






FPGA-accelerated bird detection algorithm for paddy farmers

Muhammad Ili Firdaus Mohamad Sa'ad¹ , Rizalafande Che Ismail^{2,3,*} ,
Siti Zarina Md Naziri^{2,3} , Mohd Nazrin Md Isa^{2,3} ,
Ahmad Husni Mohd Shapri^{2,3} 

¹Altera Corporation, Bayan Lepas Technoplex, 11900 Penang, Malaysia.

²Centre of Excellence for Micro System Technology, Universiti Malaysia Perlis, 02600 Perlis, Malaysia.

³Faculty of Electronic Engineering & Technology, Universiti Malaysia Perlis, 02600 Perlis, Malaysia.

*Corresponding author: rizalafande@unimap.edu.my

Original Research

Received:
5 May 2025
Revised:
28 June 2025
Accepted:
14 July 2025
Published online:
4 August 2025
Published in issue:
25 September 2025

© 2025 The Author(s). Published by
the OICC Press under the terms of
the [Creative Commons Attribution
License](#), which permits use, distribu-
tion and reproduction in any medium,
provided the original work is prop-
erly cited.

Abstract:

Paddy farmers face significant challenges from bird pests, particularly species such as pipits and sparrows, which can reduce yields by up to 70%, especially during the grain-filling stage, leading to substantial economic losses. Traditional pest control methods such as physical barriers, scare tactics, and chemical deterrents are often inefficient and labour-intensive. To address this issue, this research develops an artificial intelligence (AI)-based bird detection system to protect paddy fields. The solution involves using a Field-Programmable Gate Array (FPGA)-accelerated object detection model to accurately identify bird activity in real time. The system integrates a notification mechanism via Telegram to alert farmers immediately, enabling swift manual intervention. The research employed the DETection TRansformer with ResNet-50 backbone (DETR-ResNet50) model for its precision and high confidence in detection, running on both Central Processing Unit (CPU) and FPGA configurations to optimize performance. Results showed significant improvements in latency and frames per second (FPS) when using FPGA acceleration, demonstrating effective real-time bird detection capabilities. The system's implementation enhanced crop protection, promoted eco-friendly practices, and improved overall farming efficiency by reducing manual surveillance and providing valuable data for long-term pest management strategies. Key quantitative findings revealed that FPGA acceleration improved FPS by over 200% compared to CPU performance.

Keywords: Artificial intelligence; FPGA; Object detection model; Smart farming; Image processing

1. Introduction

Nowadays, object detection is widely used in various fields such as transportation, surveillance, and artificial intelligence (AI). Computer vision is typically employed to detect objects, which are instances of specific items within images and videos. Object detection algorithms identify the outlines, locations, dimensions, and characteristics of objects in an image for subsequent identification, categorization, and monitoring.

Accurate object detection can segment and contour the objects contained in images. Commonly used object detection algorithms often rely on machine learning or deep learning to yield meaningful results, enabling them to recognize and classify objects within images or videos after being trained on large datasets. Object detection has various applications,

including crime intention detection, where it can identify individuals carrying firearms and alert authorities before a crime occurs [2].

Paddy farmers face significant challenges from bird pests, especially species like pipits and sparrows, which can cause severe crop damage during the grain-filling stage. These birds can reduce yields by up to 70%, leading to substantial economic losses [3]. Most farmers do not have the time or resources to monitor their fields continuously, creating gaps in protection and making it difficult to prevent damage. This challenge highlights the need for innovative solutions that can monitor and deter threats in real time. Traditional methods such as scarecrows or periodic field checks are no longer sufficient, particularly for larger fields or during peak harvest seasons. This underscores the importance of

automated systems, such as AI-based bird detection using Field-Programmable Gate Array (FPGA) technology, which offer the potential to bridge these gaps in protection while optimizing farmers' time and resources.

This paper proposes an AI-based solution that utilizes object detection technology to help farmers protect their paddy fields. The system detects bird activity through continuous monitoring and sends alerts when a specific number of birds are identified. Upon detection, it transmits images and notifications to farmers via Telegram, allowing for immediate action to deter the birds. This reduces the need for constant manual surveillance and promotes sustainable farming practices. Additionally, by providing data on bird activity patterns, the system enables farmers to develop more effective long-term pest control strategies. The main objectives of this research are to enhance crop protection, encourage eco-friendly practices, and improve overall farm efficiency resulting in a more effective and sustainable approach to managing bird pests in rice fields.

2. Literature review

Object detection has become a crucial area of research within the fields of computer vision and artificial intelligence, driving advancements in applications such as autonomous driving, surveillance, and the Internet of Things (IoT). This review examines the development and performance of three prominent object detection algorithms: You Only Look Once (YOLO), Detection Transformer (DETR), and CenterNet (Center Network).

2.1 Integration of FPGA-accelerated object detection

FPGAs have emerged as a compelling solution for accelerating object detection algorithms due to their parallel processing capabilities and reconfigurability. The integration of FPGAs with CPUs in object detection tasks involves a co-design approach, where computationally intensive operations are offloaded to the FPGA, while the CPU manages control flow, preprocessing, and postprocessing tasks [4]. In a typical setup, the FPGA is responsible for accelerating the core computations of object detection algorithms, such as convolutional neural network (CNN) layers, which are resource-intensive [1, 5]. To optimize the selected models for FPGA execution, the Intel OpenVINO toolkit was em-

ployed. The CPU handles tasks such as image acquisition, initial preprocessing, and postprocessing. During preprocessing, image resizing ensures uniform input dimensions, normalization scales pixel values to a standard range, and data augmentation enhances model robustness by introducing variations in the training data.

