International Innovative Research Journal of Engineering and Technology
Vol. 12, Issue. 1, Sep 2026, pp. 86~98
ISSN: 2456-1983, DOI: 10.32595/iirjet.org/v12i1.2026.255

Design and FPGA Implementation of a Quantized YOLO Accelerator for Low-Latency Edge Object Detection

Dr. Sivachandiran Sivaraju[1], R.Anandbabu[2]

[1]Deputy Dean, International Training Institute, Port Moresby, Papua New Guinea.
[2]International Training Institute, Port Moresby, Papua New Guinea.

Article Info

Article History:

Received Jun 18, 2026
Revised Jul 20, 2026
Accepted Aug 18, 2026

Keywords:

YOLO,
FPGA Accelerator,
Model Quantization,
Edge Computing,
Object Detection,
Low-Latency Inference,
Parallel Processing,
Hardware Optimization

ABSTRACT

Real-time object detection on edge devices is challenging because such platforms must deliver low inference latency and high throughput under tight hardware-resource and power budgets. This work presents the design and field-programmable gate array (FPGA) implementation of a quantized YOLO (You Only Look Once) accelerator for edge object detection. The proposed approach combines a YOLO-based detection model with reduced arithmetic precision, parallel processing, pipelining, and memory-efficient dataflow to improve inference efficiency. Model quantization lowers the memory footprint and computational demand of the detector while limiting accuracy loss, and the FPGA architecture is optimized to accelerate computationally intensive operations such as convolution. The accelerator is evaluated in terms of mean average precision (mAP), precision, recall, inference latency, throughput, FPGA resource utilization, power consumption, and energy per inference, and it is compared with the corresponding floating-point and quantized software models. In the measured comparison, the accelerator that is proposed achieves the highest across both latency, throughput and power and energy efficiency in technical data collected while maintaining a detection accuracy that reaches 91% instead of 93% that the floating point approach achieves. The results demonstrate that it is possible to utilize quantization with the parallel, pipelined architectural approach based on FPGA in order to significantly lower cost of YOLO inferences executed on edge devices with very little accuracy decrease.

Corresponding Author:

Dr Sivachandiran Sivaraju,
Deputy dean,
International Training Institute,
Port Moresby, Papua New Guinea.

1. INTRODUCTION

Computer vision systems' capabilities have been substantially improved by the rapid the rise of deep learning and artificial intelligence technologies, particularly in terms of their application in real-time object detection. The technology of object detection is being deployed in various domains; they include robotics, industrial inspection, autonomous systems, monitoring, smart transport, and applications utilizing edge-based vision technology. However, implementation

of modern deep learning algorithms and techniques on an edge devices with limited resources is rather challenging due to their demanding computational and memory bandwidth requirements. Thus, there is a need in hardware architecture with high throughput, low latency, and improved energy efficiency to satisfy the growing technologies according to non-time-consuming approach [1] [2].

YOLO and its relatives have gained great popularity in the field of computer vision, a popularity that stems from their capability of determining not only the existence of an object in a frame but also its location and the probability of class occurrence. These techniques manage to strike a better balance between time/cost and performance than any traditional cascade detector. They have advanced also due to innovations in the area of modern network architecture and training methods their suitability for applications that depend on latency. Nevertheless, these models' computing requirements may still be too great for embedded vision systems with limited resources, which is why specialized accelerators are required [1].

Accelerators for FPGAs can improve inference efficiency through parallel operation and customizable dataflow architectures. But YOLO networks remain computationally intensive, as their convolution layers involve a massive number of high data traffic and multiply-accumulate procedures. Sha et al. [3] examined YOLOv6's acceleration on FPGA and determined optimizing calculation and memory access. Therefore, an efficient FPGA implementation must jointly consider computational capability, hardware resources as well as memory bandwidth.

Quantization is an essential model-optimization technique in which the weights and activations of a neural network are shown using values with low precision. Integer in low bits representations consume less memory than floating-point values and allow simpler arithmetic on appropriate hardware. Yan et al. [4] showed an FPGA-based YOLOv5 accelerator using 4-bit quantization, which cut the amount of DSP used by about 50% at the cost of a 1.5% reduction in precision on average (mAP). Their results show the potential of hardware-aware quantization, although its effectiveness depends on the FPGA architecture, dataset, quantization strategy, and model.

In complement to quantization, parallel processing, pipelining and optimizing memory-dataflow are essential for FPGA-based acceleration of neural networks. Parallel processing performs concurrently performing several arithmetic computations, while pipelining overlaps the execution of successive computation phases for a higher throughput. Bandwidth-reduction methods and layer-wise hardware mapping minimizes unnecessary data transfers and make efficient utilization of the available FPGA resources. These techniques have been applied in FPGA-based YOLO implementations [2], [5].

