 6 trucks, and 2 traffic lights
YOLO11x	Image 5	16.50	43,144.60	10.30	43,171.40	7 persons, 2 cars, 1 airplane, and 1 tie

(b) RK3566 NPU-based inference using RKNN

Model	Picture	Preprocess (ms)	Inference (ms)	Postprocess (ms)	Total time (ms)	Detection result
YOLO11n	Image 1	35.90	266.10	12.90	314.90	8 persons
YOLO11n	Image 2	49.00	270.10	13.10	332.20	12 persons, 3 cars, and 4 traffic lights
YOLO11n	Image 3	33.90	273.50	14.50	321.90	6 persons and 2 trucks
YOLO11n	Image 4	33.90	271.70	13.40	319.00	22 cars and 7 trucks
YOLO11n	Image 5	21.20	280.10	14.40	315.70	7 persons and 4 cars
YOLO11n	Image 1	42.50	429.40	13.00	484.90	8 persons
YOLO11n	Image 2	48.50	434.30	13.40	496.20	12 persons, 2 cars, 1 traffic light, 3 handbags, 1 tie, and 1 cup
YOLO11n	Image 3	33.60	437.40	13.20	484.20	8 persons and 2 trucks
YOLO11n	Image 4	34.10	435.90	13.90	483.90	1 person, 29 cars, and 4 trucks
YOLO11n	Image 5	21.40	444.10	13.10	478.60	7 persons, 2 cars, and 1 handbag
YOLO11m	Image 1	37.40	946.60	13.00	997.00	8 persons and 1 tie
YOLO11m	Image 2	49.00	952.20	13.90	1,015.10	17 persons, 3 cars, 4 traffic lights, and 5 handbags
YOLO11m	Image 3	33.40	955.40	13.00	1,001.80	5 persons
YOLO11m	Image 4	34.10	956.10	14.10	1,004.30	1 person, 29 cars, and 5 trucks
YOLO11m	Image 5	20.70	959.90	13.00	993.60	7 persons, 2 cars, 1 backpack, and 1 baseball glove
YOLO11l	Image 1	35.80	1,153.10	13.00	1,201.90	8 persons and 1 tie
YOLO11l	Image 2	49.00	1,158.50	13.70	1,221.20	17 persons, 4 cars, 3 traffic lights, and 5 handbags
YOLO11l	Image 3	33.60	1,160.50	14.10	1,208.20	7 persons
YOLO11l	Image 4	33.90	1,160.40	13.70	1,208.00	1 person, 29 cars, 7 trucks, and 2 traffic lights
YOLO11l	Image 5	21.10	1,168.80	13.20	1,203.10	7 persons and 2 cars
YOLO11x	Image 1	42.50	2,525.10	13.00	2,580.60	8 persons and 1tie
YOLO11x	Image 2	49.10	2,642.00	13.50	2,704.60	14 persons, 3 cars, 1 traffic light, and 5 handbags
YOLO11x	Image 3	33.80	2,646.80	13.20	2,693.80	6 persons, 1 truck, and 1 dining table
YOLO11x	Image 4	33.70	2,644.20	13.50	2,691.40	1 person, 22 cars, 6 trucks, and 2 traffic lights
YOLO11x	Image 5	21.40	2,612.40	13.00	2,646.80	7 persons and 2 cars

After recording preprocessing, inference, postprocessing, and total elapsed time for each image, we computed the mean for each YOLO11 variant. The results are reported for both CPU and NPU executions. The data were organized by model type and processing path, providing an interpretable view of workload translation from general-purpose to hardware-accelerated environments.

The arithmetic mean for each pipeline stage was calculated using the standard definition, expressed (1)-(4):

$$Avg.Preprocess = \frac{\sum(Preprocess)}{Number\ of\ Data\ Preprocess} \quad (1)$$

$$Avg.Inference = \frac{\sum(Inference)}{Number\ of\ Data\ Inference} \quad (2)$$

$$Avg.Postprocess = \frac{\sum(Postprocess)}{Number\ of\ Data\ Postprocess} \quad (3)$$

$$Avg.Total\ Time = \frac{\sum(Total\ Time)}{Number\ of\ Data\ Total\ Time} \quad (4)$$

The evaluation results of YOLO11 on CPU and NPU are presented in Table 2 and Figures 6(a)–(d), which compare execution times for preprocessing [32], inference [33], [34], and postprocessing [35]. The NPU consistently outperformed the CPU, particularly during inference, where hardware acceleration played the dominant role. For YOLO11n, inference time decreased from 1,542.96 ms on the CPU to 272.30 ms on the NPU, representing an 82.4% reduction. Regarding larger models, for YOLO11l it dropped from 20,138.28 to 1,160.26 ms (-94.2%), whereas for YOLO11x, the largest variant, it decreased from 40,403.34 to 2,614.10 ms (-93.5%). End-to-end total time also fell sharply: for YOLO11s, it decreased from 4,897.76 to 485.56 ms (-90.1%), and for YOLO11m, it dropped from 16,060.10 to 1,002.36 ms (-93.8%). Latency differences in preprocessing and postprocessing were relatively small, indicating that interface overhead and optimization routines had minimal impact on these phases. Inference thus remained the dominant component of overall latency and benefited most from NPU acceleration. Figure 6(d) provides a heatmap of execution time patterns across all variants (n, s, m, l, and x), where darker shades correspond to shorter durations, particularly in NPU runs. NPUs enable acceleration of up to 724× compared with software-only microcontroller implementations, as reported by Manor and Greenberg [36]; our results corroborate this, demonstrating substantial inference latency reduction at the edge. Energy consumption was measured in real time with a digital analyzer that logged instantaneous voltage, current, and power during model execution on the Orange Pi 3B (RK3566), ensuring accurate calculations.

Table 2. Average time calculation results

Processing stage	YOLO11n		YOLO11s		YOLO11m		YOLO11l		YOLO11x	
	CPU	NPU	CPU	NPU	CPU	NPU	CPU	NPU	CPU	NPU
Preprocess (ms)	32.08	34.78	28.92	36.02	28.80	34.92	28.56	34.68	27.88	36.10
Inference (ms)	1,542.96	272.30	4,858.60	436.22	16,020.72	954.04	20,138.28	1,160.26	40,403.34	2,614.10
Postprocess (ms)	12.10	13.66	10.24	13.32	10.58	13.40	10.42	13.54	10.22	13.24
Total time (ms)	1,587.14	320.74	4,897.76	485.56	16,060.10	1,002.36	20,177.26	1,208.48	40,441.44	2,663.44

Elapsed time was measured within the execution script, with each condition repeated several times to confirm repeatability and accuracy. The results are summarized in comparative tables contrasting CPU and NPU executions, from which mean values were extracted, highlighting the speed gains from NPU acceleration and reductions in total processing time.

