








Article

A Performance Analysis of You Only Look Once Models for Deployment on Constrained Computational Edge Devices in Drone Applications

Lucas Rey , Ana M. Bernardos * , Andrzej D. Dobrzycki , David Carramiñana , Luca Bergesio ,
Juan A. Besada  and José Ramón Casar 

Information Processing and Telecommunications Center, ETSI Telecomunicación, Universidad Politécnica de Madrid, Av. Complutense 30, 28040 Madrid, Spain; luca.bergesio@upm.es (L.B.); juanalberto.besada@upm.es (J.A.B.); joseramon.casar@upm.es (J.R.C.)

* Correspondence: anamaria.bernardos@upm.es

Abstract: Advancements in embedded systems and Artificial Intelligence (AI) have enhanced the capabilities of Unmanned Aircraft Vehicles (UAVs) in computer vision. However, the integration of AI techniques on-board drones is constrained by their processing capabilities. In this sense, this study evaluates the deployment of object detection models (YOLOv8n and YOLOv8s) on both resource-constrained edge devices and cloud environments. The objective is to carry out a comparative performance analysis using a representative real-time UAV image processing pipeline. Specifically, the NVIDIA Jetson Orin Nano, Orin NX, and Raspberry Pi 5 (RPI5) devices have been tested to measure their detection accuracy, inference speed, and energy consumption, and the effects of post-training quantization (PTQ). The results show that YOLOv8n surpasses YOLOv8s in its inference speed, achieving 52 FPS on the Jetson Orin NX and 65 fps with INT8 quantization. Conversely, the RPI5 failed to satisfy the real-time processing needs in spite of its suitability for low-energy consumption applications. An analysis of both the cloud-based and edge-based end-to-end processing times showed that increased communication latencies hindered real-time applications, revealing trade-offs between edge (low latency) and cloud processing (quick processing). Overall, these findings contribute to providing recommendations and optimization strategies for the deployment of AI models on UAVs.

Keywords: YOLOv8; edge computing; TensorRT; NCNN; inference performance; quantization; NVIDIA Jetson; Raspberry Pi 5; autonomous drones



Academic Editors: Sergio Saponara and Abdussalam Elhanashi

Received: 10 January 2025

Revised: 30 January 2025

Accepted: 4 February 2025

Published: 6 February 2025

Citation: Rey, L.; Bernardos, A.M.; Dobrzycki, A.D.; Carramiñana, D.; Bergesio, L.; Besada, J.A.; Casar, J.R. A Performance Analysis of You Only Look Once Models for Deployment on Constrained Computational Edge Devices in Drone Applications.

Electronics **2025**, *14*, 638.

<https://doi.org/10.3390/electronics14030638>

Copyright: © 2025 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<https://creativecommons.org/licenses/by/4.0/>).

1. Introduction

In recent years, the integration of unmanned aerial vehicles (UAVs), commonly referred to as drones, with machine learning techniques has significantly advanced the field of aerial robotics. Machine learning algorithms, particularly deep learning approaches, enable drone-based systems to process large volumes of data collected from onboard sensors. The incorporation of computer vision algorithms into drone technologies has enhanced their autonomous capabilities, allowing drones to detect and track objects with high accuracy. Additionally, these advancements support collaborative data fusion, where information from multiple drones and sensors is combined to improve situational awareness and inform decision-making processes. Such capabilities contribute to the effective management and coordination of drone operations in complex environments [1,2], opening the door to possible applications such as vision-driven swarming [3].

While these intelligent capabilities offer significant advantages, embedding them into onboard or even edge systems remains a challenge. By integrating these algorithms directly into drones, they can operate autonomously without relying on constant communication with a central system. This autonomy is important in low-connectivity scenario applications, such as unsupervised maintenance flights in remote areas (e.g., oil pipelines or power pylons), defense operations, or emergency response situations where the connectivity may be limited [4,5]. In these cases, which also seek energy-efficient methods to avoid early battery drain, drones must be able to independently detect and track objects, and algorithms like YOLO (You Only Look Once) enhance their ability to do so effectively.

YOLOv8, as a state-of-the-art object detection algorithm, is well suited to these scenarios due to its multiple versions, with each optimized for different hardware constraints. For instance, versions such as YOLOv8-small (YOLOv8s) and YOLOv8-nano (YOLOv8n) contain fewer parameters, making them appropriate for deployment on the smaller, computationally constrained devices commonly used in edge environments. These versions trade off some accuracy for computational efficiency, allowing for real-time processing even on resource-constrained devices [6]. Moreover, YOLOv8 demonstrates adaptability to varying input resolutions, which is important in applications such as drone operations, where the cameras can range from thermal imaging for rescue missions to high-definition cameras for other purposes [7]. Additionally, its integration with tracking algorithms like DeepSORT supports object tracking in sequences of frames, which is valuable for scenarios that require dynamic monitoring, such as surveillance or search and rescue missions [8]. YOLOv8 also supports multitasking by combining object detection with classification and segmentation in a single model, enabling it to address various computer vision tasks in controlled settings [7]. As an evolving framework, using YOLOv8 as one of the latest versions of YOLO eases the transition to future developments of the algorithm.

Furthermore, techniques such as quantization can facilitate enhanced performance through reductions in the bit width, which refers to the number of bits used to represent data in computations. By reducing the bit width, the model performs calculations with fewer bits, thereby decreasing both the memory footprint and the computational load. This is relevant when running complex algorithms on devices with limited hardware capabilities, as it allows for more efficient use of the available resources without sacrificing the real-time performance. By applying quantization in combination with different fine-tuned YOLO models of varied sizes, we aim to derive guidelines for the optimal use of object detection algorithms like YOLOv8 in constrained environments. These insights may be useful for the future integration of such algorithms into distributed agents at the edge, where autonomy and performance are paramount [4,9,10].

Given the challenges outlined, the primary objectives of this article are as follows:

1. To comprehensively evaluate the performance of YOLOv8 models across different embedded platforms, including the Jetson Orin Nano, Jetson Orin NX, and Raspberry Pi 5, in order to determine their suitability for real-time object detection.
2. To investigate the impact of various quantization techniques and floating-point precision models on the balance between the detection accuracy, processing speed, and power consumption, optimizing these trade-offs to ensure low-latency inference in edge computing applications.
3. To validate the performance of the aforementioned devices in a representative deployment scenario for drone-based applications, which consists of integrating the models into a real drone image processing workflow. This integration also enables comparing edge and cloud solutions for object detection inference.
4. To assess the specific limitations and potential of these platforms for real-time drone operations, identifying the constraints in terms of energy efficiency, algo-

rhythmic processing time, and scalability within real architectures, and to derive practical guidelines for selecting the appropriate models and devices for various deployment scenarios.

The organization of this article is as follows. Section 2 provides a comprehensive review of the state of the art, detailing advancements in deploying deep learning on energy-efficient devices into a distributed edge architecture and exploring quantization techniques in object detection models. Section 3 describes the methodology, including the evaluation setup, dataset creation, and the optimization of YOLO object detection models tailored to edge devices, alongside a detailed testing strategy. Section 4.1 presents a rigorous evaluation of quantized YOLO models on resource-constrained devices, assessing their inference performance, accuracy degradation, and energy consumption. Section 5 expands on the performance in a realistic testbed, comparing edge and cloud solutions for object detection and discussing the practical implications. Finally, Section 6 synthesizes the findings and proposes guidelines, while Section 7 concludes this paper, highlighting potential directions for future research, including the integration of digital twin simulations and enhanced edge–cloud collaboration for drone operations.

2. The State of the Art

This section is divided into two subsections. The first Section 2.1, *Deploying Deep Learning on Energy-Efficient Devices*, evaluates the performance and optimization techniques for deep learning models deployed on edge hardware, with a focus on energy efficiency and real-time processing. The second subsection, Section 2.2, *Quantization Techniques in Object Detection Models*, reviews strategies to optimize the computational and memory demands in object detection algorithms, analyzing their feasibility for edge devices in constrained environments.

2.1. *Deploying Deep Learning on Energy-Efficient Devices*

In recent years, significant research has focused on optimizing and evaluating the performance of deep learning models on edge devices, particularly for object detection tasks in resource-constrained environments. Various studies have benchmarked the hardware configurations to identify the optimal settings for achieving a balance between the inference speed, energy consumption, and real-time processing requirements. For instance, Baller et al. [9] evaluated a range of edge devices, including the NVIDIA Jetson Nano, and identified the optimal configurations in terms of the inference speed and energy consumption across various deep learning frameworks and classification models. Such evaluations are crucial for understanding the limitations and potential of hardware in real-time scenarios.

Building on hardware-specific optimizations, Li et al. introduced [11], an optimization framework for CNN inference on edge devices for application in an IoT context. This framework combines offline pruning and quantization techniques with runtime optimizations, such as multiplication reduction, data layout adjustments, and data parallelization. The experimental results showed performance improvements of $1.96\times$ in high-end devices and $1.73\times$ in low-end devices in improving their edge device efficiency using advanced model compression techniques. Similarly, Ferraz et al. [12] benchmarked CNN inference on edge devices, such as the NVIDIA Jetson AGX Xavier and the Movidius Neural Stick, noting considerable improvements in the inference time and power efficiency.

Recent studies also emphasize the use of distributed and cooperative processing to enhance edge device performance. Hou et al. [13] introduced *DistrEdge*, a method designed to enhance CNN inference across distributed edge devices by employing deep reinforcement learning. This approach distributes inference tasks among multiple devices,

optimizing the computational load and significantly improving the overall inference speed. DistrEdge dynamically adapts to the heterogeneity of devices and varying network conditions, ensuring efficient load balancing and reduced latency. This adaptability makes it particularly effective in real-time processing scenarios. In another cooperative approach, Liang et al. [14] presented Edge YOLO, leveraging edge–cloud cooperation for real-time object detection in autonomous driving, underscoring the benefits of a distributed architecture in complex, time-sensitive tasks.

In the realm of object detection, Shin and Kim [15] explored the performance of YOLO object detection models on NVIDIA Jetson devices, highlighting the advantages of using TensorFlow-TensorRT (TF-TRT) to optimize real-time inference. They also found that while TF-Lite is effective for mobile, it fails to utilize the GPU resources on the Jetson, highlighting the need to select appropriate frameworks based on the hardware and application needs. Xiong et al. [16] further explored YOLO models by optimizing YOLOv5s for UAV image detection on the Jetson Xavier NX, achieving notable improvements in FPS and model compression. Similarly, Hu et al. [17] proposed an improved YOLO-Tiny-attention algorithm for fault detection on wind turbine blades, which was successfully deployed on the same hardware with increased detection precision. These findings highlight the adaptability of YOLO-based models in different edge applications.