Another essential element for object detection based on edges is power consumption, since embedded vision systems frequently run on platforms with a limited power supply. Chiam et al. [6] experimentally compared quantized YOLO inference on edge devices and found that the Zedboard FPGA and Jetson Nano offered the best balance of speed, accuracy, and power consumption among the tested platforms. However, the results varied significantly with the accelerator and the model architecture. This shows that efficient edge-based object detection must consider power consumption alongside performance gains.

In spite of these advancements, achieving an suitable equilibrium between FPGA resource use, power consumption, throughput, inference latency, and detection accuracy continues to be a significant challenge. Increasing hardware parallelism raises throughput but also increases utilization of FPGA resources and power consumption, whereas intense quantization reduces

hardware complexity but may degrade detection accuracy. Therefore, an efficient hardware–software co-design approach is required to create a low-latency and resource-efficient object-detection accelerator.

2. LITERATURE REVIEW

FPGA-based acceleration of YOLO object-detection models has been the subject of significant research due to the demands for low-latency, high-throughput, and energy-efficient edge inference. FPGAs offer the benefits of reconfigurable hardware resources that enable parallel computation, pipelined execution, local memory buffering, and application-specific dataflow optimization. Recent studies have investigated model quantization, convolution acceleration, memory reuse, and processing-element architectures to accelerate YOLO's deployment on resource-constrained platforms [2], [5], [7].

Heng et al. [7] provided optimization methods for FPGA-based YOLO accelerators, covering both model-level optimizations and the key factors in hardware-architecture design. The review offers useful insight into approaches for improving the performance of FPGA-based object-detection accelerators.

Salah et al. [8] presented a quantized YOLOv10-based FPGA implementation for real-time object detection. The work used full-integer post-training quantization and hardware-aware optimization to reduce computational and memory requirements while limiting accuracy degradation. It also emphasizes how essential it is to properly choose the calibration approach and quantization parameters.

Wang et al. [9] suggested an energy-efficient FPGA accelerator for quantized YOLO-based object detection. The design assembles low-precision math with ping-pong buffering, enhanced data flow and Dual-Multiply Packing to improve computational efficiency and data reuse. The authors argue that efficient use of memory bandwidth and hardware-friendly computation are critical for achieving high performance as energy consumption is decreased. Eydivandi and Beitollahi [10] proposed SARPAR, a systolic-array-based YOLO FPGA accelerator models. The design exploits processing-element parallelism, buffering, and reduced-precision arithmetic, and it demonstrates the substantial role using data reuse and systolic arrays to speed up neural network operations.

Qian et al. [11] created a YOLOv5 based on FPGA accelerator for edge deployment. Optimized convolution procedures are used in the design and dedicated data-access patterns to simplify computation and memory access overhead, highlighting the significance of finding equilibrium between data access and arithmetic calculation in accelerator design.

Kim and Kim [12] suggested using a Zynq-7000 FPGA to construct a YOLOv2 accelerator. Applying the techniques of convolution tiling, pipelining, filter reuse, the utilization of line-buffer-based feature-map reuse, as well as applying 16-bit integer quantization in computations, this work has obtained high detection efficiency with low computational complexity and hardware consumption. It has been emphasized that proper bit-width selection as well as the combination of data reuse and pipelining play a very significant role in the effective implementation of object detection on FPGAs.

Liu et al. [13] investigated FPGA-accelerated deep learning for industrial and engineering applications below the resource constraints. The authors accentuate the need for careful design of

the computational design, memory access, and hardware-resource allocation which facilitates the motivation of the current project.

2.1 Research Gap

Several techniques have been proposed by previous research to speed up the process of YOLO inference on FPGAs, including strategies such as reduced precision quantization, parallelization, pipelining, systolic array configuration, and memory dataflow optimization. These methods emphasize the need to work on the combined improvements of the model algorithms and hardware architectures. However, because these studies differ in YOLO design, dataset, quantization strategy, Resources and implementation for FPGAs technique, their consequences on detection accuracy, latency, and hardware efficiency are not directly equivalent.

The emphasis of this research paper is on the design and evaluation of a quantized YOLO accelerator having reductions in precision, parallelism, pipelining, and memory access. Benchmarking is carried out in accordance with clearly defined baseline criteria, such as qualities of detection accuracy, evaluation speed, hardware utilization, consumption of power, and energy use. While doing so, the paper aims to analyze the key trade-offs related to detection accuracy, inference throughput, and hardware utilization.