Although NPU acceleration drastically reduced inference time, preprocessing [37] and postprocessing latencies increased modestly (approximately 8–30%) compared with those in CPU-only execution. Postprocessing, dominated by CPU-executed NMS with quadratic complexity, remained small in absolute terms relative to the inference speedup.

The increase arises because only inference is offloaded to the RK3566 NPU, whereas preprocessing (resize, normalization, reformatting) [37] and postprocessing (non-maximum suppression, bounding-box adjustment) [38]-[40] remain on the CPU. Host-device tensor transfers (CPU↔NPU) including input upload and output download introduce interface latency, memory copies, and format conversions that account for most of the residual overhead (cost).

Despite these trade-offs, detection quality is preserved. Table 3 compares mAP@0.5:0.95 between the official benchmark and the Orange Pi 3B results. The 500-image subset shows larger fluctuations (Δ ≈ -0.010 to -0.074), whereas the full 5,000-image evaluation is consistent, with deviations of approximately -0.026 to -0.030 across YOLO11 variants. These findings confirm that RKNN conversion preserved baseline accuracy within a narrow margin of the Ultralytics benchmark while delivering substantial efficiency improvements [41].

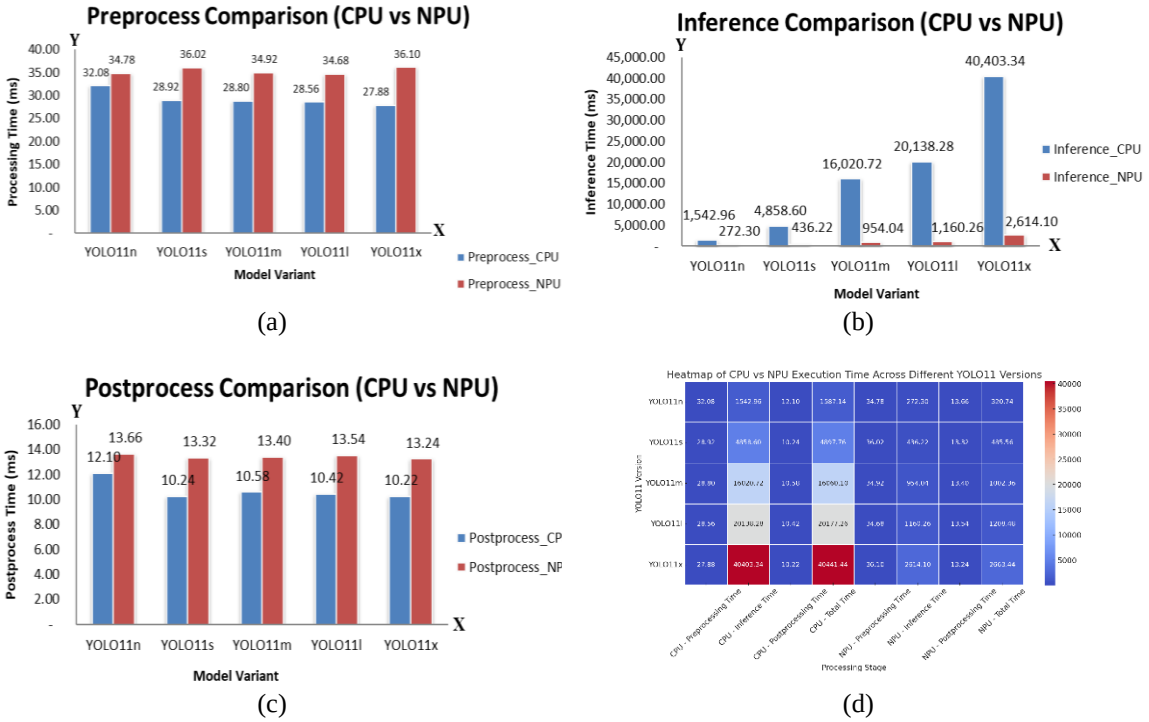

Figure 6. CPU vs. NPU performance comparison: (a) preprocessing, (b) inference, (c) postprocessing, and (d) execution time heatmap across YOLO11 variants. X-axis = model variant (YOLO11n–YOLO11x); Y-axis = execution time (ms)

Table 3. Comparison of YOLO11 mAP results (official Ultralytics vs. Orange Pi 3B, COCO2017 val2017)

Model	Official mAP50–95	On-device mAP (500)	On-device mAP (5,000)	Δ on-device -500	Δ on-device -5,000
YOLO11n	0.395	0.361	0.369	–0.034	–0.026
YOLO11s	0.470	0.427	0.441	–0.043	–0.029
YOLO11m	0.515	0.505	0.487	–0.010	–0.028
YOLO11l	0.534	0.488	0.504	–0.046	–0.030
YOLO11x	0.547	0.473	0.517	–0.074	–0.030

Note: on-device mAP refers to the accuracy measured directly on the Orange Pi 3B platform, whereas the official mAP denotes the benchmark reported by Ultralytics on the COCO2017 validation set.

After confirming consistent detection accuracy across YOLO11 variants with minimal mAP deviations (-0.026 to -0.030), we examined energy efficiency. Table 4 presents nearly identical average power consumption for CPU (3.60 W) and NPU (3.59 W), differing by less than 0.3%. However, the much lower inference latency of the NPU (82–94% faster) sharply reduced energy per inference: YOLO11x consumed 145.59 J on the CPU versus 9.56 J on the NPU, a 93.4% reduction. These results confirm the NPU as the most energy-efficient processor for YOLO11 workloads on the Orange Pi 3B.

NPUs allow edge devices such as the Orange Pi 3B to perform substantially more inference passes within the same power budget, an advantage for platforms powered by batteries or other low-wattage sources. The benefit comes not only from throughput but also from sharply reduced latency at nearly constant average power. This enhances the feasibility of energy-intensive AI workloads and supports near-real-time deep learning in low-power networks.

Verification confirmed that acceleration on the RK3566 NPU did not degrade detection quality [42]. Evaluations used the COCO2017 validation set (val2017). The official YOLO11 results were based on the full

5,000-image set, whereas Orange Pi 3B tests employed both a random 500-image subset and the full set of 5,000 images to ensure consistency.