After inference, postprocessing refines the detection outputs, including bounding box refinement, confidence thresholding, and non-maximum suppression (NMS). NMS plays a crucial role in eliminating redundant overlapping boxes by selecting the bounding box with the highest confidence score and suppressing others with significant overlap, thereby reducing false positives. This division of labor ensures that each component operates within its optimal performance envelope, leading to enhanced overall system efficiency.

Communication between the CPU and FPGA is facilitated through high-speed interfaces, ensuring low-latency data transfer [6]. Synchronization mechanisms are essential to maintain data coherence and ensure seamless pipeline operation. For instance, in real-time object detection systems, the CPU captures and preprocesses image frames, which are then dispatched to the FPGA for accelerated inference. The results are subsequently retrieved by the CPU for further processing and decision-making.

To illustrate this workflow, Fig. 1 depicts the process of integrating object detection algorithms into an FPGA-accelerated pipeline. It highlights the critical steps of requirement analysis, hardware optimization, and evaluation metrics, which are essential for the efficient co-design and deployment of FPGA-accelerated systems. The figure also aligns with object detection models like YOLO and DETR, where FPGA acceleration optimizes the execution of convolutional layers or transformer operations to achieve improved speed and throughput.

It is also shown that the typical architecture of an FPGA-based object detection system involves a division of computational tasks between the CPU and FPGA [7]. The CPU handles preprocessing tasks, such as resizing and normalization, before forwarding the data to the FPGA for accelerated inference. Postprocessing operations, including bounding box refinement and confidence score calculation, are also managed by the CPU. This pipeline ensures efficient utiliza-

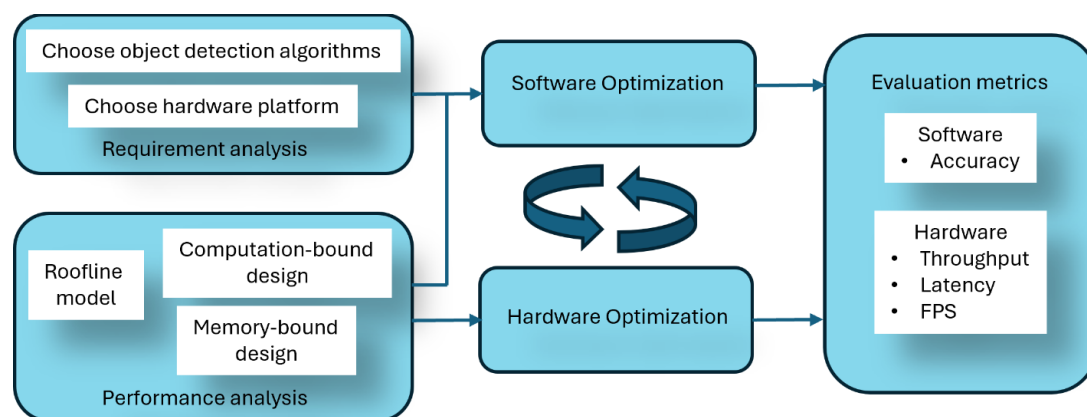


Figure 1. The design flow of FPGA-based object detection architecture [1].

tion of resources, reducing latency and enhancing throughput, as highlighted in previous studies.

Recent research has demonstrated the efficacy of this integration. For example, an FPGA-based accelerator for object detection was shown to significantly enhance performance by offloading computationally intensive tasks from the CPU to the FPGA, resulting in improved throughput and reduced latency [9]. Another study highlighted the benefits of FPGA acceleration in real-time object detection and classification systems, emphasizing the importance of efficient integration between hardware and software components [10].

2.2 Comparison of object detection models

Object detection models differ in their architectures and computational efficiencies, directly influencing their suitability for FPGA acceleration. The three models selected for this study were chosen based on their architectural compatibility with FPGA implementation, balancing accuracy, computational efficiency, and resource utilization.

- **detr-resnet50**: This transformer-based object detection model eliminates the need for anchor boxes and non-maximum suppression (NMS), enabling end-to-end detection. While DETR's innovative architecture offers high accuracy, its complexity and computational demands present challenges for FPGA deployment. Nevertheless, its potential for precise detection makes it a promising candidate for FPGA exploration [8].
- **yolov4-tf**: As a one-stage detection framework optimized for both speed and accuracy, YOLOv4 is well-suited for real-time applications on resource-constrained hardware. Its design emphasizes efficient computation, aligning with the parallel processing capabilities of FPGAs. Previous studies have demonstrated its adaptability to FPGA platforms, highlighting its balance between performance and resource utilization [11].
- **ctdet_coco_dlav0_512**: This keypoint-based approach detects object center points, reducing computational complexity while maintaining accuracy. CenterNet's methodology simplifies the detection process, which is advantageous for FPGA implementation. Its reduced computational requirements allow for efficient mapping onto FPGA architectures, facilitating real-time performance [11].

Alternative models such as YOLOv5 and YOLOv7 are widely used in embedded AI applications due to their im-

proved efficiency and accuracy. However, they were excluded from this study due to the lack of mature, optimized FPGA implementations and the focus on models with well-documented FPGA compatibility. While emerging efforts aim to adapt YOLOv5 for FPGA platforms—such as a novel 4-bit quantization-based neural network accelerator designed to enhance real-time processing—these implementations remain under development and are not yet standardized, making them unsuitable for the objectives of this study [12]. Similarly, although FPGA implementations of YOLOv7 have been reported, the performance trade-offs and resource utilization require further investigation to ensure compatibility with the specific requirements of this work [13].