Table 1 summarizes the key findings from these studies, highlighting the energy consumption and inference performance across different edge devices and models.

Table 1. Comparison of energy efficiency and inference latency across reviewed studies.

Authors	Device	Model	Metrics	Application Area	Key Findings
Baller et al. 2021 [9]	NVIDIA Jetson Nano	MobileNetV2, DNNs	Inference speed, accuracy, energy consumption	General classification	Optimized frameworks needed to improve inference time and energy use in Jetson Nano.
Li et al. 2023 [11]	NVIDIA Jetson Nano, Raspberry Pi	MobileNetV2, VGG16	Inference speed, energy consumption	IoT applications	Jetson Nano is faster and more energy-efficient than Raspberry Pi, but neither meets real-time needs for complex tasks.
Ferraz et al. 2023 [12]	NVIDIA Jetson AGX Xavier, Movidius Neural Stick	SqueezeNet, CNNs	Inference speed, energy consumption	Edge devices	Pruning and quantization improve energy efficiency and performance on AGX Xavier.
Hou et al. 2022 [13]	NVIDIA Jetson Nano, TX2, Xavier	CNN	Distributed inference performance, speedup	Edge devices	Distributed inference framework (DistrEdge) with reinforcement learning improves real-time performance.

Table 1. Cont.

Authors	Device	Model	Metrics	Application Area	Key Findings
Liang et al. 2022 [14]	Edge–cloud, NVIDIA Jetson	Edge YOLO	Real-time performance, object detection accuracy	Autonomous vehicles	Pruning and feature compression improve efficiency in edge–cloud systems, enhancing autonomous vehicle operations.
Shin and Kim 2022 [15]	NVIDIA Jetson AGX Xavier	YOLOv4-Native, YOLOv4-Tiny	Latency, energy consumption	Real-time object detection	TF-TRT and TensorRT optimizations improve real-time inference and reduce energy use on Jetson AGX Xavier.
Xiong et al. 2023 [16]	NVIDIA Jetson Xavier NX	GCGE-YOLO	mAP, FPS, energy consumption	UAV image detection	Uses GhostNet and coordinate attention to reduce computational load and improve object detection accuracy.
Hu et al. 2023 [17]	NVIDIA Jetson Xavier NX, Jetson Nano	YOLO-Tiny-attention	FPS, detection precision	Wind turbine fault detection	Attention mechanisms improve fault detection accuracy on low-power edge devices, supporting real-time monitoring.

While many studies focus on evaluating deep learning algorithms on individual edge devices, such as the NVIDIA Jetson Nano or AGX Xavier, there is growing recognition of the need to explore more complex architectures. These include distributed computing systems and edge–cloud cooperation, which are gaining significance for handling sophisticated tasks in real-time applications. Several studies, such as Hou et al. [13] and Liang et al. [14], have begun to explore the potential of distributed architectures to overcome the limitations of individual devices, demonstrating the benefits of dynamic workload balancing and feature compression in real-time object detection tasks.

Distributed edge architectures, when integrated with tactical cloud technology and 5G networks, enable real-time data flow among edge devices [18]. This integration enhances the performance of autonomous drones by supporting efficient coordination and rapid decision-making. By combining local data processing with continual connectivity, such architectures improve the operational efficiency and autonomy in dynamic environments [19]. In addition, the network slice feature segments the network into specific virtual environments to enhance the resource allocation and service quality per application [20]. This ensures that in drone systems, each mission or drone group receives a tailored “slice” with low latency and high reliability, important for enabling efficient operations.

By integrating these advanced communication frameworks with quantization techniques [21–23], researchers can improve the energy efficiency and performance in

edge systems. This combination of tactical cloud, 5G, and model quantization addresses the specific challenges in distributed architectures, such as latency reduction and resource optimization. These improvements support the development of more autonomous and efficient systems capable of operating effectively in energy-constrained real-world scenarios.

2.2. Quantization Techniques in Object Detection Models

Quantization techniques are valuable for optimizing deep learning models for deployment in edge devices, improving the inference speed and energy efficiency, especially in real-time applications such as autonomous drone navigation and object detection. Common techniques include post-training quantization (PTQ), Quantization-Aware Training (QAT), and mixed-precision quantization (MPQ), all of which enhance the energy efficiency in reducing the computational complexity while maintaining the model's performance.

PTQ is a technique that reduces the precision of a trained model to formats such as 8-bit integers by converting the weights and activations from floating-point representations. One of its primary advantages is that it reduces the size of the model and increases the inference speed without requiring retraining, preserving the original training process. For instance, Przewlocka-Rus et al. [22] demonstrated that PTQ applied to object detection models on the NVIDIA Jetson Nano decreased the inference times by up to 35% with a less than 2% degradation in accuracy. Similarly, Jiang et al. [21] evaluated various PTQ approaches, including affine, logarithmic, and dynamic quantization, and found that these methods effectively minimized the size of models while maintaining high accuracy with minimal losses.

QAT integrates quantization into the training process, allowing the model to learn to minimize quantization errors. This method typically results in higher accuracy than PTQ, especially for complex models and tasks. Gupta and Asthana [24] applied QAT to object detection models such as YOLO, achieving a 1.8% improvement in mAP compared to PTQ, with reductions in the model size and inference time.

MPQ uses different precisions for various parts of the model, balancing trade-offs between size, speed, and accuracy. Critical layers use higher precision, while less critical layers use lower precision. Park et al. [25] explored mixed-precision quantization in object tracking models, achieving energy consumption reductions of up to 45% and inference time improvements of 30% on edge devices like the Jetson Nano and Raspberry Pi. Furthermore, Al-Hamid and Kim [23] proposed a Unified Scaling-Based Pure-Integer Quantization (USPIQ) method that reduced the on-chip memory by 75% with only a 0.61% loss in the mAP while achieving a 2.84x speedup in the inference time compared to traditional methods.

These quantization approaches improve the energy and space efficiency in edge device deployments, making the implementation of sophisticated object detection and tracking models viable. However, the effects of these techniques on the latest YOLO models and their integration into real-time drone applications are still being researched. Studies on deploying YOLO models in different edge–cloud or embedded edge implementations are limited [21–23,25].

Despite significant advances in the research on object detection at the edge, several gaps remain. Multiple studies have assessed edge devices in isolation without results on the latency in the interaction between the hardware and software processes.

To address these gaps, this study focuses on YOLOv8 models applied to edge environments, emphasizing their accuracy, inference speed, and energy consumption. By incorporating quantization techniques, this research offers new insights into the potential of quantization approaches to optimize YOLOv8 for resource-limited edge scenarios. The findings deliver practical recommendations for selecting the configurations in power-

constrained environments, with a particular focus on the Jetson Orin Nano, Jetson Orin NX, and Raspberry Pi 5 as innovative edge devices.

3. Methodology

In this study, three edge computing devices are used to evaluate the performance of YOLOv8 models and their quantized versions: the Raspberry Pi 5, the Jetson Orin Nano, and the Jetson Orin NX. These devices were chosen due to their different hardware capabilities, allowing for an evaluation of object detection tasks under various processing constraints. Table 2 provides an overview of the key hardware specifications for each device.

Table 2. Comparison of hardware characteristics of the devices used in this study.

Feature	Raspberry Pi 5	Jetson Orin Nano	Jetson Orin NX
CPU	4-core ARM Cortex-A76 at 2.4 GHz	6-core ARM Cortex-A78AE	8-core ARM Cortex-A78AE
GPU	VideoCore VII	1024-core Ampere + 32 Tensor Cores	2048-core Ampere + 64 Tensor Cores
RAM	8 GB	8 GB	16 GB
Memory Bandwidth	Limited	Moderate	High
Typical Power Consumption	5–7 W	7–15 W	10–25 W
TensorRT Support	No	Yes	Yes
Dimensions	85.6 × 56.5 mm	100 × 69.6 mm	100 × 87 mm

These devices will be used to evaluate the performance of YOLOv8 models and their quantized versions under uniform experimental conditions. The Raspberry Pi 5, a CPU-centric device, was selected to assess the capabilities of a modern single-board computer without a dedicated GPU in object detection tasks. In contrast, the Jetson Orin Nano and Jetson Orin NX, which are GPU-powered platforms, were chosen for their high-performance processing capabilities. Together, these devices represent a range of edge computing architectures, enabling an evaluation of the inference speed, accuracy, and energy consumption across diverse hardware configurations.

In order to carry out these evaluations, it is necessary to first have an object detection model. To this end, Section 3.1 describes the process of creating such a model, including the generation of a dedicated dataset in Section 3.1.1, model selection and training in Section 3.1.2, and model optimization in Section 3.1.3. From the model, two types of experiments are developed:

- Experiments that deploy the model in isolation on each of the previous hardware platforms. These experiments aim to evaluate the inference speed and power consumption, for which the results are detailed in Section 4.
- Experiments that integrate the object detection model into a typical drone image processing pipeline. As described in Section 5, these experiments seek to compare the capabilities of edge devices against a cloud deployment under a representative scenario.

In these experiments, a set of metrics are used to evaluate the performance of the models. These metrics are described in Section 3.2.

3.1. Object Detection Model Selection and Training

3.1.1. Dataset Creation

To train the evaluated object detection models, a dataset of annotated images was created. This dataset was collected within the indoor testbed described by the authors

in [26] and depicted in Figure 1. The test setup consists of an indoor room with varied furniture elements to introduce a complex environment. Within the room, a fleet of drones (DJI Tello) can be controlled, together with a fleet of ground robots (TurtleBot) that serve as the targets (i.e., objects) to be detected from the aerial images. The experimentation platform allows you to easily capture frames from the drone's onboard cameras. Also, positioning information can be simultaneously gathered thanks to an OptiTrack system (i.e., a commercial high-precision positioning system based on high-speed motion capture cameras). Although this former capability was not exploited, it could be useful in the future to enable automatic target labeling in captured images.