Tables 4(a) and (b) detail the recorded voltage, current, and power during detection. Table 5 then consolidates latency, speedup, per-inference energy, and accuracy deviation across YOLO11 variants. The results show that average power consumption was nearly identical between CPU and NPU executions. However, the NPU consistently achieved 5–17× faster inference and ~80–94% lower energy per inference [43], [44], while maintaining accuracy within -0.03 mAP of the official baseline. Energy per inference was calculated as average power multiplied by total latency.

Table 4. Recorded voltage, current, and power consumption during the YOLO11 execution on the two platforms

(a) CPU-based

Model	Picture	Average per picture			Average per model		
		Voltage	Current	Power (W)	Voltage	Current	Power (W)
YOLO11n	Image 1	5.35	0.68	3.69	5.35	0.66	3.58
	Image 2	5.35	0.65	3.53			
	Image 3	5.36	0.65	3.51			
	Image 4	5.35	0.66	3.54			
	Image 5	5.35	0.67	3.65			
YOLO11s	Image 1	5.34	0.71	3.81	5.35	0.69	3.72
	Image 2	5.35	0.70	3.75			
	Image 3	5.36	0.70	3.76			
	Image 4	5.35	0.66	3.58			
	Image 5	5.35	0.68	3.69			
YOLO11m	Image 1	5.36	0.66	3.59	5.35	0.68	3.67
	Image 2	5.34	0.69	3.72			
	Image 3	5.35	0.68	3.68			
	Image 4	5.35	0.67	3.64			
	Image 5	5.35	0.69	3.71			
YOLO11l	Image 1	5.35	0.68	3.69	5.35	0.67	3.62
	Image 2	5.35	0.69	3.73			
	Image 3	5.36	0.64	3.49			
	Image 4	5.35	0.66	3.59			
	Image 5	5.36	0.66	3.58			
YOLO11x	Image 1	5.33	0.67	3.59	5.35	0.67	3.62
	Image 2	5.36	0.66	3.59			
	Image 3	5.35	0.67	3.62			
	Image 4	5.35	0.68	3.66			
	Image 5	5.35	0.67	3.63			

(b) NPU-based

Model	Picture	Average per picture			Average per model		
		Voltage (V)	Current (A)	Power (W)	Voltage (V)	Current (A)	Power (W)
YOLO11n	Image 1	5.37	0.68	3.67	5.38	0.66	3.57
	Image 2	5.38	0.66	3.59			
	Image 3	5.38	0.65	3.54			
	Image 4	5.38	0.65	3.53			
	Image 5	5.38	0.65	3.50			
YOLO11s	Image 1	5.38	0.66	3.56	5.38	0.65	3.61
	Image 2	5.38	0.66	3.76			
	Image 3	5.38	0.65	3.72			
	Image 4	5.38	0.65	3.52			
	Image 5	5.38	0.64	3.48			
YOLO11m	Image 1	5.38	0.68	3.67	5.38	0.66	3.54
	Image 2	5.39	0.64	3.47			
	Image 3	5.38	0.65	3.53			
	Image 4	5.39	0.64	3.48			
	Image 5	5.38	0.66	3.57			
YOLO11l	Image 1	5.38	0.67	3.60	5.38	0.65	3.50
	Image 2	5.38	0.65	3.48			
	Image 3	5.38	0.64	3.45			
	Image 4	5.39	0.64	3.46			
	Image 5	5.39	0.65	3.51			
YOLO11x	Image 1	5.38	0.67	3.60	5.38	0.66	3.57
	Image 2	5.38	0.66	3.61			
	Image 3	5.39	0.66	3.55			
	Image 4	5.38	0.65	3.50			
	Image 5	5.38	0.66	3.56			

Table 5. Unified summary of CPU vs. NPU performance for YOLO11 variants, including latency, energy per inference, and accuracy

Model	CPU time (ms)	NPU time (ms)	Speedup (×)	CPU power (W)	NPU power (W)	CPU energy (J)	NPU energy (J)	Energy saving (%)	Official YOLO11 mAP	On-device mAP (5,000)	Δ mAP
YOLO11n	1,587.14	320.74	4.95	3.60	3.59	5.71	1.15	79.8	0.395	0.369	-0.026
YOLO11s	4,897.76	485.56	10.09	3.60	3.59	17.63	1.74	90.1	0.470	0.441	-0.029
YOLO11m	16,060.10	1,002.36	16.02	3.60	3.59	57.82	3.60	93.8	0.515	0.487	-0.028
YOLO11l	20,177.26	1,208.48	16.70	3.60	3.59	72.64	4.34	94.0	0.534	0.504	-0.030
YOLO11x	40,441.44	2,663.44	15.18	3.60	3.59	145.59	9.56	93.4	0.547	0.517	-0.030

These findings support real-time or near-real-time deployment of modern object detection on affordable embedded platforms, which has practical significance. The RK3566 NPU enables 5–17× faster inference at comparable accuracy (ΔmAP ≈ -0.03) with ~80–94% lower energy per inference, increasing the feasibility of applications in surveillance, robotics, and IoT edge workloads.

$$Speedup\ (\times)$$
$$Speedup = \frac{CPU\ Time}{NPU\ Time} \quad (5)$$

$$CPU\ Energy\ (J)$$
$$E_{CPU} = P_{CPU}\ x\ \frac{CPU\ Time}{1000} \quad (6)$$

$$NPU\ Energy\ (J)$$
$$E_{NPU} = P_{NPU}\ x\ \frac{NPU\ Time}{1000} \quad (7)$$

$$Energy\ Saving\ (\%)$$
$$Energy\ Saving\ (\%)\ = \frac{E_{CPU} - E_{NPU}}{E_{CPU}}\ x\ 100 \quad (8)$$

$$\Delta\ mAP$$
$$\Delta mAP\ =\ on-device\ mAP\ 5{,}000\ -\ Official\ mAP \quad (9)$$

To explain the origin of latency reductions, Table 6 reports the relative changes in preprocessing, inference, and postprocessing times when shifting from CPU to NPU, showing that modest CPU-bound overheads are outweighed by dominant inference acceleration.