3. METHODOLOGY

The paper proposes a quantized YOLO accelerator for FPGA-based edge object detection. The method consists of five stages, including input preprocessing, model quantization, FPGA accelerator design, parallel and pipelined computation, and detection post-processing. The first step involves resizing and normalizing the input images according to the targeted YOLO model's specification. Then, the given model is quantized to the desired weight and activation precision level to minimize the storage requirements. Next, the resulting operations are partitioned and scheduled on the FPGA accelerator, which employs parallel processing elements, computational pipelines, and on-chip memory banks to maximize the convolution throughput and reduce redundant data movements. The obtained feature maps are then processed by the detection module, which outputs the bounding-box coordinates, class probabilities, and confidence scores. Finally, the overlapping bounding boxes are merged using the Non-Maximum Suppression algorithm. The method is evaluated in terms of detection accuracy, latency, throughput, FPGA resource utilization, power consumption, and energy per inference.

Figure 1 shows the proposed architecture of the quantized YOLO accelerator. The input image is resized and normalized to fit the input layer of the quantized YOLO network. The network framework comprises a backbone system that is utilized for feature extraction from images, a neck that serves the purpose of making adjustments to features across multiple scales, and a detection head responsible for finding and identifying the various items visible in the image. The use of low-precision arithmetic leads to reduced computational and storage demands of the implemented processes. The core element of FPGA is enabled to perform convolution process and execute its operations due to parallel pipelines receiving signals through different stages. Temporary storage of weights and values belonging to processes can be either internal or externally connected to the working unit to provide the results needed for making predictions of object classes and bounding boxes of the target object. The overall performance is evaluated using detection metrics and hardware performance indicators.

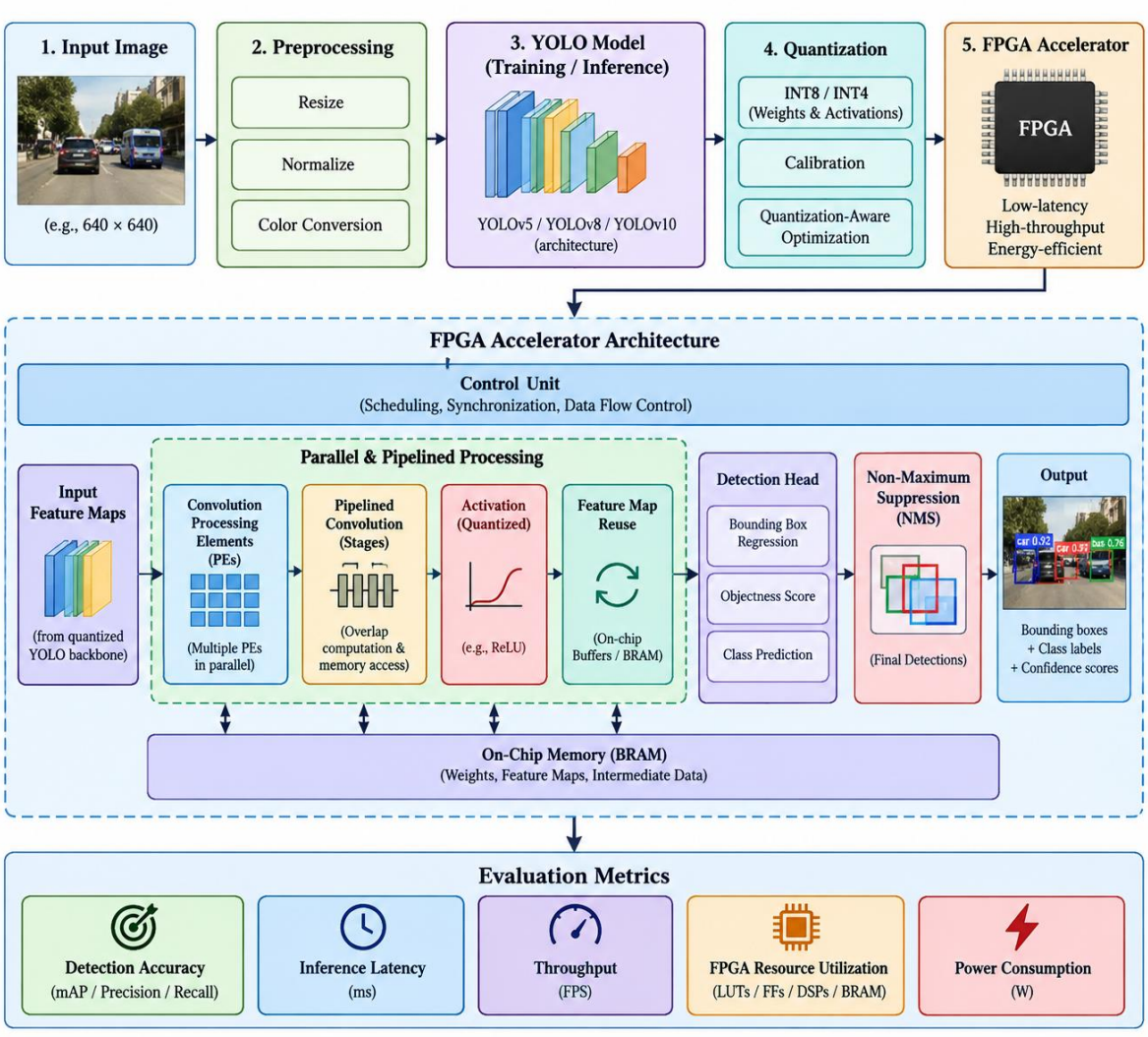


Figure 1. Architecture of the proposed quantized YOLO accelerator for FPGA-based edge object detection.