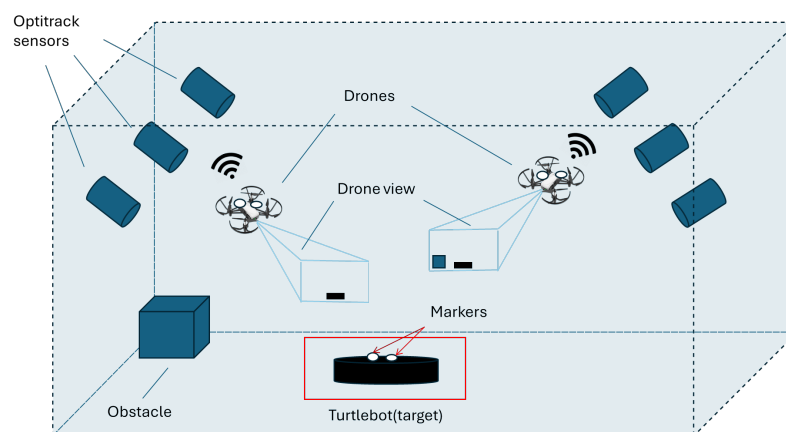


Figure 1. Diagram of the indoor controlled flight environment equipped with OptiTrack sensors, drones with access points, and a TurtleBot target on the ground.

The dataset is composed of 6000 aerial images (640×640 resolution) with the target in different positions relative to the drone's field of view. In order to ensure that the dataset includes a range of angles and perspectives, the drones were flown following various straight-line and diagonal trajectories relative to the target position. Also, altitude changes were also introduced to capture different viewing angles, with certain trajectories varying from 1.5 m down to 0.5. After image gathering, the dataset was manually annotated, as demonstrated by the example images in Figure 2. In general, this dataset serves as the foundation for training and evaluating object detection models, with annotated images integrated into the YOLO object detection pipeline.

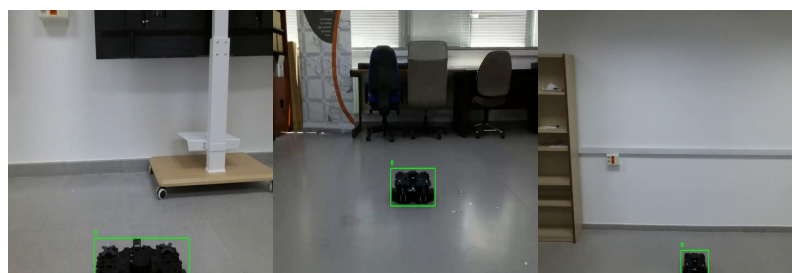


Figure 2. Examples of different dataset images where the object to be detected has been manually annotated. Images are captured from different angles and heights and with different background environments.

3.1.2. Selection and Training of Object Detection Models

For the selection of the model for this study, it was necessary to evaluate different versions of the YOLO architecture in terms of their performance and efficiency. YOLOv9 was considered a potential candidate; however, YOLOv8 demonstrated superior optimization for real-time performance and latency reduction. Moreover, YOLOv8 was elected as the most stable version during the experimental advance of this research.

To align with the objective of deploying models on devices with limited computational resources further, this study specifically focused on the ‘small’ (YOLOv8s) and ‘nano’ (YOLOv8n) versions of YOLOv8. These versions are designed for computationally constrained devices by reducing the depth and number of parameters, making them suitable for various edge deployment scenarios. YOLOv8n, the smallest and fastest version of the model in this family, is optimized for devices with strict resource limitations. Its architecture prioritizes the inference speed by using fewer convolutional layers and simplifying the feature extraction stages.

In contrast, YOLOv8s includes more convolutional layers and feature extraction steps, improving the detection accuracy while maintaining computational efficiency. This model provides a balanced approach, offering a slight increase in the computational demand for better precision, making it suitable for scenarios requiring both fast response times and high accuracy. Both models share features such as anchor-free detection and enhanced feature pyramid networks, supporting their adaptability to diverse edge deployment requirements.

The models were trained using the AdamW optimizer, chosen for its ability to efficiently manage weight updates while applying decoupled weight decay. A learning rate of 0.002 was selected, ensuring fast convergence while avoiding divergence or overfitting, and a weight decay of 0.0005 was applied to most weights, excluding biases. This regularization approach encourages smaller weights, making the model more robust to noise, while excluding biases avoids penalizing parameters critical to feature representation. The momentum was set to 0.9 as the standard value to smooth optimization trajectories and ensure stable weight updates over training iterations. Training was conducted over 100 epochs with a batch size of 16, which was limited by the GPU hardware constraints of the machines used for training, balancing the computational cost and generalization performance. Early stopping with a patience of 10 epochs prevented overfitting by halting the training when no significant validation improvements were observed, saving computational resources. Data augmentation techniques, including mosaic and random erasing, were employed to enhance the diversity of the training samples. Mosaic augmentation combined multiple images into a single input, enabling the model to learn from varying object positions and scales, which is particularly effective for detection tasks. Random erasing increased the robustness by encouraging the model to focus on discriminative features rather than memorizing specific regions. During validation, predictions were made with a confidence threshold of 0.5, ensuring only reliable detections were considered. An IoU threshold of 0.7 defined the overlap criterion for successful detections, achieving a balance between precision and recall.

3.1.3. Optimization Techniques for Edge Deployment

To optimize the original models (small and nano) for edge devices, several techniques were employed, including reducing the precision to FP16 and applying INT8 quantization.

FP16, or Half-Precision Floating Point, reduces the bit precision of floating-point numbers from 32 bits (FP32) to 16 bits. This precision reduction, often implemented using tools such as TensorRT during model export, enhances the performance by decreasing the computational load and memory usage. Despite reducing the precision, FP16 offers inference speed improvements without significantly impacting the accuracy.

INT8 quantization optimizes models by converting weights and activations from floating-point precision into 8-bit integers. This process, often applied through post-training quantization (PTQ), significantly reduces the computational demand and improves the energy efficiency and inference speed, making it suitable for resource-constrained edge devices. However, unlike FP16, which typically preserves the model accuracy, INT8

quantization can lead to more noticeable accuracy degradation, particularly in complex tasks or when it is applied to models with high parameter sensitivity.

In this work, TensorRT and NCNN were selected as the primary frameworks for optimizing and deploying the YOLOv8 models on the Jetson devices and the Raspberry Pi 5, respectively. TensorRT, deeply integrated with NVIDIA hardware, improves the model performance through layer fusion and memory optimization, enabling efficient inference with support for FP32, FP16, and INT8 quantization. ONNX (Open Neural Network Exchange) served as an intermediary format, facilitating precision reduction and compatibility during the model's conversion to TensorRT. For ARM-based architectures like the Raspberry Pi, NCNN was chosen due to its lightweight design and low-latency inference capabilities, which are particularly effective on mobile and edge devices. While other frameworks, such as CoreML and TorchScript, were considered, they were deemed less suitable for this project's goals. CoreML, tailored to iOS, was excluded due to the platform's irrelevance, and TorchScript did not align as closely with the optimization requirements as TensorRT and NCNN.

By tailoring the frameworks and quantization techniques to each deployment environment, the optimization strategy leveraged hardware-specific strengths. For the Jetson Orin Nano and NX, TensorRT was used to apply three quantization levels—FP32, FP16, and INT8—to establish a comparison. On the Raspberry Pi 5, only the FP32 format, as the original format for YOLO, was utilized via the NCNN framework.

3.2. The Evaluation Strategy and Metrics

As previously introduced, the experiments for measuring the performance of the YOLO models were performed in two stages to measure the set of metrics described in this section, shown on Table 3. Initially, in a controlled and isolated setting, we measured the latency, throughput, and energy efficiency to balance the inference speed and accuracy against edge device constraints. The experiments carried out, whose results are described in Section 4.1, included the following:

- **Inference Time Measurement** : This experiment assessed the computational efficiency of each model in isolation by measuring the time to perform inferences on 5000 images at 640×640 resolution. The average inference time was recorded, and the MIS (FPS) metric was derived to indicate the processing capacity.
- **Continuous Inference Evaluation (Inferences Per Minute, IPM)**: In this experiment, each device performed inferences continuously over 60 s, accounting for all of the latency factors, including model loading, network overhead, and processing delays. The IPM metric provided insight into the system's throughput under real-world conditions.
- **Energy Consumption Analysis**: Real-time power usage was monitored using `tegrastats` for Jetson devices and `vcgencmd pmic_read_adc` for the Raspberry Pi 5. Power data, averaged over the duration, were used to calculate the energy consumption (W·s) and energy per inference (J/inference), indicating the energy efficiency necessary for battery-powered deployments.

Secondly, trials were performed in real-world scenarios to assess the performance under actual conditions. This systematic testing provides a comprehensive grasp of the abilities of the model, from isolated evaluations to complete operational contexts. In the latter case, the following test was performed for the results discussed in Section 5:

- **Throughput and Latency Assessment**: The throughput times were measured in both edge and cloud environments using a timestamp-based approach involving a video server, a processing agent, and a prediction client. Timestamps captured the communication times and processing delays, with the Round-Trip Time (RTT)

and prediction time for cloud processing allowing for precise comparisons of the bottlenecks across setups.

Table 3 provides a concise overview of the key metrics used throughout the previous experiments for evaluating the model efficiency, energy usage, and system responsiveness. These metrics were selected to capture distinct aspects of the system performance for real-time processing on edge devices. MIS and IPM measure the inference speed and hardware throughput, respectively, while energy consumption and EPI focus on energy efficiency, needed not only for the design of battery-powered operations but also for maximizing the relative energy utilization based on the relative energy wasted per inference. Together, these metrics provide a comprehensive framework for analyzing and comparing models across different devices and configurations.

In addition to computational metrics, predictive accuracy was assessed using mAP50 and mAP50-95, which evaluate the detection quality across quantization levels and settings. These metrics ensure that the whole evaluation captures both efficiency and reliability.

Table 3. Summary of the performance metrics with their purpose and formulas.

Metric	Purpose	Formula
Model Inference Speed or MIS (FPS)	Measures the speed of model inference in frames per second (FPS), reflecting the real-time processing capability. This measure will only measure the capability of the algorithm in an isolated way, without counting extra processing tasks that produce latency.	$\text{MIS} = \frac{1}{\text{Mean Inference Time (s)}}$
Inferences Per Minute or IPM	Calculates the total inferences in 60 s, representing the overall device throughput. IPM assesses the entire process within the device, including I/O operations, to evaluate the hardware performance comprehensively. It measures the efficiency of the program within the isolated environment of the device.	$\text{IPM} = \text{Total inferences in one minute}$
Energy Consumption or E (W·s)	Indicates the total energy used during model inference, assessing the power consumption. Measured using device-specific tools (tegrastats for the Jetson and vcgencmd pmic_read_adc for the Raspberry Pi), allowing for energy estimation through the average power consumption.	$E = \bar{P} \cdot (t_2 - t_1)$
Energy Per Inference or EPI (J/inference)	Lower EPI values signify a more efficient use of energy per prediction, providing insight into how effectively the model utilizes the available power for inference tasks. As a relative metric, EPI advances our understanding of energy efficiency comparisons across different models and systems.	$\text{EPI} = \frac{\text{Mean Power (W)} \times 60 \text{ s}}{\text{IPM}}$
Round-Trip Time or RTT (ms)	Measures the end-to-end time for a complete inference cycle, including the network latency and processing time, providing a comprehensive assessment of the system's whole architecture's real-time performance.	$\text{RTT} = \text{Client Reception time} - \text{Request sent time}$

Table 3. Cont.