Table 6. Relative change in processing time when shifting from CPU to NPU (Δ time in %, relative to CPU execution)

Model	Preprocess Δ time (%)	Inference Δ time (%)	Postprocess Δ time (%)	Total time Δ (%)
YOLO11n	+ 8.40	- 82.3	+ 12.9	- 79.8
YOLO11s	+ 24.5	- 91.0	+ 30.1	- 90.1
YOLO11m	+ 21.3	- 94.0	+ 26.7	- 93.8
YOLO11l	+ 21.4	- 94.2	+ 30.0	- 94.0
YOLO11x	+ 29.5	- 93.5	+ 29.6	- 93.4

Table 5 summarizes the aggregate performance latency, energy per inference, and accuracy across YOLO11 variants. However, these metrics do not capture scene-specific behaviors. Table 6 provides a stage-wise breakdown, showing that preprocessing and postprocessing incur modest increases (8–30%), whereas inference dominates acceleration with reductions of 82–94%. This view clarifies the source of latency gains in Table 5.

To contextualize these quantitative results, Figure 6 presents representative detection scenarios. These qualitative case studies illustrate how the latency and accuracy patterns in Table 5 appear in real images, highlighting localized variations under diverse conditions. In the warehouse scene, as shown in Figure 7(a), CPU inference required 1,466.3 ms and detected four persons and two trucks, whereas the NPU completed inference in 321.9 ms and detected six persons and two trucks, including one additional person at a confidence of 0.28. Both configurations occasionally misclassified stacked boxes as trucks.

In the crosswalk scenario, as shown in Figure 7(b), the inference duration for the CPU was 5,162.1 ms, and it detected 13 persons, 2 cars, 2 traffic lights, 3 handbags, 1 tie, and 1 cup. The NPU, in 496.2 ms, reported 12 persons, 2 cars, 1 traffic light, 3 handbags, 1 tie, and 1 cup. The crowded setting caused missed detections of small or occluded pedestrians, lowering frame-level recall Table 7.

In the parking lot test Figure 7(c), CPU processing lasted 14,807.5 ms, yielding one person, 26 cars, and five trucks. The NPU, at 1,004.3 ms, detected one person, 29 cars, and five trucks, showing more consistent vehicle counts under dense traffic. This latency improvements align with the trends observed from Table 6.

In the classroom image Figure 7(d), CPU and NPU produced consistent results across YOLO11 variants, detecting 7–8 people and occasionally one tie. Tie detections occurred in both outputs, reflecting model behavior rather than hardware differences.

In the nighttime CCTV test Figure 7(e), CPU processing consumed 43,171.4 ms, detecting seven persons, two cars, one airplane, and one tie. The NPU, in 2,646.8 ms, found seven persons and two cars, omitting smaller objects but maintaining higher confidence on detected individuals. These results illustrate the latency gap in Table 6 and precision–recall trade-offs in Table 7.

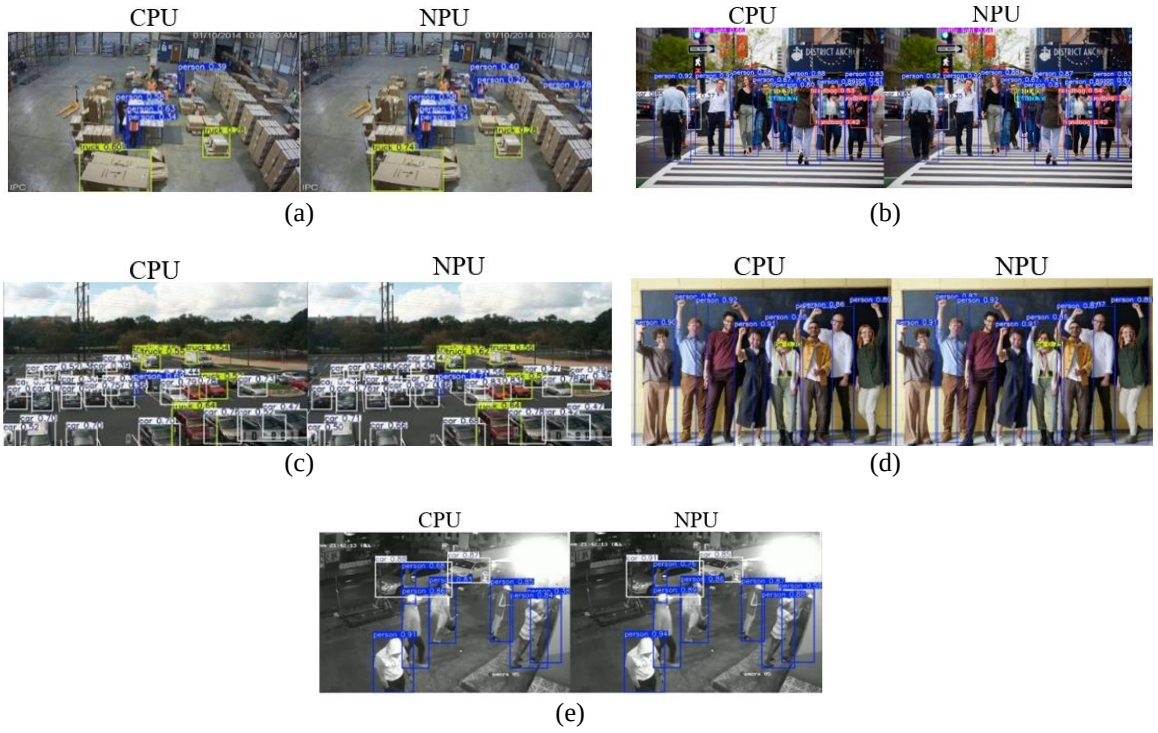

Figure 7. CPU (left) and NPU (right) inference results across representative scenarios: (a) warehouse (YOLO11n), (b) crosswalk (YOLO11s), (c) parking lot (YOLO11m), (d) classroom (YOLO11l), and (e) nighttime CCTV using YOLO11 variants (YOLO11x)

Overall, these cases show that although aggregate mAP deviation from the official benchmark is small (-0.026 to -0.030 across variants, Table 4), localized variations, missed detections, over-detections, and misclassifications persist. CPU and NPU inferences are broadly consistent, yet subtle divergences emerge under challenging conditions, underscoring the need to balance speed and energy gains with robustness in safety-critical deployments.

Table 6, Figure 7, and Table 7 provide a comprehensive view of CPU–NPU differences. They also highlight detection variations across YOLO11 scales. The NPU consistently shortens processing time [45] by over 80% while maintaining narrow accuracy deviations. Local anomalies, such as false positives or missed detections, appeared only in certain scenes, confirming situational rather than systemic differences.

