electronics

MDPI

Article

Neural Network Implementation for Fire Detection in Critical Infrastructures: A Comparative Analysis on Embedded Edge Devices

Jon Aramendia [1,*,†], Andrea Cabrera [1,†], Jon Martín [1,*,†], Jose Ángel Gumiel [1] and Koldo Basterretxea [2]

[1] Fundación Tekniker, 20600 Eibar, Spain; acabrera@tekniker.es (A.C.); joseangel.gumiel@tekniker.es (J.Á.G.)
[2] Department of Electronics Technology, University of the Basque Country (UPV/EHU), 48013 Bilbao, Spain; koldo.basterretxea@ehu.eus
* Correspondence: jon.aramendia@tekniker.es (J.A.); jon.martin@tekniker.es (J.M.)
† These authors contributed equally to this work.

Abstract: This paper explores the application of artificial intelligence on edge devices to enhance security in critical infrastructures, with a specific focus on the use case of a battery-powered mobile system for fire detection in tunnels. The study leverages the YOLOv5 convolutional neural network (CNN) for real-time detection, focusing on a comparative analysis across three low-power platforms, NXP i.MX93, Xilinx Kria KV260, and NVIDIA Jetson Orin Nano, evaluating their performance in terms of detection accuracy (mAP), inference time, and energy consumption. The paper also presents a methodology for implementing neural networks on various platforms, aiming to provide a scalable approach to edge artificial intelligence (AI) deployment. The findings offer valuable insights into the trade-offs between computational efficiency and power consumption, guiding the selection of edge computing solutions in security-critical applications.

Keywords: edge AI; energy-efficient AI; low-power embedded systems; critical infrastructure protection; fire and smoke detection; YOLOv5

Academic Editor: Domenico Rosaci

Received: 1 April 2025
Revised: 17 April 2025
Accepted: 22 April 2025
Published: 29 April 2025

Citation: Aramendia, J.; Cabrera, A.; Martín, J.; Gumiel, J.Á.; Basterretxea, K. Neural Network Implementation for Fire Detection in Critical Infrastructures: A Comparative Analysis on Embedded Edge Devices. *Electronics* **2025**, *14*, 1809. https://doi.org/10.3390/electronics14091809

1. Introduction

Critical infrastructure (CI) is essential for societal stability, economic growth, and public safety. Ensuring their protection has become increasingly urgent for most current governments, or the European Union which has launched, among others, the TESTUDO project [1] for the autonomous swarm of heterogeneous resources in infrastructure protection via threat prediction and prevention. Road transport is considered an important critical infrastructure in several countries [2] and its disruption can lead to severe economic consequences and human casualties. Among the various risks associated with road transport, tunnels present particular hazards. One of the most serious threats to both material assets and human lives is fire, which can escalate quickly in these confined spaces, making it especially dangerous.

Traditional sensor-based detection systems analyze the chemical properties of smoke to trigger alarms but can be prone to false alerts or may trigger alarms too late. Moreover, these systems may not be effective in large areas, wild environments, or high-temperature conditions, where they could miss important signals [3].

Several systems based on wireless sensor networks (WSNs) integrate temperature, humidity, gas, and flame sensors to detect early signs of combustion in rural areas. These

systems typically transmit data to a web server using mobile networks like GPRS and TCP/IP protocols, offering real-time alerts. Although promising, WSN-based systems often lack visual confirmation and spatial granularity, underscoring the need to combine traditional sensing with AI-driven computer vision for more robust fire monitoring [4]. In this context, two paradigms are typically considered for video analytics [5]: cloud computing, where inference is offloaded to remote servers [6], and edge computing, where processing occurs locally on the device [5]. However, relying on streaming video to perform detection on remote servers can be computationally expensive and impractical in certain scenarios. In environments where network coverage may be limited or unreliable, continuous high-bandwidth video transmission is not always feasible. In such cases, it is more efficient to perform detection directly on embedded surveillance systems, which can process video locally and send only relevant events and detections to the server, reducing bandwidth usage.