FPGAs are increasingly favored for object detection due to their unparalleled parallel processing capabilities, low latency, and power efficiency. Unlike CPUs and GPUs, FPGAs can be dynamically reconfigured to optimize specific computational tasks—such as convolutional operations in deep learning models—making them ideal for real-time applications [6, 14]. Notable algorithms like **yolov4-tf**, **ctdet_coco_dlav0_512**, and **detr-resnet50** have already been successfully adapted for FPGA platforms, meeting the demands of high-performance, low-power object detection systems.

DETR, in particular, stands out with its end-to-end object detection framework. It employs a transformer-based encoder-decoder architecture combined with a ResNet-50 backbone, as shown in Fig. 2, to process global relationships between objects. Unlike traditional detectors, DETR removes the need for post-processing steps such as NMS and anchor generation by leveraging a bipartite matching loss. This direct set prediction approach simplifies the detection pipeline while maintaining competitive accuracy. Although DETR demonstrates robust performance for large objects, it faces challenges in detecting small objects—a limitation that can be mitigated through further optimization [8].

The **ctdet_coco_dlav0_512** model, as illustrated in Fig. 3, adopts a center-point detection approach in which objects are represented as keypoints. It employs a fully convolutional network to generate heatmaps for object centers, making it particularly efficient for small-scale applications. While its computational efficiency is notable, its performance can vary depending on object density and size within the input images. Owing to its lightweight design, this model is a strong candidate for real-time applications on FPGA platforms [8].

The **yolov4-tf** model is another prominent algorithm

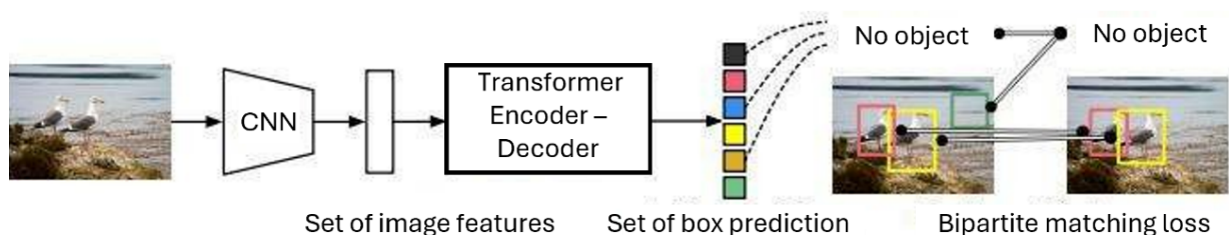


Figure 2. Diagram of **detr-resnet50** Architecture (encoder-decoder with resnet50 backbone) [8].

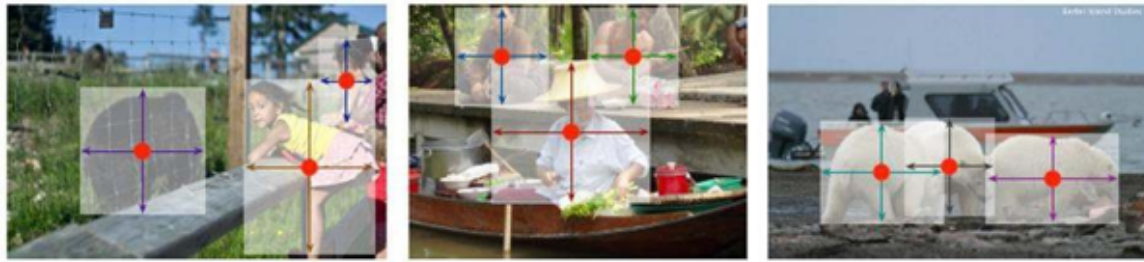


Figure 3. Example of `ctdet_coco_dlav0.512` keypoint detection process [11].

adapted for FPGA systems, focusing on multi-scale detection as illustrated in Fig. 4. Renowned for its balance between speed and accuracy, it is well-suited for real-time applications. **YOLOv4-tf** incorporates an improved feature pyramid network (FPN) and path aggregation network (PAN) to enhance detection performance across different object scales. Its straightforward architecture and high throughput make it a popular choice for FPGA deployment, particularly in scenarios requiring real-time detection of multiple object types [9, 15].

FPGAs excel not only because of their adaptability but also due to their energy efficiency, a critical advantage in resource-constrained environments. A 2022 survey by Zeng et al. highlights the suitability of FPGAs for object detection, particularly in edge computing applications where low power consumption and real-time performance are paramount [9]. These findings underscore the growing importance of FPGA-based solutions in developing advanced and efficient object detection systems.

The synergy between CPUs and FPGAs plays a pivotal role in achieving optimal performance. CPUs typically manage high-level orchestration tasks, such as system control and coordination, while FPGAs handle computationally intensive processes like feature extraction, convolutional operations, and bounding box calculations. By offloading these tasks, FPGAs eliminate bottlenecks that can hinder a system's real-time capabilities. Further optimizations, such as quantization (reducing model precision without significantly compromising accuracy) and loop unrolling (minimizing iteration overhead), enhance computational efficiency. Studies have shown that FPGA platforms such as the Zynq UltraScale + MPSoC can achieve frame rates exceeding 60 FPS while maintaining remarkably low power consumption, making them ideal for both standalone and integrated applications.