Figure 7 illustrates CPU–NPU differences across diverse scenes. It reveals occasional false positives and missed detections. Confidence also shifts in crowded or low-light conditions. To quantify these variations, Table 6 summarizes the frame-level ground truth alongside detection outcomes for both backends across the same five scenarios. Precision and recall values show that CPU and NPU deliver broadly comparable accuracy,

with on-device mAP deviations confined to -0.026 to -0.030. Local discrepancies occur sporadically rather than systematically, confirming that the NPU maintains detection quality relative to the CPU baseline while achieving substantial improvements in processing efficiency.

Although all YOLO11 variants (n, s, m, l, and x) used identical pretrained weights, CPU and NPU inference showed small differences. PyTorch executed models on the CPU, whereas RKNN kernels ran on the NPU with parallel scheduling. Kernel-level differences reorder floating-point operations, which are nonassociative in FP32, producing small rounding variations. With wide receptive fields, these shifts can alter feature activations and confidence scores.

Further variations stem from preprocessing and postprocessing. The RKNN pipeline tiles inputs for device transfer and applies interpolation and normalization that is not bit-identical to PyTorch. Its NMS also uses different tie-breaking rules; thus, boxes near thresholds (e.g., 0.49 vs. 0.51) may be retained on one runtime but suppressed on another. These add slight preprocessing and postprocessing overheads, though NPU inference remains much faster.

In the warehouse case Figure 7(a), the CPU detected four persons and two trucks in 1,466.3 ms, whereas the NPU detected six persons and two trucks in 321.9 ms. One NPU detection (confidence 0.28) misclassified a hanging raincoat as a person, and both runtimes sometimes labeled stacked boxes as trucks. Such errors reflect the probabilistic nature of YOLO11, which assigns the most likely class from training data, so visually ambiguous objects may align with incorrect labels.

These factors explain occasional low confidence or borderline differences between CPU and NPU. Deviations were local, with mAP bias limited to -0.026 to -0.030. Detection quality remained consistent, but the NPU achieved up to 16.7× lower latency and superior energy efficiency.

Table 7 summarizes relative processing-time changes when shifting from CPU to NPU. Preprocessing and postprocessing rose modestly (8–30%) due to interface and conversion overheads, while inference dropped substantially (82–94%), yielding ~80–94% overall latency reduction. This confirms NPU overheads are minor compared with dominant inference acceleration.

Table 7. Frame-level ground truth and comparative detection performance between CPU and NPU

Object / image	GT (person / vehicle / traffic)	CPU detection	NPU detection	Precision (CPU)	Recall (CPU)	Precision (NPU)	Recall (NPU)
Image 1 (classroom)	8 / 0 / 0	8 persons, 1 tie (FP)	8 persons, 1 tie	0.90	1.00	1.00	1.00
Image 2 (crosswalk)	13/ 4 / 2	13 persons, 2 cars, 2 traffic lights, extras	12 persons, 2 cars, 1 traffic light	0.93	0.86	0.89	0.79
Image 3 (warehouse)	7 / 0 / 0	4 persons, 2 trucks (FP)	6 persons, 2 trucks (1 FP low-conf)	0.67	0.57	0.75	0.86
Image 4 (parking lot)	1 / 40 / 0	1 person, 26 cars, 5 trucks	1 person, 29 cars, 5 trucks	1.00	0.87	1.00	0.97
Image 5 (nighttime)	7 / 2 / 0	7 persons, 2 cars, +FP airplane/tie	7 persons, 2 cars	0.75	1.00	1.00	1.00

A comparative evaluation of YOLO11 inference on CPU versus NPU shows clear benefits from hardware acceleration. End-to-end latency decreased by 80–94%, confirming the RK3566 NPU's suitability for low-latency image inference. Inference time drops sharply, whereas preprocessing and postprocessing increase modestly (8–30%) due to transfer, synchronization, and format conversion overheads. These stages form the main bottlenecks, suggesting that future designs should offload postprocessing to the NPU, minimize host–device transfers, and standardize data interfaces. The findings apply only to per-image inference on static inputs; sequential or temporal processing was excluded.

INT8 instability likely arises from unsupported or partially supported RKNN operators and calibration-range mismatch; per-tensor quantization with activation outliers can saturate the scales, driving class logits below thresholds and yielding empty or degraded detections. In our RK3566 pipeline, NMS remains on the CPU because a compatible batched-NMS kernel is unavailable; coupled with host–device tensor shuttling, this explains the modest increase in postprocessing despite large NPU-side speedups.

Accuracy was stable, with mAP deviations of -0.026 to -0.030 relative to the benchmark. Qualitative differences were limited to low-confidence or ambiguous cases (e.g., raincoats as persons, stacked boxes as trucks), reflecting dataset limits rather than hardware constraints [46]. Future work should expand training diversity, include rare categories, and apply domain-adaptive fine-tuning.

This study extends that of Tan and Cao [47], who explored hybrid CPU–NPU partitioning on early FP16/INT8 NPUs where full offload risked accuracy loss. With the RK3566 and PyTorch–ONNX–RKNN toolchain, YOLO11 runs fully on the NPU without accuracy degradation, achieving up to 16.7× speedup and major energy savings. Hybrid partitioning still applies under strict precision or quantized pipelines, while modern stacks enable accurate full-NPU inference.

To provide a compact numerical baseline, Table 8 summarizes throughput (FPS) and mAP across various platforms: RK3566, Intel Myriad X on Raspberry Pi 4 as reported by Feng *et al.* [48], and Coral Edge TPU as reported by Amanatidis *et al.* [49]. Because the architectures, precisions, and input sizes differ, these figures are contextual baselines rather than a head-to-head comparison. Future work will extend to sequential inference (e.g., streaming, tracking) and reassess NMS placement and INT8 quantization stability on the RK3566 NPU.

Overall, the RK3566 NPU delivered substantial efficiency gains without compromising detection quality. Continued progress depends on optimizing the preprocessing and postprocessing pipeline and enriching datasets to mitigate misclassification in complex real-world scenes. Future work will extend to sequential inference (e.g., streaming and tracking) and reassess NMS placement and INT8 quantization stability on the RK3566 NPU. All data supporting this study are openly available on Zenodo [50].