One challenge in implementing AI-based vision systems in embedded devices is the high computational requirements of convolutional neural networks (CNNs), which results in significant power consumption and slower inference time. This makes it particularly difficult to deploy such models in battery-powered embedded systems, where energy efficiency is crucial. As a result, one of the main research in the field focuses on developing techniques to optimize and adapt these models for deployment in low-power embedded devices, ensuring reliable real-time detection even in harsh environment scenarios like the one described earlier.

This paper presents a performance comparison of the YOLOv5 (You Only Look Once, version 5) convolutional neural network (CNN) for real-time fire and smoke detection on critical roadway infrastructures, with a focus on deployment in battery-powered embedded systems. The primary objective of this study is to determine which of three distinct processing platforms—NXP i.MX93 (which integrates a microprocessor and a neural processing unit, or NPU), Xilinx Kria KV260 (which combines a microprocessor with a field-programmable gate array, or FPGA), and NVIDIA Jetson Orin Nano (which features a microprocessor and a graphics processing unit, or GPU)—is the most efficient for running the YOLOv5 model, considering key metrics such as energy consumption, mean average precision (mAP), and inference time.

Several researchers have explored the benchmarking of embedded neural networks, often evaluating performance metrics such as accuracy or inference speed. However, there remains a clear gap in the literature regarding power consumption, which is a key concern in embedded systems. Moreover, to the best of our knowledge, no prior study has conducted a comparative analysis of FPGA, NPU, and GPU platforms under the same experimental conditions—although these accelerators have been studied individually [7–9].

While prior research has demonstrated the feasibility of deploying CNNs like YOLOv5 for fire detection [10], existing studies often overlook critical trade-offs in real-world embedded deployments. Most works focus on optimizing performance for a single hardware platform in isolation or neglect to quantify energy efficiency under standardized workloads despite its importance for battery-operated systems [11]. Furthermore, comparative analyses of heterogeneous accelerators (NPUs, FPGAs, and GPUs) in fire detection scenarios remain limited, particularly for infrastructure monitoring where latency, accuracy, and power constraints intersect.

Using a standardized methodology and assessment tools, the study aims to provide a detailed comparison of these platforms' performance in executing the deep learning model. By evaluating mAP, inference time, and energy consumption jointly, this work addresses a gap in the literature and offers actionable insights for deploying AI in energy-constrained edge applications. This research contributes to understanding how different hardware

architectures, such as system-on-chip (SoC) designs that combine a microprocessor with specialized accelerators, impact performance, and efficiency. These insights help guide the selection of optimal platforms for battery-powered, real-time embedded applications.

The paper is structured in the following manner: Section 2 provides a comprehensive review of the state-of-the-art, focusing on the object detection models used for fire detection and the hardware used to accelerate convolutional neural networks on embedded devices. Section 3 details the materials and methods, including descriptions of the model, dataset, hardware platforms, training, and methods used. Section 4 presents the results, offering a comparative analysis of platform performance in terms of energy consumption, mAP, and inference time. Section 5 presents a discussion of the results of the study, summarizing key findings and proposing directions for future research. Subsequently, Section 6 offers the conclusions.

2. State of the Art: Advances in Fire Detection and CNN Hardware Acceleration

2.1. AI-Based Fire Detection

With the growing adoption of AI in safety-critical applications, deep learning-based fire detection has received significant attention in recent years. CNNs, especially those from the YOLO family, have been widely applied in this domain due to their real-time object detection capabilities. Studies have shown YOLO variants achieving high detection accuracy on fire and smoke datasets, but often rely on high-performance, power-hungry GPU systems [3,12,13]. While these results are promising, such platforms are impractical for many real-world deployments where size, cost, and energy constraints are paramount. To address these limitations, the field is shifting toward edge computing—bringing inference closer to the data source using compact, energy-efficient hardware. This is especially relevant in IoT and smart infrastructure scenarios such as tunnel monitoring, where fast response times and autonomous operation are essential.