Metric	Purpose	Formula
Throughput of the System (Inferences/s)	Represents the system’s capacity for continuous inferences, derived from the inverse of the RTT. Indicates the efficiency of the whole architecture in handling continuous inference requests over time.	Throughput = $\frac{1}{\text{Average RTT (s)}}$

This experimental setup offers a robust framework for evaluating YOLOv8 models on edge devices. By systematically progressing from isolated testing to real-world deployment, it combines deployment techniques with a detailed performance analysis. This approach supports energy-efficient model optimization tailored to diverse hardware scenarios, ensuring suitability for real-time, low-powered applications and aligning with the goal of delivering practical evaluation strategies for edge-based implementations.

4. Performance Evaluation of Quantized YOLO over Resource-Constrained Devices

This section provides a comprehensive evaluation of the performance of quantized YOLOv8 models on different resource-constrained devices, emphasizing the effects of the quantization techniques and hardware constraints on the model’s efficiency and accuracy. Figure 3 depicts the structured workflow, from dataset creation to model training, followed by quantization and deployment. It showcases the distinct quantization formats (FP32, FP16, and INT8) applied to various devices like the Jetson Orin Nano, Jetson Orin NX, and Raspberry Pi 5, along with their associated production deployment. In particular, FP32 represents the default training format used for this type of model, which is sometimes also treated as the “original” model in the following sections. This visual workflow supports comprehension of the process and identifies the specific models examined in the experiments.

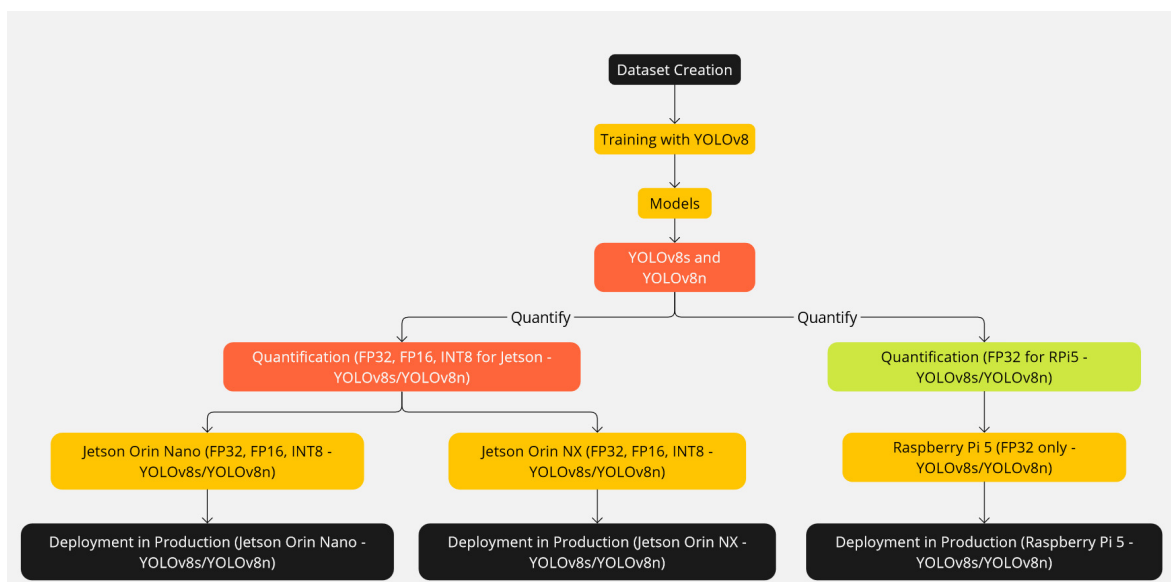


Figure 3. Quantization and deployment process of YOLOv8 models on the Jetson Orin Nano, Jetson Orin NX, and Raspberry Pi 5.

4.1. The Inference Performance

Figure 4 illustrates the Mean Inference Speed (MIS) for the YOLO models on the Orin Nano, Orin NX, and Raspberry Pi devices, comparing the performance with TensorRT (TRT) on the Orin devices and NCNN on the Raspberry Pi.

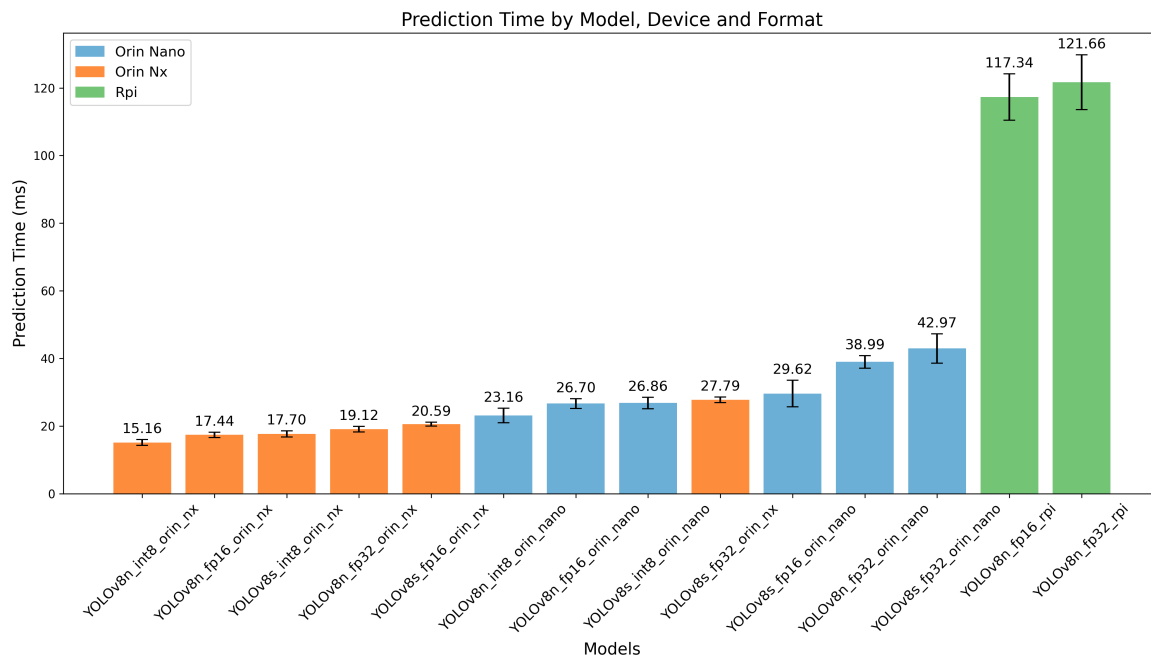


Figure 4. Figure showing the results on the mean iteration times of each model (YOLOv8s or YOLOv8n) with different quantization versions (FP32, FP16, or INT8) within a device (the Orin NX, Orin Nano, or Raspberry Pi 5).

On the Orin Nano, the YOLOv8n_INT8 model excelled with an average iteration time of 23.16 ms, followed by YOLOv8n_FP16 at 26.70 ms and YOLOv8s_INT8 at 28.25 ms. The YOLOv8s_FP32 model had the longest iteration time of 42.97 ms, illustrating the advantages of INT8 quantization in terms of the time performance. The Orin NX showed an even better performance, with YOLOv8n_INT8 achieving the shortest iteration time of 15.16 ms, YOLOv8n_FP16 coming in at 17.44 ms, and YOLOv8s_INT8 at 17.70 ms. In contrast, YOLOv8s_FP32 had the longest time at 27.79 ms, underperforming compared to the leading three Orin Nano models. On the Raspberry Pi, the NCNN-exported models had longer inference times; YOLOv8n_FP32 averaged at 118.00 ms, and YOLOv8s_FP32 reached 211.00 ms, highlighting the hardware constraints and differing efficiencies between CPU and GPU model optimizations. In summary, the Orin Nano and Orin NX vastly outperformed the Raspberry Pi, with the Orin NX being suitable for real-time object detection.

During our trials, we attempted to reduce the numerical precision of the YOLOv8 model on the Raspberry Pi 5 using the available libraries. However, neither FP16 nor INT8 configurations were feasible due to limitations in the precision support of the libraries, particularly with the use of the NCNN framework. As a result, there was no improvement in the performance and a notable loss in the prediction accuracy during these tests. Consequently, only the primary FP32 model configuration was included in the comparative analysis, as it was the only stable and functional setup on the Raspberry Pi 5.

4.2. Degradation Analysis of Model Accuracy

This section compares the YOLOv8s and YOLOv8n models across various edge devices to assess the impact of quantization on both their inference speed and accuracy.

Table 4 provides a summary of the mAP and FPS results for different devices and quantizations, while Figure 5 illustrates the mean iteration times for each model and configuration.