Table 8. Cross-platform summary

Platform	Model (precision and input)	Throughput	Accuracy / ΔmAP
Orange Pi 3B + RK3566	YOLO11 n/s/m/l/x (FP32, 640)	3.12 / 2.06 / 1.00 / 0.83 / 0.38 FPS	−0.026 / −0.029 / −0.028 / −0.030 / −0.030
RPi4 + Intel Myriad X (NCS2)	YOLOv3 (INT8/FP16, 416)	≈ 2.5 FPS	n/a (pipeline-dependent)
RPi4 + Intel Myriad X (NCS2)	YOLOv3-tiny (INT8/FP16, 416)	≈ 19.0 FPS	n/a (unstable across sequences)
RPi4 + Coral Edge TPU	YOLOv5 (INT8, 224)	≈ 10.1–11.1 FPS	mAP ≈ 0.44–0.45

- n/a (pipeline-dependent): accuracy not reported due to conversion variability (e.g., region layers, anchors, INT8 calibration).
- n/a (unstable): mean confidence fluctuates across sequences; no representative accuracy.

4. CONCLUSION

To our knowledge, this is the first comprehensive benchmark of YOLO11 on the Orange Pi 3B with the RK3566 NPU. Evaluations of latency, energy, and accuracy show >80% reduction in inference time, up to 94% lower end-to-end latency, and similar reductions in energy per inference, while maintaining detection reliability (mAP deviations of -0.026 to -0.030 vs. the official Ultralytics map report). These results establish RK3566 as a feasible, efficient, low-cost platform for edge object detection.

Two limitations emerged: i) CPU-bound preprocessing/postprocessing overheads from data transfers and format conversions and ii) INT8 quantization on RK3566, which did not yield valid results, requiring FP32 for all experiments. Future work should integrate postprocessing into the NPU pipeline, streamline conversion workflows to reduce interface costs, stabilize INT8 for efficient execution, and validate performance in domain deployments (e.g., traffic monitoring, industrial inspection, and mobile robotics) and distributed multi-board settings. This demonstrates that low-cost NPUs can enable real-time edge-AI applications without compromising accuracy.

FUNDING INFORMATION

This research received no external funding; no research grant or contract supported this work. All costs were covered by the authors' personal funds.

AUTHOR CONTRIBUTIONS STATEMENT

This journal uses the Contributor Roles Taxonomy (CRediT) to recognize individual author contributions, reduce authorship disputes, and facilitate collaboration.

Name of Author	C	M	So	Va	Fo	I	R	D	O	E	Vi	Su	P	Fu
Alwin Hartono Limaran	✓	✓	✓	✓	✓	✓	✓	✓	✓					✓
Agung Wicaksono			✓		✓		✓	✓		✓	✓			✓
Patah Herwanto				✓	✓	✓		✓		✓		✓	✓	

C : **C**onceptualization
M : **M**ethodology
So : **So**ftware
Va : **Va**lidation
Fo : **Fo**rmal analysis
I : **I**nvestigation
R : **R**esources
D : **D**ata Curation
O : Writing - **O**riginal Draft
E : Writing - Review & **E**diting
Vi : **Vi**sualization
Su : **Su**pervision
P : **P**roject administration
Fu : **Fu**nding acquisition

CONFLICT OF INTEREST STATEMENT

The authors declare no conflicts of interest.

DATA AVAILABILITY

The data supporting the findings of this study are openly available at Zenodo under the following DOI: https://doi.org/10.5281/zenodo.17175124.

REFERENCES

[1] P. Mittal, "A comprehensive survey of deep learning-based lightweight object detection models for edge devices," *Artificial Intelligence Review*, vol. 57, no. 9, p. 242, Aug. 2024, doi: 10.1007/s10462-024-10877-1.

[2] J. Rajasekhar, "Understanding YOLO: real-time object detection explained," I*nternational Journal of Scientific Research in Engineering and Management* (IJSREM), vol. 08, no. 07, pp. 1–9, Jul. 2024, doi: 10.55041/ijsrem36359.

[3] H. S. Abdul-Ameer, H. J. Hassan, and S. H. Abdullah, "Development smart eyeglasses for visually impaired people based on you only look once," *TELKOMNIKA (Telecommunication Computing Electronics and Control)*, vol. 20, no. 1, pp. 109–115, Feb. 2022, doi: 10.12928/telkomnika.v20i1.22457.

[4] C. P. Thrupthi, K. Chitra, and V. M. Harilakshmi, "Object Detection for Autonomous Vehicles Using YOLO Algorithm," *International Journal of Advanced Research in Science, Communication and Technology (IJARSCT)*, pp. 180–184, Nov. 2024, doi: 10.48175/ijarsct-22539.

[5] M.-T. Chen, C.-H. Chen, C.-T. Lin, and Y.-Y. Chang, "Enhancing YOLOv8 by adding global attention mechanism to identify targets in complex backgrounds," *Sensors and Materials*, vol. 37, no. 7, pp. 2935–2948, Jul. 2025, doi: 10.18494/SAM5696.

[6] M. Yang and X. Fan, "YOLOv8-Lite: A lightweight object detection model for real-time autonomous driving systems," *IECE Transactions on Emerging Topics in Artificial Intelligence*, vol. 1, no. 1, pp. 1–16, Apr. 2024, doi: 10.62762/TETAI.2024.894227.

[7] J.-S. Park *et al.*, "A multi-mode 8k-MAC HW-utilization-aware neural processing unit with a unified multi-precision Datapath in 4-nm flagship mobile SoC," *IEEE Journal of Solid-State Circuits*, vol. 58, no. 1, pp. 189–202, Jan. 2023, doi: 10.1109/JSSC.2022.3205713.

[8] N. Herterich, K. Liu, and A. Stein, "Accelerating weed detection for smart agricultural sprayers using a neural processing unit," *Computers and Electronics in Agriculture*, vol. 237, Oct. 2025, doi: 10.1016/j.compag.2025.110608.

[9] K. M. Hosny, A. Magdi, A. Salah, O. El-Komy, and N. A. Lashin, "Internet of things applications using Raspberry-Pi: a survey," *International Journal of Electrical and Computer Engineering (IJECE)*, vol. 13, no. 1, p. 902, Feb. 2023, doi: 10.11591/ijece.v13i1.pp902-910.

[10] S. Karthikeyan, R. A. Raj, M. V. Cruz, L. Chen, J. L. A. Vishal, and V. S. Rohith, "A Systematic Analysis on Raspberry Pi Prototyping: Uses, Challenges, Benefits, and Drawbacks," *IEEE Internet of Things Journal*, vol. 10, no. 16, pp. 14397–14417, Aug. 2023, doi: 10.1109/jiot.2023.3262942.

[11] S. Kim, C. Kim, and S. Kim, "Improving performance of real-time object detection in edge device through concurrent multi-frame processing," *IEEE Access*, vol. 13, pp. 1522–1533, 2025, doi: 10.1109/ACCESS.2024.3520240.