Several lightweight models and optimizations have been proposed for embedded platforms, with implementations on NPUs (e.g., i.MX series), FPGAs (e.g., Xilinx DPUs), and GPUs (e.g., NVIDIA Jetson family) [14–17]. However, these studies often focus on a single platform or specific model optimization and lack a unified comparison under consistent workloads. The systematic evaluation of such edge platforms—known as edge AI benchmarking—has thus become an emerging research priority [18]. Unlike benchmarking in cloud or HPC environments, edge benchmarking must contend with platform heterogeneity, resource constraints, and deployment variability. Recent surveys have classified benchmarking efforts into explicit tools (e.g., EdgeBench [19], pCAMP [20], AIoTBench [21]) and implicit studies using custom performance tests [22,23]. These benchmarks evaluate metrics such as inference time, energy consumption, and model accuracy, but they rarely target security-critical use cases or perform comparative analysis across diverse accelerator types.

In particular, there is a gap in benchmarking studies that

- Compare multiple low-power hardware accelerators (NPU, FPGA, GPU) under the same model and workload conditions;
- Focus on fire and smoke detection for infrastructure protection;
- Analyze the trade-offs between detection accuracy (mAP), inference latency, and power consumption in realistic embedded scenarios.

This paper addresses that gap by benchmarking the YOLOv5 object detection model across three heterogeneous platforms: NXP i.MX93 (NPU), Xilinx Kria KV260 (FPGA), and NVIDIA Jetson Orin Nano (GPU). The goal is to provide a comparative evaluation that

informs the selection of edge AI hardware for deployment in mobile, battery-powered systems designed for tunnel fire detection.

2.2. Hardware Accelerators for CNN

When deploying neural networks in embedded systems, hardware accelerators are a significant research focus to improve performance while maintaining energy efficiency [18]. Several works have focused on the use of hardware accelerators to efficiently execute neural networks for object detection in embedded systems.

2.2.1. NPU

NPUs have emerged as specialized hardware accelerators designed to optimize deep learning tasks. Unlike traditional CPUs and GPUs, NPUs are tailored for efficient execution of neural network operations, such as matrix multiplications and convolutions. Their architecture improves performance and energy efficiency, making them a good option for AI-driven applications in mobile devices, autonomous systems, and embedded devices.

Several studies have explored the use of YOLO-based models on embedded platforms accelerated by NPUs, particularly in mobile or energy-constrained environments. These works are relevant to this study due to their emphasis on low-power object detection using variants of the YOLO architecture. For example, a human detection system was deployed on an embedded platform with an i.MX NPU [24]. In the context of unmanned aerial vehicles (UAVs), YOLOv3-Tiny was executed on an NPU-based platform achieving a mAP of 0.591 with only 6 million parameters [25], while another UAV implementation using YOLOv3 reached a mAP of 0.897 [26]. These scores demonstrate the feasibility of deploying object detection on low-power hardware with a lightweight architecture.

2.2.2. FPGA and DPU

The inherent parallelism of the FPGA makes them a very good option at the time of accelerating CNNs, and it benefits the inference of a deep learning model by reducing execution time and enhancing program accuracy.

FPGAs are used to describe the hardware and provide the opportunity to design custom acceleration engines at the hardware level using description languages. There are works that design their own dedicated engines such as Jinguji et al. [27] that compared a YOLO model in an FPGA (with its own accelerator engine) and a mobile GPU that achieved x3.85 higher FPS speed and x2.00 better FPS/W power consumption in the FPGA.

A major limitation of FPGA-based solutions is their relatively long development time compared to software-based implementations, due to the need for hardware description languages and specialized design flows. This complexity has limited the accessibility of FPGAs for many developers. To address this, FPGA vendors have introduced dedicated IP cores for neural network acceleration, such as Xilinx's deep learning processor unit (DPU). These DPUs are configurable, pre-designed components that allow designers to tailor resource usage according to application needs. Thanks to their balance between performance and ease of integration, DPUs are increasingly adopted in computer vision applications [28–30].