In agricultural applications, FPGAs often serve as primary accelerators for processing high-throughput video streams

and real-time object detection tasks. For instance, FPGA-based bird detection systems can process high-resolution images in under 50 milliseconds, enabling real-time detection and immediate alerts via communication platforms like Telegram. Such systems allow farmers to respond rapidly to potential crop threats, reducing damage and promoting sustainable agricultural practices. Moreover, these FPGA solutions provide eco-friendly alternatives to chemical deterrents by offering non-invasive, data-driven pest control methods [15].

The applicability of FPGA-based systems extends well beyond agriculture. They have been successfully deployed in edge computing scenarios for surveillance, autonomous navigation, and industrial monitoring. Zeng et al. notably demonstrated the flexibility and scalability of FPGA-based solutions, showcasing their ability to adapt to evolving detection requirements across diverse industries [9]. These innovations position FPGAs not only as co-processors but also as standalone accelerators capable of transforming real-time systems through their reconfigurability, high performance, and energy efficiency.

This review highlights how FPGA-accelerated systems address the computational challenges of real-time object detection while providing scalable and energy-efficient solutions. By emphasizing their transformative potential in agricultural and other real-world applications, this section establishes a solid foundation for the methodology and system design discussed in the following sections.

3. Methodology

This study aimed to develop an AI-based bird detection system using FPGA technology to address the real-time detection requirements of agricultural applications. The methodology followed a systematic process involving iterative stages of software and hardware implementation, optimization, and testing to maximize system efficiency. The development process, illustrated in Fig. 5, begins with data

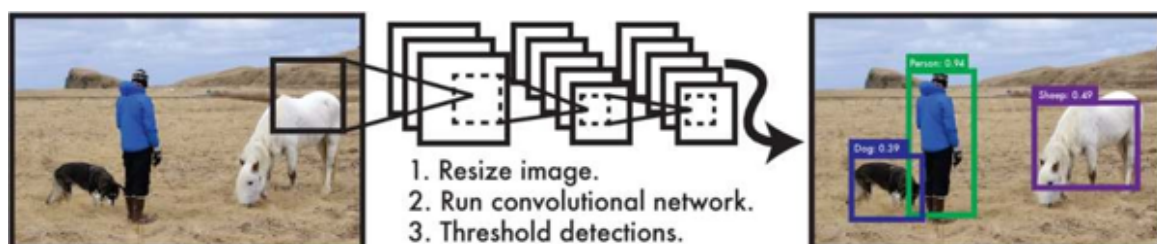


Figure 4. The `yolov4-tf` architecture focusing on multi-scale detection [14].

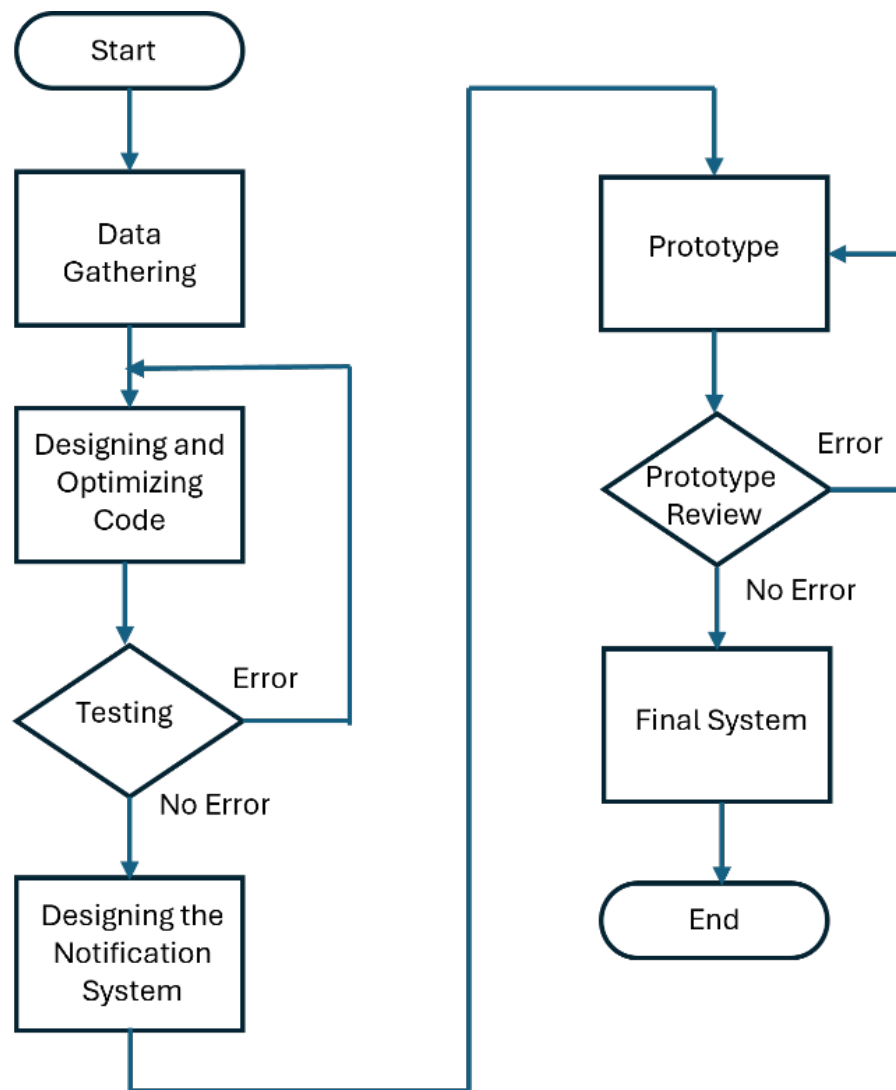


Figure 5. Flowchart of the development process.

acquisition, where relevant datasets of birds in paddy fields were collected. A smart camera system was deployed in a controlled environment replicating paddy field conditions. Video footage of birds was captured at a fixed distance of 10 meters to simulate real-world scenarios, with specific markers placed to ensure accurate distance measurements during testing. This allowed precise evaluation of detection sensitivity and accuracy. The video resolution was set to 720×480 pixels, and recordings were made at 60 frames per second (FPS), ensuring high-quality inputs for model assessment. These datasets were used for both training and evaluating the object detection models.