[12] M. L. Ali and Z. Zhang, "The YOLO framework: A comprehensive review of evolution, applications, and benchmarks in object detection," *Computers*, vol. 13, no. 12, Dec. 2024, doi: 10.3390/computers13120336.

[13] A. Zagitov, E. Chebotareva, A. Toschev, and E. Magid, "Comparative analysis of neural network models performance on low-power devices for a real-time object detection task," *Computer Optics*, vol. 48, no. 2, pp. 242–252, Apr. 2024, doi: 10.18287/2412-6179-CO-1343.

[14] G. Jocher and J. Qiu, "Ultralytics YOLO11," *GitHub*, 2024. https://github.com/ultralytics/ultralytics

[15] L. Boyle, J. Moosmann, N. Baumann, S. Heo, and M. Magno, "DSORT-MCU: detecting small objects in real time on microcontroller units," *IEEE Sensors Journal*, vol. 24, no. 24, pp. 40231–40239, Dec. 2024, doi: 10.1109/JSEN.2024.3425904.

[16] X. Wei, Y. Zhang, and Y. Zheng, "BSFCDet: Bidirectional spatial–semantic fusion network coupled with channel attention for object detection in satellite images," *Remote Sensing*, vol. 15, no. 13, Jun. 2023, doi: 10.3390/rs15133213.

[17] T. Hu, W. Wang, J. Gu, Z. Xia, J. Zhang, and B. Wang, "Research on apple object detection and localization method based on improved YOLOX and RGB-D images," *Agronomy*, vol. 13, no. 7, Jul. 2023, doi: 10.3390/agronomy13071816.

[18] O. Spantidi *et al.*, "Targeting DNN inference via efficient utilization of heterogeneous precision DNN accelerators," *IEEE Transactions on Emerging Topics in Computing*, vol. 11, no. 1, pp. 112–125, Jan. 2023, doi: 10.1109/TETC.2022.3178730.

[19] C. Wang, Y. Luo, W. Du, K. Wang, N. Gu, and J. Yu, "Faster and stronger: Unleashing data processing potential through hardware heterogeneity," *IEEE Internet of Things Journal*, vol. 12, no. 10, pp. 14559–14576, May 2025, doi: 10.1109/JIOT.2025.3526662.

[20] J. Fowers *et al.*, "A configurable cloud-scale DNN processor for real-time AI," in *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*, Jun. 2018, pp. 1–14. doi: 10.1109/ISCA.2018.00012.

[21] Y. Yang, F. Feng, G. Liu, and J. Di, "MEL-YOLO: A novel YOLO network with multi-scale, effective, and lightweight methods for small object detection in aerial images," *IEEE Access*, vol. 12, pp. 194280–194295, 2024, doi: 10.1109/ACCESS.2024.3517663.

[22] X. Liu *et al.*, "A 33nW fully autonomous SoC with distributed cooperative energy harvesting and multi-chip power management for mm-scale system-in-fiber," *IEEE Transactions on Biomedical Circuits and Systems*, vol. 17, no. 6, pp. 1185–1201, Dec. 2023, doi: 10.1109/TBCAS.2023.3309779.

[23] J. Park, Y. Kwon, Y. Park, and D. Jeon, "Microarchitecture-aware code generation for deep learning on single-ISA heterogeneous multi-core mobile processors," *IEEE Access*, vol. 7, pp. 52371–52378, 2019, doi: 10.1109/ACCESS.2019.2910559.

[24] N. P. Jouppi *et al.*, "In-datacenter performance analysis of a tensor processing unit," in *Proceedings of the 44th Annual International*