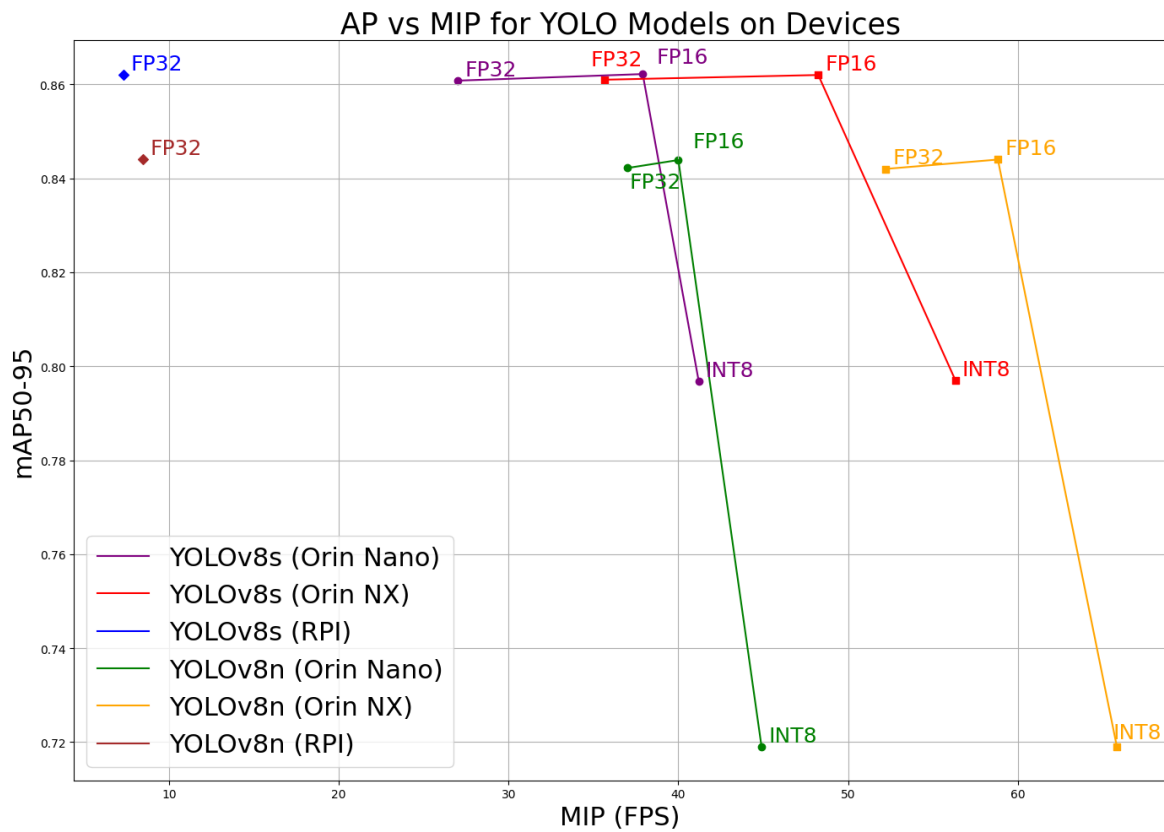


Figure 5. Figure showing the results of the FPS tests of each isolated model (YOLOv8s or YOLOv8n) with different quantization versions (FP32, FP16, and INT8) within a device.

Table 4. Table synthesizing the inference frame rate, energy consumption, inferences per minute, power, and energy per inference in joules.

Device	Model	Quantization	mAP50-95	mAP50	MIS (FPS)	Energy Consumption (W·s)	Inferences per Minute	Energy per Inference (J/inference)
Orin Nano	YOLOv8s	FP32	0.8608	0.9771	27.00	8.790	1391	0.379
	YOLOv8s	FP16	0.8622	0.9771	37.90	7.836	1558	0.302
	YOLOv8s	INT8	0.7968	0.9189	41.20	7.257	1949	0.223
	YOLOv8n	FP32	0.8422	0.9619	37.00	8.345	2041	0.245
	YOLOv8n	FP16	0.8439	0.9619	40.00	7.422	2144	0.208
	YOLOv8n	INT8	0.7190	0.8272	44.90	7.483	2425	0.185
Orin NX	YOLOv8s	FP32	0.8610	0.9781	35.65	14.155	2129	0.399
	YOLOv8s	FP16	0.8620	0.9780	48.24	12.815	2760	0.279
	YOLOv8s	INT8	0.7970	0.9195	56.32	11.002	2836	0.233
	YOLOv8n	FP32	0.8420	0.9621	52.19	12.480	3182	0.235
	YOLOv8n	FP16	0.8440	0.9622	58.82	10.853	3214	0.203
	YOLOv8n	INT8	0.7190	0.8276	65.83	10.305	3443	0.179
RPI5	YOLOv8s	FP32	0.8620	0.9621	7.32	5.422	217	1.498
	YOLOv8n	FP32	0.8440	0.9612	8.47	5.441	442	0.738

Table 4 presents a summary of the inference performance (in frames per second), energy consumption, the number of inferences per minute, and the energy consumed per inference in joules for the devices Jetson Orin NX, Jetson Orin Nano, and Raspberry Pi 5. The results demonstrate the influence of the disparate quantization configurations (FP32,

FP16, INT8) on the energy efficiency and model performance. The energy consumption analysis shows differences between the absolute energy consumption demands and energy efficiency when comparing the Jetson Orin NX, Orin Nano, and Raspberry Pi 5 devices across different quantization methods (FP32, FP16, and INT8). For the Jetson Orin Nano, the energy consumption varies between 7.4 W and 8.7 W, while for the Jetson Orin NX, it ranges from 10 W to 14 W, indicating a more substantial increase.

For the Jetson Orin Nano, the energy consumption ranges from 7.4 W (FP32) to 8.7 W (INT8), reflecting a modest increase of 1.3 W. The Orin NX shows greater variation, ranging from 10 W to 14 W, which aligns with its higher processing capacity. Despite its additional energy demands, the Orin NX delivers notable improvements in its inference speed, particularly with INT8 quantization.

An interesting result is the performance of YOLOv8s on the Orin Nano with FP16 quantization, achieving 37.90 FPS and a mAP50-95 of 0.8622. This configuration slightly exceeds the performance of the Orin NX running YOLOv8s with FP32 (35.65 FPS, mAP50-95 of 0.8610), demonstrating the potential of FP16 quantization to enhance the efficiency even on less advanced hardware. For the YOLOv8n model, the Orin NX achieves 58.82 FPS with FP16, compared to 40.00 FPS on the Orin Nano, indicating its advantage in handling smaller, compact models.

The Raspberry Pi 5, using FP32 quantization via NCNN, achieves 7.32 FPS on YOLOv8s (mAP50-95 of 0.8620). While functional, its performance is limited compared to that of the Jetson devices, suggesting it is better suited to less demanding applications where real-time processing is not critical.

4.3. Energy Consumption Analysis

The mean average energy consumption was measured over 5000 images, as depicted in Figure 6.

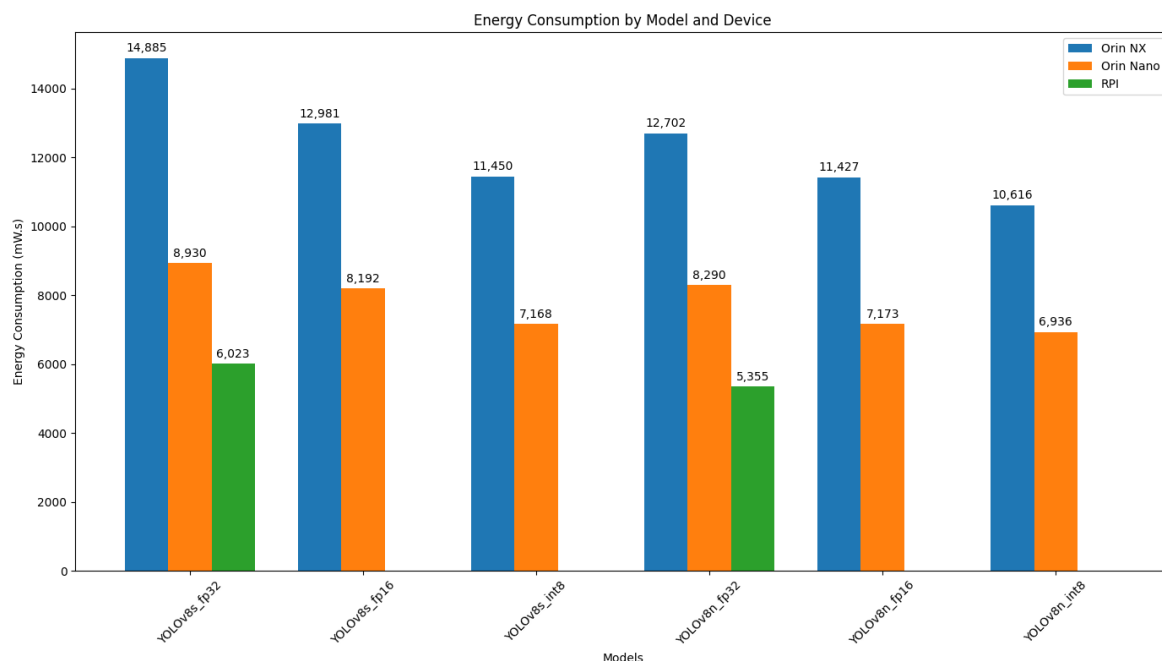


Figure 6. Energy consumption by model and device.

For instance, in the YOLOv8 FP32 setup, the Orin NX consumes 14.155 W, whereas the Orin Nano requires only 8.790 W in the same configuration. Despite the Orin NX's higher power draw, it generally achieves a greater energy efficiency—measured as the energy per inference (J/inference)—across the configurations due to its higher inference throughput.

With INT8 quantization, for example, the Orin NX achieves 0.179 J/inference for YOLOv8n, compared to 0.185 J/inference on the Orin Nano, indicating more efficient use of energy for each inference completed.

The variations in the power consumption between the quantization methods are also noteworthy. For the Orin Nano, the energy consumption rises from 7.4 W in FP32 to 8.7 W in INT8, reflecting a modest increase of 1.3 W. In contrast, the Orin NX exhibits a larger increase, with the power consumption ranging from 10 W in YOLOv8n_INT8 to 14 W in YOLOv8s_FP32, a difference of 4 W.

The Orin Nano demonstrates lower absolute energy consumption and achieves better results in terms of the energy per inference under certain configurations. For instance, in the YOLOv8s_FP16 setup, it consumes 7.836 W and processes 1558 inferences per minute, resulting in an energy consumption of 0.302 J/inference. Additionally, in FP32 mode, the Orin Nano achieves a better energy efficiency than that of the Orin NX, with 0.379 J/inference compared to 0.399 J/inference.

The Raspberry Pi 5, by comparison, shows the lowest absolute energy consumption but also the lowest energy efficiency due to the small number of predictions per time unit. In the YOLOv8s_FP32 configuration, for example, it consumes 5.422 W·s while achieving 217 inferences per minute, resulting in an energy cost of 1.498 J/inference. The Orin Nano devices, while demonstrating higher absolute energy consumption, also achieves commendable efficiency due to the much better performance in number of inferences per unit of time.