The widespread adoption of Xilinx DPUs in computer vision has led to numerous practical implementations in embedded edge systems. Two examples are particularly relevant due to their focus on real-time performance and efficient resource utilization—key concerns in fire detection scenarios. A notable case is a multi-task ADAS detection system implemented on the Xilinx KV260 platform using a B4096 DPU, which achieved 25.4 FPS while maintaining a power consumption of only 7.19 W [17]. This highlights the DPU's capacity to handle multiple inference tasks under power constraints. Another study deployed a video analytics application on an FPGA equipped with two B4096 DPUs,

resulting in a 33-fold increase in video processing throughput compared to a software GPU-accelerated baseline [16]. These cases exemplify the advantages of DPU-accelerated FPGA solutions in real-time embedded vision workloads, supporting their relevance for benchmarking fire detection models on similar platforms.

2.2.3. GPU

Graphics processing units (GPUs) have gained relevance in embedded systems to accelerate deep learning tasks, in particular convolutional neural networks (CNNs). Their parallel processing capabilities enable efficient execution of complex computations, making them suitable for real-time applications in edge computing environments.

Several studies have demonstrated the effectiveness of GPUs in embedded platforms. For example, Farooq et al. [31] evaluated thermal imaging on embedded GPU platforms for vehicular assistance systems. They trained YOLOv5 variants on a novel thermal dataset and deployed optimized models on devices like the NVIDIA Jetson Nano and Xavier NX. The TensorRT-optimized network achieved 11 frames per second (FPS) on the Jetson Nano and 60 FPS on the Xavier NX, highlighting the potential of GPUs in processing thermal images for real-time applications.

Another study by Machado et al. [32] analyzed the power consumption of the NVIDIA Jetson Nano board while performing object classification using YOLOv5 models. The results indicated that the YOLOv5n variant outperformed others in terms of throughput, achieving 12.34 FPS, and showed low power consumption, consuming 0.154 mWh per frame, in other words, about 0.55 Joules per inference.

Rey et al. [33] conducted a comprehensive benchmark of YOLOv8n and YOLOv8s models across various quantization levels (FP32, FP16, INT8) on GPU- and CPU-based platforms, including the NVIDIA Jetson Orin Nano, Orin NX, and Raspberry Pi 5. Their research focused on drone-based applications, evaluating detection accuracy, inference speed, and energy consumption. While their study provides valuable insights into the performance of different YOLOv8 models on these platforms, our research diverges by concentrating solely on the YOLOv5n model in INT8 quantization. Moreover, we extend the comparison to include diverse hardware accelerators—FPGA, NPU, and GPU—within the specific context of fire and smoke detection in critical infrastructure environments.

These studies underscore the versatility and efficiency of GPUs in embedded systems, particularly for applications requiring real-time data processing and object detection.

3. Materials and Methods

3.1. Model

The selection of the YOLOv5n model in this study is motivated by its proven efficiency and compatibility with a wide range of embedded hardware platforms. Previous studies have highlighted the limitations of alternative lightweight models such as SSD-MobileNet and EfficientDet on resource-constrained edge devices. Specifically, MobileNet-based models struggle to achieve real-time performance on devices like the Jetson Nano and Raspberry Pi 3B+ [5]. Similarly, EfficientDet poses deployment challenges on platforms like the NXP i.MX8M Plus due to its computational complexity and difficulty with INT8 quantization [34]. In contrast, YOLOv5n offers an architecture that is well-suited to INT8 quantization and has demonstrated stable, real-time performance across FPGA, NPU, and GPU platforms. This makes it a strong candidate for edge inference tasks in critical infrastructure scenarios, ensuring consistency across all target hardware.

YOLOv5 [35] is a convolutional neural network designed for object detection in real time by Ultralytics. It is part of the YOLO family, a series of computer vision models known for their speed and accuracy. YOLOv5, despite not being developed by the original creators of YOLOv1-v4, has become highly popular due to its ease of use and high performance.

The YOLOv5 network has several variants that differ in terms of model size and that affect accuracy and inference time. This means that smaller models are faster but less accurate, while larger models are slower but more accurate. Those variants are YOLOv5n, YOLOv5s, YOLOv5m, YOLOv5l, and YOLOv5x. For this project, since the goal is to implement the network on embedded systems with limited resources and low power consumption, the YOLOv5n variant has been implemented. Table 1 shows the different YOLOv5 variants with their size, inference time on a V100, and their accuracy on COCO.