To enhance hardware utilization and boost total processing throughput, pipeline stages are inserted between computational tasks. The accelerator's pipelined and parallel architecture minimizes inference time while processing feature maps effectively. The FPGA accelerator uses on-chip memory, such as Block RAM (BRAM), to store feature maps, quantization weights, and intermediate calculation outputs. By improving data reuse and lowering the frequency of external memory transfers, on-chip memory lowers memory-access latency and power consumption. The detection head receives the generated feature maps after feature extraction and uses them to produce bounding-box coordinates, objectness scores, and class predictions. The final detections are obtained by applying Non-Maximum Suppression (NMS) to eliminate redundant overlapping bounding boxes. FPGA resource utilization, power consumption, throughput, inference latency, and detection accuracy are used to assess the entire accelerator. Together, these metrics show how detection performance and hardware efficiency are traded off.

The proposed architecture provides a solution for real-time object detection in resource-constrained edge environments by combining YOLO-based detection, low-bit quantization, FPGA acceleration, parallel processing, pipelining, and efficient on-chip memory management.

4. PROPOSED QUANTIZED YOLO ACCELERATOR

The proposed system combines the YOLO object-detection framework with low-bit quantization and FPGA-based hardware acceleration to perform real-time object detection on resource-constrained edge devices. The entire design encompasses an input image processing phase, the quantized YOLO feature detector and recognition framework, various independent image processing units in parallel, effective memory buffers, and an output processing stage. Quantization reduces the number of computations and memory requirements by reducing the accuracy of weights and activations, with subsequent utilization of FPGA parallelism and pipelining allowing for executing quantized operations with minimum delay.

The accelerator is created to facilitate the essential computations of the YOLO model such as convolution, activation, generating the feature map, and relevant detection processes. Instead of executing every operation sequentially, the architecture distributes computations among several processing elements. Pipelining allows several computation stages to operate concurrently, and local memory buffers lower repeated access to external memory. By combining pipelining, parallel processing, quantization, and reduced memory traffic, the design provides an effective method to real-time edge object detection.

4.1 Network Configuration and YOLO Model

YOLO performs object detection with the help of a single neural network, which processes the entire input image. Initially, the input image is resized to the required size before it is analyzed by the neural network. The neck receives all input from various spatial resolutions, while the backbone is responsible for feature extraction from the image. Then the detection head will output the class of the object, as well as bounding-box coordinates and confidence scores. The proposed approach modifies the YOLO network for FPGA deployment by replacing computationally costly floating-point operations with fixed-point or integer-based operations. The desired detection accuracy and available FPGA resources are taken into consideration while choosing the network configuration. Lightweight versions of the YOLO algorithm could be considered for deployment on the edge due to their potential to use fewer parameters and perform less intensive computations than larger variants. The final choice of a particular model should be guided by the balance between detection accuracy and computational complexity as well as the possibility of integer format quantization and available FPGA resources.

4.2 Quantization Technique

High-precision floating-point values are transformed into lower-bit integer representations using quantization. In the proposed accelerator, quantization is used to lower the memory and computational requirements of the YOLO network.

The following is an expression for the quantization process:

$$q = clip(round\left(\frac{x}{S}\right) + Z\,, q_{min}, q_{max}\,) \tag{1}$$

where x represents the original floating-point value, q represents the quantized value, S is the scaling factor, and Z is the zero-point.

The zero-point for symmetric quantization can be set to zero, resulting in:

$$q = clip(round\left(\frac{x}{S}\right)\,, q_{min}, q_{max} \tag{2}$$

The estimated value that was reconstructed is:

$$\hat{x} = S(q - Z) \quad (3)$$

Here, $\hat{x}$ is the reconstructed (de-quantized) approximation of the original floating-point value, q is the quantized value, S is a positive scaling factor, and Z is the zero-point. (q_{min}) and (q_{max}) are the minimum and maximum representable quantized integers. In symmetric quantization, the zero-point is zero. Quantization reduces the bit-width of weights and activations, which reduces their memory footprint and enables faster integer-based processing. The impact of this optimization depends on the chosen bit-width, quantization method, and hardware platform.