5. Performance Evaluation Using a Realistic Testbed

This section aims to assess the performance of different hardware devices, specifically the Orin Nano, Orin NX, and Raspberry Pi 5, using an object detector in a realistic deployment scenario that mirrors real-world drone operations. In this sense, this section takes advantage of the platform presented by the authors in [26] which considers the core elements (e.g., telemetry, video streaming, and data processing) of a real-time drone system, where the interplay of computation and communications affects the latency and system throughput. As depicted in Figure 7, the testbed platform allows live drone images to be processed both in an edge device and in a cloud environment. This enables performance benchmarking of edge algorithms against a centralized cloud, providing a detailed comparison.

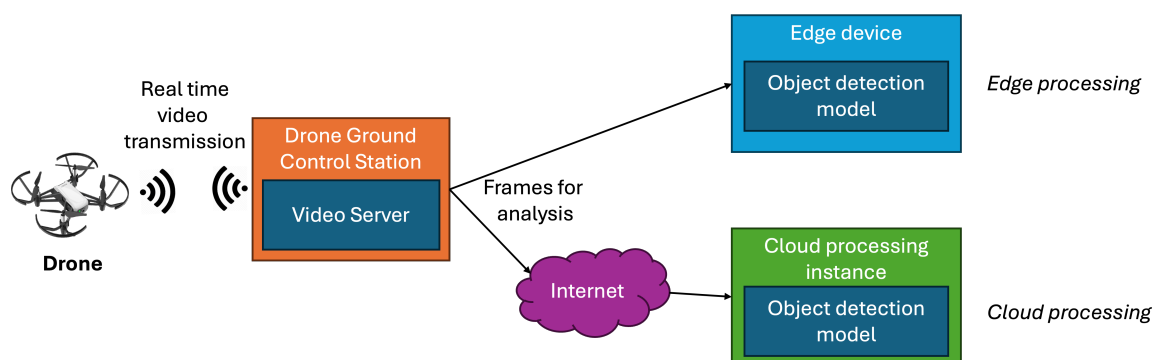


Figure 7. General system deployment architecture.

The deployment architecture consists of three main components: a drone ground control station (GCS) that gathers video in real time from a drone and broadcasts it using the WebSocket protocol, an edge computing device to perform object detection on video streams using TensorRT optimization, and a cloud processing instance (accessible through

the internet) that also allows for the deployment of the object detection model. The interaction between these entities is illustrated in Figure 8.

The workflow begins with the transmission of video from the drone to its associated ground control station. Then, using WebSocket communication for low-latency and reliable data transfer, the video frames can be forwarded either to an edge computing device, which performs the inference locally (at the edge), or to a cloud instance that performs inference remotely (in the cloud). AWS SageMaker is employed for cloud detection, allowing for a systematic comparison of the latency and processing times between the edge and cloud setups. In either case, the inferences and annotated images are sent back to the ground control station.

Within this framework, two tests were performed. In Section 5.1, different edge devices are compared in the real, representative testbed. Then, in Section 5.2, a test is performed between edge and cloud processing. The latter evaluation allows for an analysis of the trade-offs between local and centralized processing in terms of the computational power and communication delays. In both cases, by timing each phase (i.e., video transmission, object detection, and prediction distribution), it is possible to identify latency issues and potential bottlenecks.

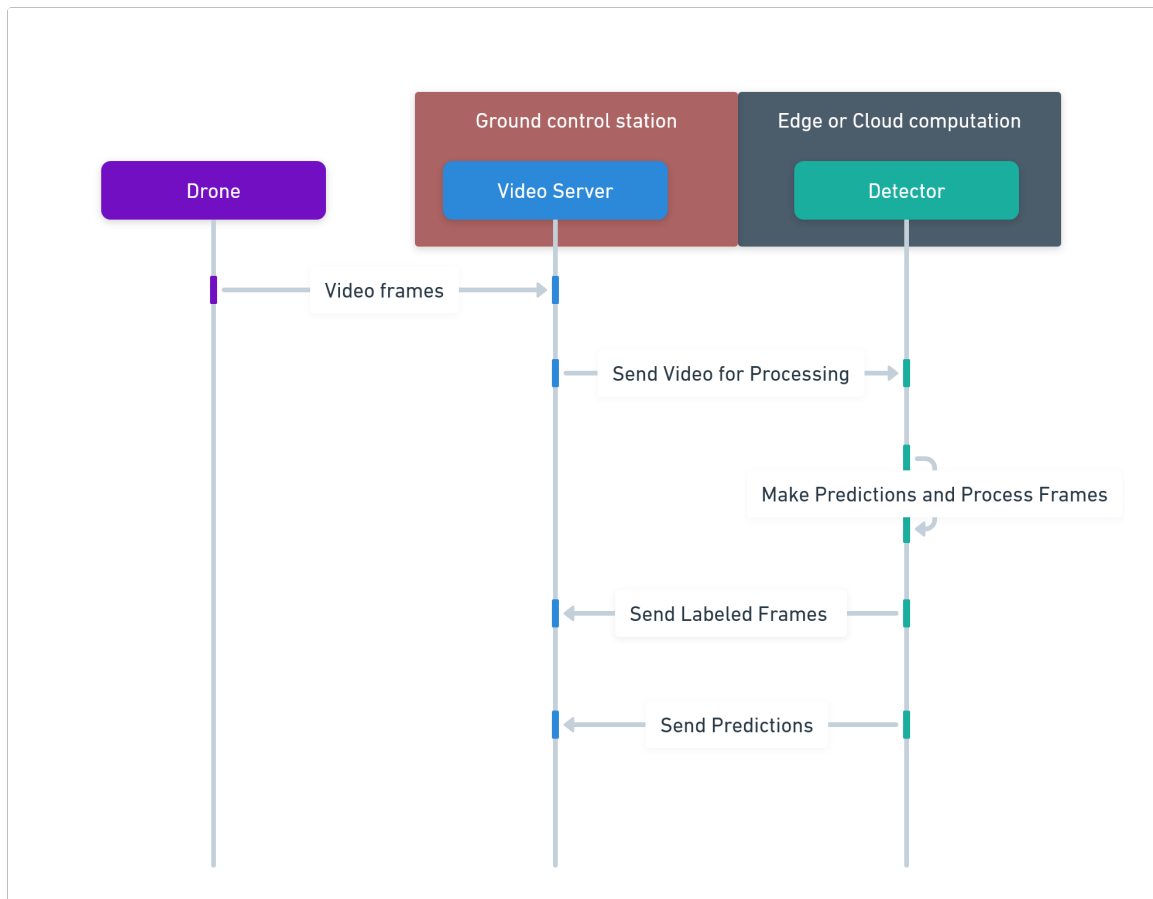


Figure 8. Data flow for real-time video processing and predictions at the edge.

5.1. Edge Deployment

Table 5 provides a summary of the RTT times, RTT deviations, processing times, and throughput (FPS) for the different models and quantization configurations.

Table 5. Table synthesizing the RTT, RTT std, processing time, and throughput of the system (FPS) for different quantization models.

Device	Model	Quantization	RTT (ms)		Processing Time (ms)		Throughput (Inferences/s)
			Mean	Std	Mean	Std	
Orin Nano	YOLOv8n	FP16	49.70	4.59	23.81	20.12	20.12
	YOLOv8n	FP32	60.52	12.24	28.77	16.52	16.52
	YOLOv8n	INT8	48.80	8.42	20.54	20.49	20.49
	YOLOv8s	FP16	49.44	6.46	28.90	20.23	20.23
	YOLOv8s	FP32	73.18	4.89	47.18	13.66	13.66
	YOLOv8s	INT8	44.27	8.27	20.62	22.59	22.59
	Orin NX	YOLOv8n	FP16	34.15	3.65	16.12	29.28
YOLOv8n		FP32	35.76	2.28	18.25	27.97	27.96
YOLOv8n		INT8	29.95	3.19	14.29	33.38	33.39
YOLOv8s		FP16	35.09	4.35	18.50	28.50	28.5
YOLOv8s		FP32	48.97	8.83	26.50	20.42	20.42
YOLOv8s		INT8	33.60	3.31	15.96	29.76	29.76

Regarding the RTT (Round-Trip Time), the Jetson Orin Nano demonstrated its best performance in terms of the RTT with the YOLOv8s INT8 model, achieving a minimum RTT of 44.27 ms. The RTT increased to 73.18 ms when using the YOLOv8s FP32 model. For the Jetson Orin NX, the minimum RTT was observed at 29.95 ms with the YOLOv8n INT8 model, while the maximum RTT occurred with the YOLOv8s FP32 model at 48.97 ms.

The RTT deviation results showed that the Orin Nano showed higher variability with the YOLOv8n FP32 model, which had a deviation of 12.24 ms. The lowest deviation for the Orin Nano was seen with the YOLOv8s FP32 model at 4.89 ms. For the Orin NX, the RTT fluctuations were smaller overall, with the highest deviation being 8.83 ms for the YOLOv8s FP32 model and the lowest at 2.28 ms for YOLOv8n FP32.

During the testing phase, we attempted to implement the architecture on the Raspberry Pi 5. However, it was not possible to complete the tests in full due to the computational limitations. This device exhibited significant latency in its predictions, and this led to instability not only in drone control but also in the acquisition and dissemination of images across the different components of the architecture. Consequently, the results obtained from the Raspberry Pi 5 are sparse and incomplete and therefore not viable as a reliable architecture for this application.

In terms of the processing time, the Orin Nano showed a minimum processing time of 20.54 ms with the YOLOv8n INT8, and the longest processing time was 47.18 ms for the YOLOv8s FP32 model. The processing times on the Orin NX were generally shorter, with a minimum of 14.29 ms for YOLOv8n INT8 and a maximum of 26.50 ms for YOLOv8s FP32.

5.2. Edge vs. Cloud Comparison

In order to assess the trade-offs between computational power and latency in edge and cloud environments, we conducted a series of experiments comparing the two setups using the YOLOv8s model in its FP16 configuration on the Orin Nano. The primary metrics evaluated were the Round-Trip Time (RTT), model processing latency, and communication latency, offering a comprehensive understanding of how both environments handle real-time data processing and communication delays.

For our cloud-based deployment, we selected a cloud instance located in London (eu-west-2) using the *ml.g4dn.xlarge* configuration, which is powered by an NVIDIA T4 GPU. To handle the model inferences efficiently, we deployed the model on a Triton Inference Server within the SageMaker environment. Triton, known for its high performance and flexibility in supporting multiple AI frameworks and backends, provided a robust platform

for managing the inference requests for our YOLOv8s model. The most decisive factor in choosing the NVIDIA Triton Inference Server was its support for TensorRT optimization, which allowed us to conduct a fair comparison under consistent conditions. Additionally, by utilizing Triton, we could take advantage of advanced features such as dynamic batching, model ensemble, and TensorRT acceleration, thereby maximizing the inference speed and fully leveraging the computational power of the cloud instance.