Following data acquisition, the model development and optimization phase was conducted. Pre-trained object detection models **yolov4-tf**, **detr-resnet50**, and **ct-det.coco_dlav0.512** were selected based on their suitability for real-time applications. These models were optimized using the Intel OpenVINO toolkit to ensure compatibility with the FPGA platform and to enhance inference efficiency. One of the critical steps in this methodology was the modification of the Python scripts and the development of a new inference core capable of operating in a hybrid HETERO

configuration (FPGA + CPU). The original OpenVINO demo code was designed to run exclusively on CPUs, limiting the ability to exploit the parallel processing power and speed of the FPGA. To overcome this limitation, a custom inference core was developed and integrated into the object detection pipeline, as illustrated in Fig. 6. This core enabled dynamic workload distribution between the FPGA and CPU, significantly reducing latency and improving overall system performance. By allowing computationally intensive tasks such as neural network inference to execute in parallel on the FPGA, the system fully leveraged the combined capabilities of both hardware components.

The decision to base the implementation on OpenVINO's pre-existing code, rather than developing an AI model from scratch, was both strategic and practical. Training a new AI model from the ground up would have required vast computational resources, extensive datasets, and significant time to ensure accuracy and robustness. In contrast, the OpenVINO toolkit provided pre-trained models already optimized for Intel hardware, offering a proven foundation for real-time object detection. This approach reduced development time and enabled the project to focus on integrating

```

if args.adapter == 'openvino':
    dla_plugins_xml_file = os.environ.get('DLA_PLUGINS_XML_FILE', default='')
    plugin_config = get_user_config(args.device, args.num_streams, args.num_threads)
    model_adapter = OpenvinoAdapter(create_core_for_fpga(dla_plugins_xml_file), args.model, device=args.device, plugin_config=plugin_config,
                                   max_num_requests=args.num_infer_requests, model_parameters = {'input_layouts': args.layout})
elif args.adapter == 'ovms':
    model_adapter = OVMSAdapter(args.model)

```

Figure 6. Modified Core for FPGA.

and enhancing these models for the specific application of bird detection in paddy fields. Furthermore, OpenVINO includes tools for fine-tuning models for different hardware configurations, ensuring they are not only functional but also highly efficient when deployed on FPGA. Leveraging this toolkit represented a logical balance between resource efficiency and achieving high-performance results. Once the models were prepared and the custom inference core developed, the Python scripts were modified to support FPGA-based inference. This optimization involved configuring several key parameters such as nireq, nstreams, and nthreads, to balance performance, latency, and hardware utilization:

- **nireq (Number of Inference Requests) = 1:** Real-time responsiveness is critical for timely alerts; therefore, single-frame processing was prioritized to avoid delays caused by queuing multiple inference requests. The FPGA's computational resources are optimized for processing one task efficiently, rather than splitting capacity across multiple simultaneous tasks, which could degrade performance.
- **nstreams (Number of Streams) = 6:** This value was determined based on the FPGA's ability to handle multiple streams without introducing CPU bottlenecks. Six streams allow the system to process data from multiple sources (e.g., live camera feeds) concurrently, maximizing hardware utilization while maintaining smooth operation under varying workloads.
- **nthreads (Number of Threads) = 4:** Modern CPUs often include multiple cores, and allocating four threads ensures adequate processing power for lightweight yet frequent tasks such as data transfer between FPGA and CPU, image resizing, and alert handling. This configuration supports smooth multitasking while leaving sufficient cores available for other processes. Careful testing confirmed that this setting avoids excessive context switching, which could otherwise degrade performance through unnecessary overhead.

Fig. 7 shows the command and arguments used for executing object detection with inference on the hybrid HETERO configuration. The final parameter values were determined experimentally, based on tests that balanced latency, frame rate, and system stability.

System integration followed, combining the object detection models with a Telegram bot for real-time notifications. As shown in Fig. 8, the bot sends annotated images of detected birds directly to the user, enabling farmers to take immediate action without the need for continuous manual field monitoring. The bot was programmed to operate seamlessly with the FPGA processing pipeline, ensuring minimal latency in communication. To prevent interference with the primary detection pipeline, the notification process was implemented using multi-threading, allowing detection and alert transmission to run concurrently without performance degradation.

```

python3 fpga_object_detection_demo.py \
    -m <path_to_model>/detr-resnet50.xml" \
    -at detr \
    --label <omz_dir>/data/dataset_classes/coco_91cl_bkgr.txt" \
    -i <path_to_video>/inputVideo.mp4 \
    -d HETERO:FPGA,CPU \
    --output_resolution 720x480 \
    -nireq 1 \
    -nstreams 6 \
    -nthreads 4 \
    --no_show \
    -r \

```

Figure 7. Command and arguments for object detection.



Figure 8. Example of a notification sent via the “Bird Detection” Telegram bot.