4.3 Accelerator Architecture Based on FPGA

The FPGA accelerator contains an input interface, preprocessing module, quantized computation engine, activation and feature-processing units, memory buffers, a detection-output interface, and the control logic that coordinates the execution. Convolutional operations are mapped to parallel processing elements, and local buffers store reusable weights and intermediate results if the available memory allows. In addition, pipelining allows subsequent operations to start before the previous ones have finished, and parallel execution enables different operations to be processed simultaneously. The design is tailored to the supported YOLO operations, numerical precision, data-transfer interfaces, and resource constraints of the target FPGA.

4.4 Convolutional Processing Elements

One of YOLO's most computationally demanding operations is convolution. The convolution operation can be expressed as follows for an input feature map X and kernel W:

$$Y(i,j) = \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} X(i+m, j+n) W(m,n) + b \quad (4)$$

Here, X is an input feature map, W is a convolution kernel, $M \times N$ is a kernel size, b is a bias, and Y is the output feature map (in this case, for a single input channel and filter). A full YOLO convolution can also perform summation (across channels) and include padding, stride, and/or multiple filters.

The FPGA implementation uses several parallel multiply-accumulate (MAC) processes. Integer arithmetic can be used to execute the multiplication and accumulation for quantized inputs and weights:

$$Y_q = \sum_{k=1}^{K} x_q(k) w_q(k) + b_q \quad (5)$$

In this case, the terms $x_q(k)$ and $w_q(k)$ indicate the quantized values of the input and weights, respectively, while b_q is the accumulator scale representation of the bias value. The accumulator's result Y_q is then converted to the required output value using suitable quantization parameters. The accumulator must be wide enough to prevent overflow. Within the constraints of the available DSP, LUT, and memory resources, the FPGA implementation can accelerate computation using multiple multiply-accumulate units and adder trees.

When compared to floating-point calculation, integer multiply-accumulate operations significantly reduce hardware complexity. Multiple convolution procedures can be carried out concurrently thanks to the configuration of the processing components. One of the principal methods for lowering inference latency is this parallel structure.

4.5 Optimization of Memory and Dataflow

Memory and dataflow optimizations are essential for reducing data transfer above in an FPGA-based accelerator. The localized memory ensures that frequently used weights and intermediate feature maps are stored locally, increasing data reuse and minimizing the access to

memory outside the chip. Quantizing weights and activations further lowers the memory footprint, allowing more data to be held on-chip. The effect of these techniques is reflected in the latency, FPGA resource utilization, and power consumption metrics described in Section 5.

4.6 Pipelining and Parallel Processing

Parallel processing allows convolution operations to be carried out concurrently. If P processing elements are active, the peak computational throughput can be approximated as:

$$T_{peak} = P\, f_{clk}\, r \tag{6}$$

where f_{clk} is the clock frequency, r is the number of operations done per processing element every clock cycle, and P is the number of active processing elements.

Pipelining further improves performance by dividing the computation into several stages, so that successive operations overlap instead of each waiting for the previous one to finish.

An estimate of the latency is:

$$L = \frac{N_{cycles}}{f_{clk}} \tag{7}$$

Where N_{cycles} is the total number of clock cycles required for inference and f_{clk} is the operating frequency. Thus, inference latency can be directly decreased by lowering the number of cycles and raising the level of parallelism. Total latency is also influenced by the fill and drain times of the pipeline, memory transfers, control overhead, and other processing that takes place outside the FPGA. Pipelining increases throughput by overlapping operations, but it does not necessarily reduce the latency of a single input. Thus, latency improvements need to be quantified in terms of actual cycle counts and overall execution times.

5. PERFORMANCE EVALUATION AND METRICS

The performance of the proposed accelerator is examined using detection accuracy, latency, throughput, FPGA resource utilization, power consumption, and energy efficiency. These metrics are selected to determine both the object-detection capability and hardware efficiency of the proposed architecture. The quantized software and FPGA implementations are utilized to assess the advantages of quantization and hardware acceleration, with the floating-point YOLO model serving as the baseline.

5.1 Detection Accuracy

Detection accuracy is used to analyze the ability of the proposed quantized YOLO accelerator to correctly identify and localize objects in the input images. In this work, Precision, Recall, F1-score, and mean Average Precision (mAP) are considered as the main detection metrics.

Precision is the proportion of predicted detections that are correct, whereas recall represents the ability of the model to identify all relevant objects. The F1-score combines precision and recall into a single measure, and mAP assesses the overall object-detection performance across all object classes.