The model was deployed and invoked directly through the SageMaker SDK API's invoke endpoint, which facilitated direct HTTP-based invocation. However, it is important to note that the invoke endpoint does not support WebSocket communication, meaning that all inference requests are handled over HTTP. This limitation may introduce additional latency in scenarios where persistent, real-time communication channels are desired. The London (eu-west-2) location helped us explore the impact of geographical data transmission delays from Madrid, providing insights into latency management for future applications.

Table 6 presents a comparative investigation of the latency metrics between the edge deployments on the Orin Nano and the cloud deployments using the YOLOv8s model in FP16 format. The edge RTT latency shows a mean of 35.09 ms, which is within the feasible limits for real-time applications. By contrast, the cloud deployment shows a much higher mean RTT of 348.21 ms due to the increased communication distance to remote GPU servers. This difference is reflected in the communication latency, where the edge setup averages at 2.50 ms, whereas the cloud averages at a considerably higher value of 341.41 ms.

Table 6. Latency results in the two different deployment scenarios, edge and cloud.

Processing Environment	RTT Latency (ms)	Model Processing Latency (ms)	Communication Latency (ms)
Edge	Mean: 35.09 Std: 4.25	Mean: 32.59 Std: 4.27	Mean: 2.50 Std: 0.34
Cloud	Mean: 348.21 Std: 69.88	Mean: 6.82 Std: 0.05	Mean: 341.41 Std: 69.89

The processing latency data highlight the computational edge of the cloud, with an average latency of 6.82 ms, significantly outperforming the edge's latency of 32.59 ms. Nonetheless, the high communication latency of the cloud counteracts its processing efficiency, resulting in performance constraints.

6. Discussion

An evaluation of various quantization techniques and their impact on inference latency has confirmed INT8 as the fastest configuration, achieving notable reductions in the processing times across all evaluated devices. For instance, on the Orin Nano, YOLOv8s with INT8 achieves 41.20 FPS compared to 27.00 FPS with FP32, representing a significant improvement in speed. However, this increase comes at the cost of reduced accuracy, with the mAP50-95 dropping from 0.8608 in FP32 to 0.7968 in INT8—a reduction of approximately 6.4 percentage points. In contrast, FP16 emerges as a balanced alternative, offering slight improvements in accuracy while reducing the latency. For example, YOLOv8s on the Orin Nano achieves 37.90 FPS in FP16 with an accuracy of 0.8622 mAP50-95, providing an optimal balance for scenarios requiring both speed and precision. This quantization approach also enables intermediate models, such as YOLOv8s, to outperform smaller versions like YOLOv8n_FP32 on more advanced devices such as the Orin NX, where YOLOv8s_FP16 achieves 48.24 FPS with a mAP50-95 of 0.8620, compared to YOLOv8n_FP32's 35.65 FPS and 0.8420 mAP50-95. Furthermore, the application of quantization techniques has been ob-

served to contribute to marginal enhancements in the model's generalization, possibly due to the reduced computational complexity, enabling more efficient image processing. This is particularly relevant for tasks involving real-time object detection, where the computational resources are constrained. Conversely, while the Raspberry Pi 5 demonstrates acceptable accuracy metrics, such as a mAP50-95 of 0.8620 for YOLOv8s in FP32, its inference latency limits its applicability. The device achieves only 7.32 FPS and 217 inferences per minute in this configuration, significantly reducing the real-time capabilities. In addition to optimizing performance, quantization techniques have a significant impact on the energy efficiency of employing devices. The results demonstrate that INT8 is the most efficient configuration in terms of energy consumption, achieving a notable reduction without a significant loss of accuracy. In this context, the Orin NX, despite its higher overall power demand, demonstrates a better performance in the EPI metric (J/inference). This makes it the preferred choice for real-time, inference-intensive applications, especially in the FP16 and INT8 configurations. The Raspberry Pi 5 has the lowest total power consumption; however, its low performance in terms of Inferences Per Minute (IPM) restricts its applicability in environments where a real-time response is required. This analysis highlights the importance of selecting optimized hardware devices, such as the Orin models, for edge architectures that demand both high performance and sustainable energy efficiency, particularly in long-running applications where controlled power consumption is essential.

The evaluation of the stability and performance of the devices within architectures that emulate real-world operating environments highlights the advantage of the Orin models over general-purpose alternatives such as the Raspberry Pi 5. In practical scenarios, the models evaluated on the Orin Nano achieved acceptable results; however, a detailed analysis suggests that improvements can still be made in frame capture and the optimization of internal operations, which could reduce the processing overhead. This overhead, identified by analyzing isolated model processing times during the full operation of the architecture, suggests that better management of data capture and data flow could significantly improve the efficiency.

The Orin NX exhibited low variability in its response times and strong stability, positioning it as optimal for tasks demanding a consistent performance throughout the architecture. Conversely, the Raspberry Pi 5 encountered issues with instability, processing interruptions, and control challenges during object detection, constraining its effectiveness in complex architectures. Thus, it is more appropriate for basic processing or simpler algorithms rather than advanced applications requiring sustained high stability and low latency.

In this way, the results reveal complementary strengths in the tested devices: the Jetson excels in its computational performance but consumes significant energy, while the Raspberry Pi offers excellent energy efficiency but struggles with real-time object detection. Hybrid architectures could combine these strengths to optimize the overall performance. For example, the Raspberry Pi could manage lightweight tasks, such as environmental monitoring or data preprocessing, while the Jetson handles critical operations like real-time object detection or obstacle avoidance. This delegation of tasks reduces the energy footprint of the individual components while maintaining the system's responsiveness. Additionally, selectively integrating cloud processing for non-critical tasks enhances the scalability and reduces latency. Exploring this approach in future implementations could help address challenges in energy-sensitive drone missions.

The cloud results underscore the advantage of deploying edge devices near data sources. Despite the low model processing latency in the cloud, the high communication latency limits its real-time viability. Using a closer AWS region may improve the results, but communication bottlenecks remain a challenge for real-time drone applications. GPU-equipped edge devices offer a practical solution, enabling rapid local processing with

some trade-offs in the computational power, showing the importance of the role of edge devices in real-time drone systems.

Based on the results obtained and from the perspective of this study, focused on drone applications, a guide has been developed to identify the most suitable hardware configurations and quantization models according to the operational needs. This guide, based on the lessons learned, provides recommendations for selecting devices and model configurations based on the specific requirements, such as extended coverage, real-time tracking, and high-precision detection.

Table 7 illustrates how particular configurations align with specific requirements in drone applications. For extended coverage and long-duration missions, the Orin NX with INT8 offers the optimal autonomy with minimal inference consumption, making it ideal for continuous surveillance and environmental monitoring. In critical tracking scenarios, the same device with YOLOv8n INT8 enables a high inference rate, which is valuable for real-time tasks in defense and security. The Orin Nano and FP16 quantization enable high-quality detection, which is a requirement in automated infrastructure inspection, while maintaining the optimal power consumption. The INT8 configuration of the Orin Nano allows low-demand tasks to be completed efficiently and cost-effectively, even when speed or accuracy is not critical. These results demonstrate the value of aligning the configuration with specific operational demands, ensuring that each device performs at its best in its designated application. These insights serve as practical guidance for drone operators and practitioners, helping them select the most appropriate hardware and quantization settings based on the specific operational needs. Researchers should examine scalability in more UAV scenarios (e.g., evaluating the outdoor performance variability or investigating energy-efficient inference to provide insights and new extended guidelines for real-world deployment). Moreover, adapting the results obtained in the indoor controlled edge environment to outdoor surroundings would assess the stability of inferences under dynamic conditions, thereby verifying the practicality of energy-efficient setups outside controlled contexts. Furthermore, as the industry advances in the hardware–software co-design for FPGAs and low-power chips, research should incorporate these technologies to align with emerging capabilities.

Table 7. Recommended configurations based on operational needs.

Operational Need	Description	Area of Interest	Recommended Configuration	Key Supporting Metrics
Extensive Coverage and Long Duration	Extends energy autonomy for continuous monitoring of large areas without frequent recharging.	Surveillance, Environmental Monitoring	Jetson Orin Nano + YOLOv8n INT8	High FPS (44.9), Low Energy Consumption (7.483 W·s), Good Accuracy (mAP50: 0.8272)
Critical Tracking (Real-Time)	High inference rate for rapid response and continuous tracking.	Defense, Security	Jetson Orin NX + YOLOv8n INT8	Very High FPS (65.83), Moderate Energy (10.305 W·s), Sufficient Accuracy (mAP50: 0.7190)
High-Quality Detection	Ensures high precision in detecting fine details, ideal for infrastructure inspection.	Infrastructure Inspection	Jetson Orin Nano + YOLOv8s FP16	High Accuracy (mAP50: 0.8622), Moderate FPS (37.9), Low Energy Consumption (7.836 W·s)

Table 7. Cont.

Operational Need	Description	Area of Interest	Recommended Configuration	Key Supporting Metrics
Balance of Speed and Precision	Balances speed and accuracy for emergency and rescue tasks.	Rescue and Emergency Response	Jetson Orin NX + YOLOv8s FP16/YOLOv8n INT8	High FPS (58.82/65.83), Balanced Accuracy (mAP50: 0.8620/0.7190), Efficient Energy Usage (10.853/10.305 W·s)
Low-Cost Processing	Optimizes costs and energy for less demanding tasks without high precision or speed requirements.	Low-Demand Tasks	Jetson Orin Nano + INT8	Moderate FPS (41.2), Low Energy Consumption (7.257 W·s), Acceptable Accuracy (mAP50: 0.7968)

7. Conclusions

This research provides practical tools and guidelines for integrating YOLOv8 models into autonomous systems while also evaluating their performance on computationally constrained drone computational devices. The findings show that the YOLOv8n (nano) and YOLOv8s (small) models can achieve efficient operation on devices such as the Raspberry Pi 5, Orin NX, and Jetson Orin Nano. However, achieving the optimal performance in real-time applications requires careful consideration of the trade-offs between the detection precision and inference speed, especially when utilizing INT8 quantization, which accelerates the processing but may slightly reduce the accuracy.

Thus, we provide a set of recommendations (Table 6) that streamlines complex recommendations into an easily accessible format for decision-makers to use to adjust their configurations according to the specific operational demands. By linking the device configurations and quantization methods with specific applications, this study explores the practical application of guidelines that enhance the performance and energy efficiency. This organized method clarifies the criteria for the optimal configuration selection while enabling practitioners to efficiently implement these advances in diverse edge computing scenarios.