Symposium on Computer Architecture, Jun. 2017, pp. 1–12. doi: 10.1145/3079856.3080246.
[25] K. Ando *et al.*, "BRein memory: A single-chip binary/ternary reconfigurable in-memory deep neural network accelerator achieving 1.4 TOPS at 0.6 W," *IEEE Journal of Solid-State Circuits*, vol. 53, no. 4, pp. 983–994, Apr. 2018, doi: 10.1109/JSSC.2017.2778702.
[26] G. Raut, P. J. Edavoor, D. Selvakumar, and R. Thakur, "A SIMD Dynamic Fixed Point Processing Engine for DNN Accelerators," *2024 25th International Symposium on Quality Electronic Design (ISQED). IEEE*, pp. 1–8, Apr. 03, 2024. doi: 10.1109/isqed60706.2024.10528758.
[27] C. Liu *et al.*, "A low-power hybrid-precision neuromorphic processor with INT8 inference and INT16 online learning in 40-nm CMOS," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 70, no. 10, pp. 4028–4039, Oct. 2023, doi: 10.1109/TCSI.2023.3300095.
[28] J. Osorio, A. Armejach, E. Petit, G. Henry, and M. Casas, "A BF16 FMA is all you need for DNN training," *IEEE Transactions on Emerging Topics in Computing*, vol. 10, no. 3, pp. 1302–1314, Jul. 2022, doi: 10.1109/TETC.2022.3187770.
[29] F. Hermens, "Automatic object detection for behavioural research using YOLOv8," *Behavior Research Methods*, vol. 56, no. 7, pp. 7307–7330, May 2024, doi: 10.3758/s13428-024-02420-5.
[30] Z. Cai and N. Vasconcelos, "Cascade R-CNN: High quality object detection and instance segmentation," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 43, no. 5, pp. 1483–1498, May 2021, doi: 10.1109/TPAMI.2019.2956516.
[31] Z. Yang, S. Liu, H. Hu, L. Wang, and S. Lin, "RepPoints: Point set representation for object detection," in *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, Oct. 2019, pp. 9656–9665. doi: 10.1109/ICCV.2019.00975.
[32] E. H. Benlahmar, E. Y. Radid, and S. El Filali, "Real-Time Image Preprocessing for Abandoned Object Detection in Urban Surveillance," *2025 International Conference on Circuit, Systems and Communication (ICCSC). IEEE*, pp. 1–4, Jun. 19, 2025. doi: 10.1109/iccsc66714.2025.11134929.
[33] F. Zandamela, D. Kunene, V. Skosana, and G. Stoltz, "Lightweight YOLO for distracted driver detection on edge devices," *MATEC Web of Conferences*, vol. 406, p. 10001, 2024, doi: 10.1051/matecconf/202440610001.
[34] S. Zhang, X. Mu, G. Kou, and J. Zhao, "Object detection based on efficient multiscale auto-inference in remote sensing images," *IEEE Geoscience and Remote Sensing Letters*, vol. 18, no. 9, pp. 1650–1654, Sep. 2021, doi: 10.1109/LGRS.2020.3004061.
[35] H. Ibrahem, A. Salem, and H.-S. Kang, "LEOD-net: Learning line-encoded bounding boxes for real-time object detection," *Sensors*, vol. 22, no. 10, May 2022, doi: 10.3390/s22103699.
[36] E. Manor and S. Greenberg, "Custom hardware inference accelerator for TensorFlow lite for microcontrollers," *IEEE Access*, vol. 10, pp. 73484–73493, 2022, doi: 10.1109/ACCESS.2022.3189776.
[37] J.-H. Jean and D.-S. Kim, "Hardware-assisted low-latency NPU virtualization method for multi-sensor AI systems," *Sensors*, vol. 24, no. 24, Dec. 2024, doi: 10.3390/s24248012.
[38] E. B. Candrasari, L. Novamizanti, and S. Aulia, "Hand gesture recognition using discrete wavelet transform and hidden Markov models," *TELKOMNIKA (Telecommunication Computing Electronics and Control)*, vol. 18, no. 5, pp. 2265–2273, Oct. 2020, doi: 10.12928/telkomnika.v18i5.13725.
[39] A. H. Al-Badri, N. A. Ismail, K. Al-Dulaimi, G. A. Salman, and M. S. H. Salam, "Adaptive non-maximum suppression for improving performance of Rumex detection," *Expert Systems with Applications*, vol. 219, Jun. 2023, doi: 10.1016/j.eswa.2023.119634.
[40] L. Rui, X. Tang, and K. Hao, "DB-NMS: improving non-maximum suppression with density-based clustering," *Neural Computing and Applications*, vol. 34, no. 6, pp. 4747–4757, Mar. 2022, doi: 10.1007/s00521-021-06628-w.
[41] L. Li, S. Zhang, and J. Wu, "Efficient object detection framework and hardware architecture for remote sensing images," *Remote Sensing*, vol. 11, no. 20, Oct. 2019, doi: 10.3390/rs11202376.
[42] H. Li, J. Liu, L. Jia, Y. Liang, Y. Wang, and M. Tan, "Downscaling and overflow-aware model compression for efficient vision processors," in *2022 IEEE 42nd International Conference on Distributed Computing Systems Workshops (ICDCSW)*, Jul. 2022, pp. 145–150. doi: 10.1109/ICDCSW56584.2022.00036.
[43] J. Schauer, P. Goodarzi, J. Morsch, and A. Schütze, "A performance study of deep neural network representations of Interpretable ML on edge devices with AI accelerators," *Sensors*, vol. 25, no. 18, Sep. 2025, doi: 10.3390/s25185681.
[44] X. Gao, "TAS: A temperature-aware scheduling for heterogeneous computing," *IEEE Access*, vol. 11, pp. 54773–54781, 2023, doi: 10.1109/ACCESS.2023.3281839.
[45] Y. Xing *et al.*, "Real-time high-quality single-lens computational imaging via enhancing lens modulation transfer function consistency," *Optics Express*, vol. 33, no. 3, Feb. 2025, doi: 10.1364/OE.552050.
[46] D. Miller, G. Goode, C. Bennie, P. Moghadam, and R. Jurdak, "Why Object Detectors Fail: Investigating the Influence of the Dataset," *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW). IEEE*, pp. 4822–4829, Jun. 2022. doi: 10.1109/cvprw56347.2022.00529.
[47] T. Tan and G. Cao, "Efficient execution of deep neural networks on mobile devices with NPU," in *Proceedings of the 20th International Conference on Information Processing in Sensor Networks (co-located with CPS-IoT Week 2021)*, May 2021, pp. 283–298. doi: 10.1145/3412382.3458272.
[48] H. Feng, G. Mu, S. Zhong, P. Zhang, and T. Yuan, "Benchmark analysis of YOLO performance on edge intelligence devices," *Cryptography*, vol. 6, no. 2, Apr. 2022, doi: 10.3390/cryptography6020016.
[49] P. Amanatidis, D. Karampatzakis, G. Iosifidis, T. Lagkas, and A. Nikitas, "Cooperative task execution for object detection in edge computing: An Internet of things application," *Applied Sciences*, vol. 13, no. 8, Apr. 2023, doi: 10.3390/app13084982.
[50] A. H. Limaran, YOLO11 n,s,m,l,x mAP Val500 COCO 2017 with Orange Pi 3B RK3566 NPU – Full Inference & Evaluation Scripts. *Zenodo*, 2025. doi: 10.5281/ZENODO.17175124.

BIOGRAPHIES OF AUTHORS

Alwin Hartono Limaran is pursuing a Bachelor's degree in Informatics Engineering at STMIK Indonesia Mandiri, Bandung, Indonesia. He works as an IT and cybersecurity professional in the private sector and holds the Certified International Information System Auditor (CIISA) credential from the American Academy of Financial Management (AAFM). His research interests include embedded artificial intelligence, neural hardware acceleration, IoT security, vision computing, and cybersecurity for low-power edge computing systems. He is affiliated with the Department of Informatics Engineering, Faculty of Information Technology, STMIK Indonesia Mandiri, Bandung 40281, Indonesia. He can be contacted at email: alwin.hartono@protonmail.com.

Agung Wicaksono is pursuing a Bachelor's degree in Informatics Engineering at STMIK Indonesia Mandiri, Bandung, Indonesia. He works as an aircraft technician at a private aviation company. His background in aerospace engineering is complemented by strong interests in embedded systems, IoT, and vision computing for intelligent systems. He is affiliated with the Department of Informatics Engineering, Faculty of Information Technology, STMIK Indonesia Mandiri, Bandung 40281, Indonesia. He can be contacted at email: agung.wicaksono.362389010@gmail.com.

Patah Herwanto is a lecturer in the Informatics Engineering Department at STMIK Indonesia Mandiri, Bandung, Indonesia. He holds a Master of Computer Science from STMIK Likmi (2011), a Bachelor of Engineering from STMIK Indonesia Mandiri (2002), and an Associate Degree in Informatics Engineering from STMIK Bandung (1998). His academic and professional interests include software engineering, embedded systems, intelligent computing, and IT infrastructure. He is affiliated with the Department of Informatics Engineering, Faculty of Information Technology, STMIK Indonesia Mandiri, Bandung 40281, Indonesia. He can be contacted at email: pherwanto@stmik-im.ac.id.