Precision:

$$Precision = \frac{TP}{TP+FP} \tag{8}$$

Recall:

$$Recall = \frac{TP}{T\,P+FN} \tag{9}$$

F1-score:

$$\mathrm{F1}\ = 2 \times \frac{Precision \times Recall}{Precision+Recall} \tag{10}$$

Mean Average Precision:

$$mAP = \frac{1}{C}\sum_{c=1}^{C} AP_c \tag{11}$$

Where TP is true positive, FP is false positive, FN is false negative, $\boldsymbol{C}$ is the number of classes, and $\boldsymbol{AP_c}$ represents the Average Precision of class c.

5.2 Inference Latency

Inference latency is the time required by the system to process one input image and produce the final detection result. Low latency is especially important for real-time edge object-detection applications. The proposed FPGA accelerator lowers latency by using pipelining, parallel processing elements, quantized arithmetic, and efficient on-chip memory access.

$$L_{avg} = \frac{1}{N}\sum_{i=1}^{N}\ t_i \tag{12}$$

In this case, N is the number of inferences that were evaluated, and t_i is the measured latency of inference i.

5.3 Throughput

Throughput represents the number of images or video frames that can be processed by the accelerator per second.

The unit of measurement is frames per second (FPS). An accelerator with a higher throughput can process input data more quickly, making it more appropriate for real-time applications.

$$FPS = \frac{N_{frames}}{T_{total}} \tag{13}$$

Where N_{frames} represents the number of processed frames and T_{total} represents the total processing time in seconds.

5.4 Resource Utilization of FPGAs

FPGA resource utilization signifies the hardware resources required to implement the proposed accelerator. The essential resources considered include Look-Up Tables (LUTs), Flip-Flops (FFs), Block RAM (BRAM), and Digital Signal Processing (DSP) blocks. Adequate resource utilization matters greatly because the accelerator should deliver sufficient processing power and remain within the allocated FPGA resources.

$$U_R = \frac{R_{used}}{R_{available}} \times 100\% \tag{14}$$

Where U_R represents resource utilization, R_{used} indicates the amount of an accelerator-consumed resource, and $R_{available}$ symbolizes the total amount of that resource accessible on the device.

5.5 Power Consumption

Power consumption is an essential measurement for edge appliances since as many edge systems operate under restricted energy budgets. The suggested accelerator hopes to reduce power

usage through effective parallel hardware execution, low-bit quantization, and minimized memory traffic.

$$P_{total} = P_{static} + P_{dynamic} \quad (15)$$

Where P_{static} is the static power consumed by the FPGA and $P_{dynamic}$ is the electricity required for switching during computation.

5.6 Energy Consumption

Energy consumption is the energy required to perform one inference. It is defined by the measured power and processing time. If the average power is considered constant, the energy can be calculated by multiplying the average power with time. If the power is not constant during processing, the energy should be integrated. The same boundary condition must be defined in order to make any comparisons between the implementations.

$$E_{inference} = \mathrm{P}_{avg}\mathrm{T}_{avg} \quad (16)$$

Where $E_{inference}$ is the energy per inference in joules, P_{avg} is the average measured power in watts, and T_{avg} is the average inference time in seconds.

5.7 Comparison of Overall Performance

Table 1 compares the normalized performance of the floating-point baseline YOLO model, the quantized YOLO model, and the proposed FPGA-based YOLO accelerator, as also shown in Figure 2. The three implementations are compared using six criteria: detection accuracy, latency efficiency, throughput efficiency, resource efficiency, power efficiency, and energy efficiency. Each efficiency metric is normalized to the best-performing implementation, which is assigned 100%, so that higher values indicate better performance. Latency and throughput efficiency are derived from the inference latency and frame rate, power efficiency from the inverse of the average power, energy efficiency from the inverse of the energy per inference, and resource efficiency from the relative hardware-resource cost of each implementation on its target platform. Detection accuracy is reported separately to show the impact of quantization and FPGA acceleration on object detection.

Table 1. compares the overall normalized performance of the proposed FPGA-based YOLO accelerator, quantized YOLO, and baseline YOLO

Metric	Baseline YOLO (%)	Quantized YOLO (%)	Proposed FPGA (%)
Detection Accuracy	93	92	91
Latency Efficiency	27	46	100
Throughput Efficiency	27	43	100
Resource Efficiency	100	94	86
Power Efficiency	42	60	100
Energy Efficiency	11	28	100