The test architecture in this study facilitates controlled and repeatable testing of real-time object detection and tracking algorithms. It enables an accurate evaluation of edge computing and onboard processing solutions within a comprehensive drone architecture in an indoor flight environment. However, the indoor setting introduces certain limitations, as it cannot fully replicate the complexities of outdoor operations, such as variable lighting conditions, environmental interferences, or multi-agent dynamics. To address these challenges, more complex mock-ups are needed to emulate more complex environments. Such improvements would enable testing under conditions closer to those in real-world applications, improving the reliability of the proposed systems.

An interesting direction for upcoming research is the evaluation of YOLOv10 and YOLOv11 as they become available, evaluating their improvements in accuracy, efficiency, and real-time performance. Comparing them with YOLOv8 in drone-based applications could provide insights into their suitability for edge computing and its integration viability in autonomous drone navigation.

Moreover, studies should explore hybrid architectures that combine the complementary strengths of devices like the Jetson, the Raspberry Pi, and the cloud. As mentioned in Section 6, Discussion, its complementary abilities may offer a study opportunity. Testing these configurations in real-world drone deployments would provide valuable insights into their feasibility and effectiveness in energy-sensitive missions.

Further integration of edge architectures with 5G and tactical cloud technologies could expand the potential of drone applications. This integration would enable the design and control of complex swarm missions, as discussed in Section 2, The State of the Art. Through network slicing, the processing tasks could be decentralized, increasing the autonomy of individual drones within the swarm and reducing reliance on a central server. Moreover, testing 5G communications in these configurations could provide insights into intelligent drone architectures, particularly regarding real-time decision-making and minimizing the latency in data transmission. These advancements would also necessitate the deployment of larger drones capable of supporting the additional weight and power requirements associated with outdoor operations in complex environments.

Author Contributions: Conceptualization: A.M.B. Methodology: A.M.B. and D.C. Software: L.R. and A.D.D. Validation: L.R., D.C. and A.D.D. Formal analysis: J.A.B., L.B. and L.R. Investigation: L.R. and D.C. Data curation: L.R. and A.D.D. Resources: L.B. and J.A.B. Writing—original draft preparation: L.R., A.M.B., A.D.D. and L.B. Writing—review and editing: all. Visualization: L.R. Supervision: A.M.B., J.A.B. and J.R.C. Project administration: A.M.B. Funding acquisition: J.R.C. and A.M.B. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by the Horizon Europe EDF Programme grant number 101103386 and under the national grants TSI-063000-2021-80 and MIA.2021. M04.0008 by the Spanish Ministry of Economic Affairs and Digital Transformation and by the EU Next Generation EU/PRTR programme. MCIN/AEI/10.13039/501100011033 under Grant PID2020-118249RB-C21.

Data Availability Statement: The raw data supporting the conclusions of this article will be made available by the authors on request.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Amato, G.; Ciampi, L.; Falchi, F.; Gennaro, C. Counting Vehicles with Deep Learning in Onboard UAV Imagery. In Proceedings of the 2019 IEEE Symposium on Computers and Communications (ISCC), Barcelona, Spain, 29 June–3 July 2019; pp. 1–6. [CrossRef]
2. Brown, A.; White, B. Autonomous Navigation in Drones Using Deep Learning. In Proceedings of the International Conference on Autonomous Systems, Barcelona, Spain, 13–17 March 2023; pp. 456–467. [CrossRef]
3. Schilling, F.; Lecoeur, J.; Schiano, F.; Floreano, D. Learning Vision-Based Flight in Drone Swarms by Imitation. *IEEE Robot. Autom. Lett.* **2019**, *4*, 4523–4530. [CrossRef]
4. Holly, S.; Wendt, A.; Lechner, M. Profiling Energy Consumption of Deep Neural Networks on NVIDIA Jetson Nano. In Proceedings of the 2020 11th International Green and Sustainable Computing Workshops (IGSC), Pullman, WA, USA, 19–22 October 2020; pp. 1–6. [CrossRef]
5. Sacco, A.; Esposito, F.; Marchetto, G.; Montuschi, P. A Self-Learning Strategy for Task Offloading in UAV Networks. *IEEE Trans. Veh. Technol.* **2022**, *71*, 4301–4311. [CrossRef]
6. Reis, D.; Kupec, J.; Hong, J.; Daoudi, A. Real-Time Flying Object Detection with YOLOv8. *arXiv* **2024**, arXiv:2305.09972.
7. Jocher, G.; Qiu, J.; Chaurasia, A. Ultralytics YOLO, Version 8.0.0. Online Resource, 2023. Available online: <https://github.com/ultralytics/ultralytics> (accessed on 29 January 2025).
8. McEnroe, P.; Wang, S.; Liyanage, M. A Survey on the Convergence of Edge Computing and AI for UAVs: Opportunities and Challenges. *IEEE Internet Things J.* **2022**, *9*, 15435–15459. [CrossRef]
9. Baller, S.P.; Jindal, A.; Chadha, M.; Gerndt, M. DeepEdgeBench: Benchmarking Deep Neural Networks on Edge Devices. *arXiv* **2021**, arXiv:2108.09457.
10. Stolfi, D.H.; Danoy, G. An Evolutionary Algorithm to Optimise a Distributed UAV Swarm Formation System. *Appl. Sci.* **2022**, *12*, 10218. [CrossRef]
11. Li, X.; Gong, X.; Wang, D.; Zhang, J.; Baker, T.; Zhou, J.; Lu, T. ABM-SpConv-SIMD: Accelerating Convolutional Neural Network Inference for Industrial IoT Applications on Edge Devices. *IEEE Trans. Netw. Sci. Eng.* **2023**, *10*, 3071–3085. [CrossRef]
12. Ferraz, O.; Araujo, H.; Silva, V.; Fernandes, G.F.P. Benchmarking Convolutional Neural Network Inference on Low-Power Edge Devices. In Proceedings of the ICASSP 2023—2023 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), Rhodes Island, Greece, 4–10 June 2023; pp. 1–5. [CrossRef]

13. Hou, X.; Guan, Y.; Han, T.; Zhang, N. DistrEdge: Speeding up Convolutional Neural Network Inference on Distributed Edge Devices. In Proceedings of the 2022 IEEE International Parallel and Distributed Processing Symposium (IPDPS), Lyon, France, 30 May–3 June 2022; pp. 1097–1107. [\[CrossRef\]](#)
14. Liang, S.; Wu, H.; Zhen, L.; Hua, Q.; Garg, S.; Kaddoum, G.; Hassan, M.; Yu, K. Edge YOLO: Real-Time Intelligent Object Detection System Based on Edge-Cloud Cooperation in Autonomous Vehicles. *IEEE Trans. Intell. Transp. Syst.* **2022**, *23*, 25345–25360. [\[CrossRef\]](#)
15. Shin, D.J.; Kim, J.J. A Deep Learning Framework Performance Evaluation to Use YOLO in Nvidia Jetson Platform. *Appl. Sci.* **2022**, *12*, 3734. [\[CrossRef\]](#)
16. Xiong, G.; Qi, J.; Wang, M.; Wu, C.; Sun, H. GCGE-YOLO: Improved YOLOv5s Algorithm for Object Detection in UAV Images. In Proceedings of the 2023 42nd Chinese Control Conference (CCC), Tianjin, China, 24–26 July 2023; pp. 7723–7728. [\[CrossRef\]](#)
17. Hu, Y.; Wang, L.; Kou, T.; Zhang, M. YOLO-Tiny-attention: An Improved Algorithm for Fault Detection of Wind Turbine Blade. In Proceedings of the 2023 8th International Conference on Intelligent Computing and Signal Processing (ICSP), Xi'an, China, 21–23 April 2023; pp. 1228–1232. [\[CrossRef\]](#)
18. Xu, C. 5G for drone networking. *Trans. Emerg. Telecommun. Technol.* **2022**, *33*, e4668. [\[CrossRef\]](#)
19. Cai, Y.; Yue, L.; Zhu, M.; Lei, M.; Zhang, J.; Hua, B.; Luo, W.; Zou, Y.; Tian, L.; Ma, L.; et al. Photonics-Assisted Millimeter-Wave Communication System Based on Low-Bit Gaussian Mixture Model Adaptive Vector Quantization. *IEEE Photonics J.* **2022**, *14*, 1–9. [\[CrossRef\]](#)
20. Coelho, A.; Fontes, H.; Campos, R.; Ricardo, M. Placement and Allocation of Communications Resources in Slicing-aware Flying Networks. *arXiv* **2021**, arXiv:2112.07048.
21. Jiang, H.; Li, Q.; Li, Y. Post Training Quantization after Neural Network. In Proceedings of the 2022 14th International Conference on Computer Research and Development (ICCRD), Shenzhen, China, 7–9 January 2022; pp. 1–6. [\[CrossRef\]](#)
22. Przewlocka-Rus, D.; Sarwar, S.S.; Sumbul, H.; Li, Y.; Salvo, B.D. Power-of-Two Quantization for Low Bitwidth and Hardware Compliant Neural Networks. *arXiv* **2022**, arXiv:2203.05025. [\[CrossRef\]](#)
23. Al-Hamid, A.A.; Kim, H. Unified Scaling-Based Pure-Integer Quantization for Low-Power Accelerator of Complex CNNs. *Electronics* **2023**, *12*, 2660. [\[CrossRef\]](#)
24. Gupta, K.; Asthana, A. Reducing the Side-Effects of Oscillations in Training of Quantized YOLO Networks. *arXiv* **2023**, arXiv:2311.05109. [\[CrossRef\]](#)
25. Zhou, M.; Guo, Y.; Fu, J.; Cai, H. Power-aware Quantization Circuits in Analog In-Memory Computing with STT-MRAM Macro. In Proceedings of the 2023 IEEE International Magnetic Conference—Short Papers (INTERMAG Short Papers), Sendai, Japan, 15–19 May 2023; pp. 1–2. [\[CrossRef\]](#)
26. Carramiñana, D.; Braga, L.R.; Bernardos, A.M.; Dobrzycki, A.D.; Bergesio, L.; Besada, J.A.; Casar, J.R. A Digital Twin Concept to Prototype Multi-Drone Based Applications. In Proceedings of the 14th International Conference on the Internet of Things (IoT 2024), Oulu, Finland, 19–22 November 2024.

Disclaimer/Publisher’s Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.