The annotated images sent via the Telegram bot include bounding boxes to highlight detected birds, providing clear and actionable insights to the user. This integration minimizes the need for physical presence in the fields and improves overall operational efficiency. Fig. 8 presents an example of notifications received through the Telegram bot during field testing.

The end-to-end delay of the system was measured by recording timestamps at three key points:

- **Frame Capture Time** – the moment the camera captured the frame.
- **Inference Completion Time** – when the FPGA completed processing and generated detection results.
- **Notification Sent Time** – when the Telegram notification was successfully delivered.

The delay was calculated using the following formula:

$$\text{Delay (ms)} = \text{Notification Sent Time} - \text{Frame Capture Time}$$

The final step of the process, as outlined in the flowchart, was prototype development and evaluation. A smart camera system was deployed in paddy fields to capture real-time footage, which was processed using the optimized DETR-ResNet50 model running on the FPGA. Rigorous testing was conducted to evaluate key performance metrics, including latency, confidence, and frame rate. The final implementation demonstrated notable improvements: latency decreased from 1429.6 ms (CPU-only) to 466.1 ms, while

the frame rate increased from 0.7 FPS to 2.1 FPS using the HETERO (FPGA + CPU) configuration.

The hardware implementation was carried out on the Intel Arria 10 FPGA, selected for its flexibility and high-performance capabilities. As shown in Fig. 9, the FPGA provided the necessary hardware platform for high-speed object detection. It was programmed using Intel Quartus Prime software, enabling custom design implementation and real-time performance optimization.

The software implementation was carried out using Python in combination with frameworks such as TensorFlow, PyTorch, and Keras. OpenVINO served as the primary optimization toolkit, enabling efficient execution of the object detection models on the FPGA. Visual Studio Code (VS Code) was employed as the integrated development environment (IDE), providing robust debugging and code enhancement capabilities. The Intel Arria 10 FPGA was programmed using Intel Quartus Prime software, delivering the high-speed hardware platform required for real-time object detection.

This detailed methodology illustrates a seamless progression from data acquisition and model optimization to system integration and performance evaluation. The iterative approach combined with strategic decisions such as leveraging pre-trained models and developing a custom optimized inference core enabled the system to achieve real-time detection with high accuracy and low latency. This practical and resource-efficient design resulted in a robust solution for effectively mitigating bird pest issues in agricultural environments.



Figure 9. Intel Arria 10 FPGA [16].

4. Results and discussion

The results presented in Table 1 reveal clear variations in latency, FPS, and reliability across the three evaluated models **detr-resnet50**, **yolov4-tf**, and **ctdet_coco_dlav0_512** under both CPU and FPGA configurations. A closer analysis of these results provides insight into the underlying factors contributing to these differences and underscores the advantages of FPGA acceleration in real-time object detection systems. These performance outcomes are directly linked to the methodological choices made during development and optimization, particularly the decision to run all three models using identical parameter settings for a fair and consistent comparison.

The **detr-resnet50** model demonstrated the most substantial improvement when transitioning from CPU to FPGA execution. Latency decreased from 1298.3 ms to 334 ms, while FPS increased from 3.1 to 11.5. Although all models were tested under identical parameter settings, **detr-resnet50**'s architecture explains its dramatic performance gains. DETR employs a transformer-based encoder–decoder mechanism that demands considerable computational resources for global attention and object matching. On the CPU, the sequential nature of these operations increases latency; in contrast, the FPGA's parallel processing capabilities substantially reduce execution time by efficiently distributing these computations. The ResNet-50 backbone, with 25.53 million parameters and requiring 8.2164 GFLOPs, adds to the model's computational intensity [17]. The FPGA's ability to accelerate convolutional operations was critical in

making **detr-resnet50** both the most resource-intensive and the most improved model in this study.

One notable advantage of FPGA-based inference over CPU processing is power efficiency. Prior studies indicate that, for complex vision pipelines, FPGAs can achieve energy/frame reductions between 1.2× and 22.3× compared to CPUs, reflecting significant efficiency gains [18]. However, FPGAs may exhibit higher static power consumption, potentially limiting their suitability for ultra-low-power applications. This study did not include precise power measurements; future work should incorporate detailed power profiling to quantify the energy efficiency of FPGA-based agricultural systems [19].

The reliability of **detr-resnet50** was evaluated under diverse field scenarios. Upon successful bird detection, the system captures the scene, processes the image, and sends a Telegram notification to the farmer, containing detection time, a snapshot of the bird, and the number of birds detected. This real-time functionality allows for prompt responses to potential threats, with average delays ranging between 1.1 s and 1.3 s depending on the model. Detr-resnet50 consistently achieved accurate detections within a 10 m range but showed slightly reduced accuracy for smaller birds at greater distances, likely due to the lower-resolution inputs challenging transformer-based architectures. The 1.3 s notification delay was primarily attributed to the attention computation overhead and the time required for data transmission via Telegram.

In contrast, **yolov4-tf** exhibited minimal performance improvement on FPGA, with latency decreasing from 949.5

Table 1. Comparison between CPU and HETERO.