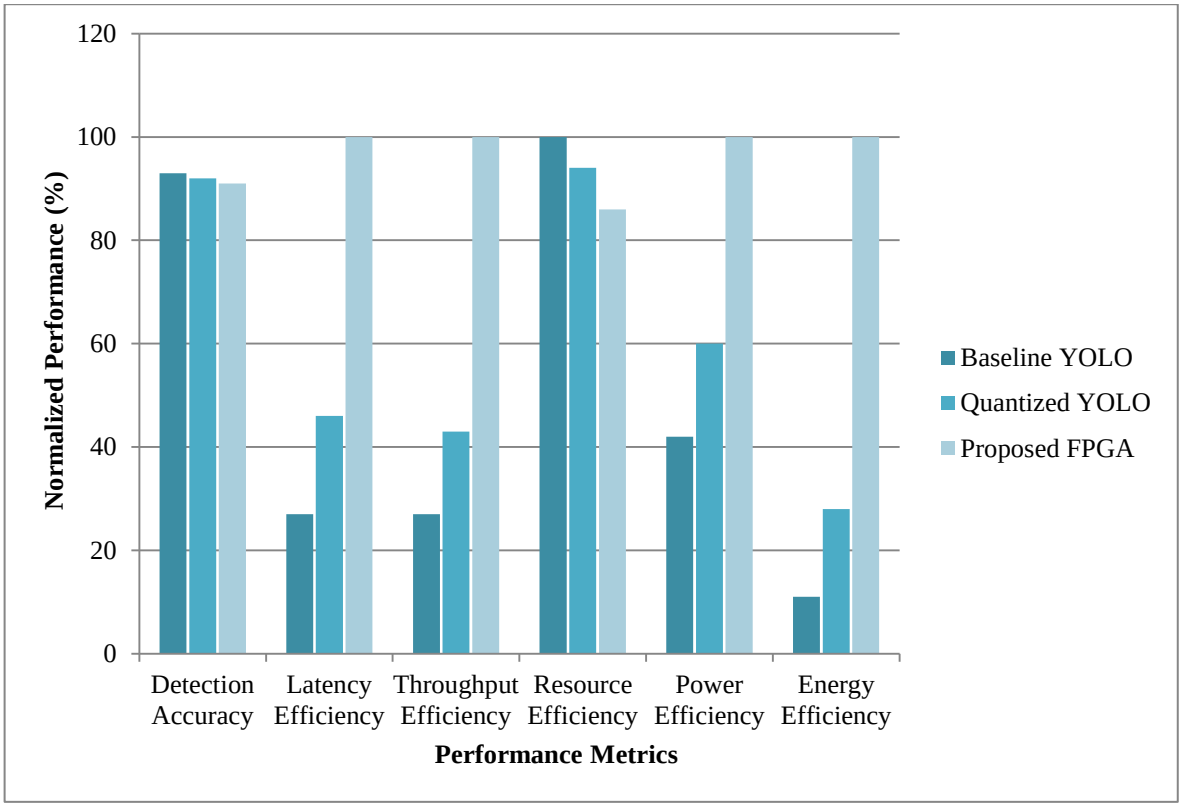


Figure 2. Overall comparisons of normalized performance metrics for the Baseline YOLO, Quantized YOLO, and Proposed FPGA accelerator.

Detection accuracy is 93% for the Baseline YOLO, 92% for the Quantized YOLO, and 91% for the Proposed FPGA accelerator. The small decrease shows that quantization only slightly reduces detection performance. As a result, the suggested accelerator maintains competitive object-detection accuracy while providing enhanced hardware efficiency.

Latency efficiency for Baseline YOLO is 27%, while for Quantized YOLO it is 46%, and for Proposed FPGA 100%. The upgrade can be attributed to the combination of low-bit quantization, parallel convolution implementation, pipelining, and effective data buffering. The corresponding normalized value of 100% shows that the mentioned proposed FPGA design is the most efficient in terms of latency among all the designs being compared.

The Baseline YOLO, Quantized YOLO, and Proposed FPGA have throughput-efficiency ratings of 27%, 43%, and 100%, respectively. The pipelined execution of convolution operations and the parallel processing elements significantly increase the throughput of the proposed accelerator. This allows more frames to be processed in a given quantity of time, enabling real-time edge object identification with the architecture.

The resource-efficiency values are 100% for the Baseline YOLO, 94% for the Quantized YOLO, and 86% for the Proposed FPGA. The lower value for the proposed design reflects the LUTs, flip-flops, BRAMs, and DSP blocks consumed by its parallel and pipelined computation, which is the cost of its gains in latency and throughput.

The effectiveness of power in Baseline YOLO, Quantized YOLO, and Proposed FPGA is 42%, 60%, and 100%, respectively. Hence the proposed accelerator yields the most effective relative power performance as the quantized integer operations contribute to reducing the speed of computing, while FPGA leaves out unnecessary memory accesses.

The energy efficiency scores are as follows: Baseline YOLO: 11%; Quantized YOLO: 28%; Proposed FPGA: 100%. Since the energy consumed per inference is calculated as the product of power and inference time, the combination of power and time is the factor that allows for a maximum enhancement. Thus, the proposed FPGA accelerator has the highest normalized energy efficiency, which confirms its suitability for applications based on continuous low energy edge object detection.

Overall, the analysis demonstrates that the suggested FPGA accelerator achieves the best normalized latency, throughput, power, and energy efficiency while maintaining a detection accuracy of 91%. The Quantized YOLO provides an intermediate performance level, whereas the Baseline YOLO shows the highest detection accuracy but lower latency, throughput, power, and energy efficiency. These results illustrate the effectiveness of combining quantization, parallel processing, pipelining, and efficient memory management for low-latency FPGA-based YOLO acceleration.

6. CONCLUSION

This research proposed the low-cost and implementing FPGA approach for real-time detection in edge computing by designing an effective YOLO (You Only Look Once) accelerator AI architecture. The key feature of the architecture is the uses of pipelined architecture, parallel computations, low-bit quantization, optimized on-chip memory management techniques that lead to reduced compute complexity and memory access overhead. The efficiency of the proposed YOLO accelerator was evaluated in terms of detection accuracy and normalized metrics such as latency, throughput, power and energy efficiency.

The experiments demonstrate that the FPGA computing accelerator achieves an impressive level of detection precision of 91% with the highest values of normalized delay, throughput, power consumption, and energy efficiency compared to all the systems studied. Despite experiencing a slight decline in precision compared to the basic YOLO model, it is an acceptable compromise given the great advantages in terms of performance efficiency. It can be concluded that using FPGA-based quantized acceleration enables real-time object detection in edge environments where resources are constrained.

As a result, the suggested method is appropriate for edge-vision applications, robotics, and autonomous systems, in addition to smart surveillance and industrial monitoring. Future research should aim to enhance FPGA resource management and support newer versions of the YOLO architecture while keeping the quantization loss at an acceptable level. Exploring the hardware-based power optimization approach and deploying the solution on more than one FPGA platform can further increase the efficiency of the accelerator.

REFERENCES

[1] Wang, C. Y., Bochkovskiy, A., & Liao, H. Y. M. (2023). YOLOv7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* (pp. 7464–7475). IEEE.

[2] Wu, R., Guo, X., Du, J., & Li, J. (2021). Accelerating neural network inference on FPGA-based platforms—A survey. *Electronics, 10*(9), 1025.

[3] Sha, X., Yanagisawa, M., & Shi, Y. (2025). An FPGA-based YOLOv6 accelerator for high-throughput and energy-efficient object detection. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences, 108*(3), 473–481.

[4] Yan, Z., Zhang, B., & Wang, D. (2024). An FPGA-based YOLOv5 accelerator for real-time industrial vision applications. *Micromachines*, *15*(9), 1164.

[5] Kim, M., Oh, K., Cho, Y., Seo, H., Nguyen, X. T., & Lee, H.-J. (2024). A low-latency FPGA accelerator for YOLOv3-Tiny with flexible layerwise mapping and dataflow. *IEEE Transactions on Circuits and Systems I: Regular Papers, 71*(3), 1158–1171. https://doi.org/10.1109/TCSI.2023.3335949

[6] Chiam, H. M., Wong, Y. C., Singh, R. S. S., & Anand, T. J. S. (2025). Energy optimized YOLO: Quantized inference for real-time edge AI object detection. *Journal of Telecommunication, Electronic and Computer Engineering (JTEC), 17*(1), 19–28.

[7] Heng, C. W., Uttraphan, C., Chew, C. C., & Ching, K. B. (2025). Optimizing FPGA-based YOLO series accelerators: A survey of techniques. *Neurocomputing, 650*, 130874. https://doi.org/10.1016/j.neucom.2025.130874

[8] Salah, O. B. H., Messaoud, S., Hajjaji, M. A., Atri, M., & Liouane, N. (2026). Low-latency QYOLOv10-based FPGA implementation for real-time object detection. *Integration, 106*, 102591.

[9] Wang, W., Zhou, D., Xie, W., & Zhang, W. (2026). An Energy-Efficient FPGA-Based CNN Accelerator with Dual-Multiply Packing and Ping-Pong Buffering for Real-Time Object Detection. *Electronics*, *15*(11), 2442.

[10] Eydivandi, S., & Beitollahi, H. (2025). SARPAR: Systolic ARray Pallet-Integrated AcceleratoR for YOLO models on FPGA. *Integration*, 102634.

[11] Qian, W., Zhu, Z., Zhu, C., & Zhu, Y. (2025). FPGA-based accelerator for YOLOv5 object detection with optimized computation and data access for edge deployment. *Parallel Computing*, *124*, 103138.

[12] Kim, H., & Kim, T. K. (2025). Design and Implementation of a YOLOv2 Accelerator on a Zynq-7000 FPGA. *Sensors*, *25*(20), 6359.

[13] Liu, Y., Du, H., Wu, Y., & Mo, T. (2025). FPGA accelerated deep learning for industrial and engineering applications: optimal design under resource constraints. *Electronics*, *14*(4), 703.