Model	Latency (ms)		FPS	
	CPU	HETERO:FPGA, CPU	CPU	HETERO:FPGA, CPU
detr-resnet50	1298.3	334	3.1	11.5
yolo-v4-tf	5787.7	609.6	4.2	3.6
ctdet_coco_dlav0_512	949.5	1116	6	11.8

ms to 609.6 ms, while FPS dropped slightly from 4.2 to 3.6. This model uses a single-shot, grid-based detection method with anchor boxes at multiple scales. While computationally efficient, **yolov4-tf**’s 129.56 GFLOPs and 64.33 million parameters make it more demanding than earlier YOLO versions [20]. Under the tested conditions, **yolov4-tf** failed to reliably detect birds within 10 meters, likely due to its reliance on anchor boxes and difficulty handling small objects. Its 1.1 s notification delay reflected efficient result transmission but could not offset its detection accuracy limitations.

The **ctdet_coco_dlav0_512** model achieved balanced performance gains on FPGA, with latency reduced from 578.7 ms to 111.6 ms, and FPS increasing from 6.0 to 11.8. CTDet uses a keypoint-based approach, representing objects as center points and generating heatmaps to predict positions. Bounding box size and placement are regressed from these keypoints. Although lighter than DETR and **yolov4-tf**, CTDet still processes 17.91 million parameters and requires 62.21 GFLOPs [21]. Like **yolov4-tf**, CTDet struggled to detect birds within 10 meters, likely due to sensitivity to occlusions and small-object detection challenges. The notification delay was 1.2 s, and the model maintained stable performance in other scenarios, but lacked the close-range accuracy required for this application.

A visual comparison of the models’ detection accuracy is shown in Table 2, Fig. 10, Fig. 11, and Fig. 12. These results clearly indicate that **detr-resnet50** successfully de-

tected birds within 10 meters, while **yolov4-tf** and **ctdet_coco_dlav0_512** failed under identical conditions. This underscores the superior accuracy and robustness of **detr-resnet50**, particularly in challenging, real-world agricultural environments.

The performance variations among the three models stem from their distinct architectural designs and the degree to which their operations align with FPGA’s parallel processing capabilities. The **detr-resnet50** model showed the most pronounced improvement due to its computationally intensive transformer architecture and ResNet-50 backbone, which leveraged parallel execution efficiently. **yolov4-tf**, designed for single-shot detection, exhibited limited acceleration gains because of its inherently efficient design and underperformed for close-range detection tasks. **ctdet_coco_dlav0_512** offered a balance between complexity and performance, benefiting moderately from FPGA’s ability to process heatmap generation and regression tasks concurrently, but struggled with detecting smaller, nearby objects.

Reliability metrics including sensitivity, notification delay, and system uptime further reinforce the real-world applicability of these findings. The **detr-resnet50** emerged as the optimal choice for scenarios demanding precise detection within a 10-meter range, offering both high accuracy and substantial FPGA optimization benefits. The **yolov4-tf** performed better for long-range detection but lacked dependability at shorter distances, while CTDet provided balanced,

Table 2. Comparative performance and detection accuracy of models.

Model	Latency (ms) (CPU)	Latency (ms) (FPGA)	FPS (CPU)	FPS (FPGA)	Notification Delay (s)	Detection accuracy	Sensitivity range
detr-resnet50	1298.3	334	3.1	11.5	1.3	Detected target	Up to 10 meters
yolo-v4-tf	949.5	609.6	4.2	3.6	1.1	Missed target	Failed within 10 meters
ctdet_coco_dlav0_512	578.7	111.6	6.0	11.8	1.2	Missed target	Failed within 10 meters



Figure 10. Birds’ detection using **detr-resnet50** model.



Figure 11. Birds' detection using **ctdet.coco.dlav0.512** model. Birds' detection using **ctdet.coco.dlav0.512** model.



Figure 12. Birds' detection using **yolo-v4-tf**.

medium-range detection capabilities but was hindered by object occlusions. Overall, these results underscore the importance of hardware–software co-optimization in achieving real-time object detection for agricultural applications such as bird monitoring in paddy fields.

5. Conclusion

This project successfully developed a real-time bird detection system leveraging FPGA-accelerated object detection models to safeguard paddy fields from bird pests. Three models named **detr-resnet50**, **ctdet.coco.dlav0.512**, and **yolo-v4-tf** were evaluated based on latency, FPS, and detection confidence. Quantitative analysis revealed that **detr-resnet50** achieved the best performance within a 10-meter detection range, reducing inference latency from 1298.3 ms (CPU-only) to 334 ms (FPGA) and increasing FPS from 3.1 to 11.5 representing an improvement of over 200% compared to CPU performance. In contrast,

yolo-v4-tf and **ctdet.coco.dlav0.512** exhibited reduced accuracy in close-range detection, further emphasizing **detr-resnet50** as the most suitable model for this application. To maintain high performance under concurrent operations, multi-threading was implemented, ensuring that tasks such as detection and notification ran without bottlenecks. A Telegram bot notification system delivered annotated images to farmers with an average delay of just 1.3 seconds, substantially reducing the need for constant manual field monitoring. By combining AI-driven object detection with FPGA acceleration, this system not only enhanced crop protection but also promoted eco-friendly pest control and improved operational efficiency. The solution provides a scalable and robust framework for real-time pest management while generating valuable data for long-term agricultural strategies, demonstrating the transformative potential of AI–FPGA integration in precision agriculture.

Authors contributions

Authors have contributed equally in preparing and writing the manuscript.

Availability of data and materials

The data that support the findings of this study are available from the corresponding author upon reasonable request.

Conflict of interests

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

- [1] Y. Wang, Y. Liao, Y. Yang, J. Wang, and H. C Zhao. "An FPGA-based Online Reconfigurable CNN Edge Computing Device for Object Detection.". *Microelectronics J*, 137:105805, 2023. DOI: <https://doi.org/10.1016/j.mejo.2023.105805>.
- [2] A. Osman Hashi, A. A. Abdirahman, M. Abdirahman Elmi, O. Ernest, and R. Rodriguez. "Deep Learning Models for Crime Intention Detection Using Object Detection.". *Int. J. Adv. Comput. Sci. Appl.*, 14(4):1–20, 2023. DOI: <https://doi.org/10.14569/ijacsa.2023.0140434>.
- [3] B. Razali. "Pengurusan burung perosak di sawah padi (Management of Bird Pests in Rice Field)". *Buletin Teknologi MARDI Bil*, 34:29–44, 2022.
- [4] Z. Li. "Investigation of Reconfigurable Hardware Acceleration for Low-Power Embedded Neural Networks.". Université Côte d'Azur, Theses De Doctorat, 2024.
- [5] C. Vasile and A. Ulm. "Image Processing Hardware Acceleration — A Review of Operations Involved and Current Hardware Approaches.". *Journal of Imaging*, 10(12):298, 2011. DOI: <https://doi.org/10.3390/jimaging10120298>.
- [6] A. D. Santi. "Reducing Latency of Object Detection Systems Using Bayer Filters Reducing Latency of Object Detection Systems Using Bayer Filters.". *KTH, School of Electrical Engineering and Computer Science Dissertation (Master)*, 2024.
- [7] P. Xiyuan, Y. Jinxiang, Y. Bowen, and L. Liansheng. "A Review of FPGA-Based Custom Computing Architecture for Convolutional Neural Network Inference.". *Chinese Journal of Electronics*, 30(1):1–17, 2021. DOI: <https://doi.org/10.1049/cje.2020.11.002>.
- [8] N. Carion, F. Massa, G. Synnaeve, N. Usunier, A. Kirillov, and S. Zagoruyko. "End- to-End Object Detection with Transformers.". *European Conference on Computer Vision*, pages 213–229, 2020. DOI: <https://doi.org/10.1109/access.2022.3208889>.
- [9] K. Zeng, Q. Ma, J. W. Wu, Z. Chen, T. Shen, and C. Yan. "FPGA-based Accelerator for Object Detection: A Comprehensive Survey.". *Journal of Supercomputing*, 78(12):14096–14136, 2022. DOI: <https://doi.org/10.1007/s11227-022-04415-5>.
- [10] A. Montgomerie-Corcoran, P. Toupas, Z. Yu, and C. S. Bouganis. "SATAY: A Streaming Architecture Toolflow for Accelerating YOLO Models on FPGA Devices.". *Proceedings-International Conference on Field-Programmable Technology, ICFPT*, page 179–187, 2023. DOI: <https://doi.org/10.1109/icfpt59805.2023.00025>.
- [11] X. Zhou, D. Wang, and P. Krähenbühl. "Objects as Points.". *arXiv preprint arXiv:1904.07850*, 2019.
- [12] Z. Yan, B. Zhang, and D. Wang. "An FPGA-Based YOLOv5 Accelerator for Real-Time Industrial Vision Applications.". *Micromachines (Basel)*, 15(9):1164, 2024. DOI: <https://doi.org/10.3390/mi15091164>.
- [13] X. Li, Y. Wei, J. Li, W. Duan, X. Zhang, and Y. Huang. "Improved YOLOv7 Algorithm for Small Object Detection in Unmanned Aerial Vehicle Image Scenarios.". *Applied Sciences*, 14(4):1664, 2024. DOI: <https://doi.org/10.3390/app14041664>.
- [14] A. Bochkovskiy, C.-Y. Wang, and H.-Y. M. Liao. "YOLOv4: Optimal Speed and Accuracy of Object Detection.". *arXiv preprint arXiv:2004.10934*, 2020.
- [15] Peraka, Shyam, R. Sudheer, B. Narasimha Rao, A. R. Teja, and E. N. Kumar. "Smart irrigation based on crops using IoT.". *15th International Conference on Industrial and Information Systems (ICIIS)*, pages 611–616, 2020. DOI: <https://doi.org/10.1109/iciis51140.2020.9342736>.
- [16] C. Robinson. "Intel Arria 10 GX FPGA Card for Servers Released.". URL <https://www.servethehome.com/intel-arria-10-gx-fpga-card-for-servers-released/>.
- [17] E. Wovchena. "detr-resnet50.". Github repository, 2022. URL https://github.com/openvinotoolkit/open_model_zoo/blob/master/models/public/detr-resnet50/README.md.
- [18] Q. Murad, K. Denolf, J. Lo, K. Vissers, J. Zambreno, and P. H. Jones. "Comparing energy efficiency of CPU, GPU and FPGA implementations for vision kernels.". *IEEE International Conference on Embedded Software and Systems (ICES)*, pages 1–8, 2019. DOI: <https://doi.org/10.1109/ices.2019.8782524>.
- [19] Y. A. Gean and K. Ganesan. "Measuring the Power-Constrained Performance and Energy Gap between FPGAs and Processors.". *ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, pages 285–285, 2017. DOI: <https://doi.org/10.1145/3020078.3021756>.
- [20] W. Tianwen. "yolo-v4-tf — OpenVINO.". Github repository, 2020. URL https://github.com/TNTWEN/OpenVINO-YOLOV4/blob/master/yolo_v4.py.
- [21] A. Mironova. "ctdet_coco_dlav0.512.". Github repository, 2020. URL https://github.com/openvinotoolkit/open_model_zoo/blob/master/tools/accuracy_checker/configs/ctdet_coco_dlav0.512.yml.