Table 1. YOLOv5 variants with their size, inference time on a V100 and their accuracy on COCO [35].

Model	Size (MB_{FP16})	Parameters (M)	mAP_{COCO}
YOLOv5n (Nano)	4	1.9	28.0
YOLOv5s (Small)	14	7.2	37.4
YOLOv5m (Medium)	41	21.2	45.4
YOLOv5l (Large)	89	46.5	49.0
YOLOv5x (XLarge)	166	86.7	50.7

To enable the model to be implemented across all platforms, some changes have been made to the model's structure. First, the original SiLU activation functions are replaced with LeakyReLU activation functions with a negative slope of 26/256 to ensure compatibility across all platforms while maintaining acceptable performance. For the model to be implemented on the FPGA, a second modification is required. As the Vitis-AI compilation and quantization tool does not support the permute and view functions, the last layer of the detection head will be removed from the model, and this part will later be implemented on the CPU.

This step was carried out manually as the Vitis-AI toolchain does not currently support specific PyTorch v.1.13.1 operations such as permute and view. As a result, the final detection head had to be detached from the model graph and handled in post-processing on the CPU. This represents a platform-specific intervention, in contrast to the largely automated pipelines used for the i.MX93 and Jetson Orin Nano.

As a result, the network runs with modified activation functions on both the Orin Nano and i.MX93 platforms, while on the KV260, it not only has the modified activations but also lacks the final detection layer, which is instead executed on the CPU. This means that inference execution times on the KV260 depend not only on the DPU but also partially on the CPU.

3.2. Dataset and Training

The dataset that had been used for fire detection is D-Fire [36]. This dataset contains a total of 21,527 images and annotations in YOLO format. For each image, there is a corresponding txt file with the annotations for that image. The dataset has two classes, named 'fire' and 'smoke', so the resulting network will be capable of detecting fire and smoke. Of the 21,527 images in the dataset, 1164 images contain only fire, 5867 contain only smoke, 4658 contain both smoke and fire, and finally, there are 9838 images that contain neither. In total, across all images, there are 14,692 fire predictions and 11,865 smoke predictions. Table 2 presents the distribution of images by category in the D-Fire dataset.

The dataset was divided into three main subsets: training (17,221 images, 80%), validation (3306 images, 15%), and testing (1000 images, 5%). In addition to this standard split, an additional "evolve" folder was created, which was further subdivided into three subsets: training (900 images), validation (100 images), and testing (100 images). The images in the "evolve" set are copies of the original dataset, but each of them contains at least one detection. Table 3 shows the distribution of the dataset subsets.

The training process was carried out using the YOLOv5 tools on a high-performance computer with an NVIDIA GeForce RTX 4080 SUPER. The training was performed using the default hyper-parameters provided by the YOLOv5 training libraries. Table 4 presents the precision values obtained with the trained model. These values were calculated using YOLO's evaluation libraries on the validation subset of the dataset.

Table 2. Distribution of images by category in the D-Fire dataset.

Category	Images	Percentage
Fire only	1164	5.4%
Smoke only	5867	27.3%
Fire & Smoke	4658	21.6%
No fire/No smoke	9838	45.7%
Total	**21,527**	**100%**

Table 3. Distribution of the D-Fire dataset subsets, including the number of images and their corresponding percentage of the total.

Subset	Images	Percentage
Training	17,221	80%
Validation	3306	15%
Testing	1000	5%
Total	**21,527**	**100%**

Table 4. Performance metrics of YOLOv5n trained with D-Fire and measured on validation subset.

Class	Image	Instances	Precision	Recall	mAP@50	mAP@50:95
all	3301	4076	76.7	70.7	77.3	42.0
smoke	3301	1691	84.2	70.9	79.7	45.9
fire	3301	2385	69.1	70.5	74.7	38.0

3.3. Embedded Platforms

In this study, we aim to evaluate the performance of a neural network implementation on three different hardware platforms: NXP i.MX93, Xilinx Kria KV260, and NVIDIA Jetson Orin Nano. These platforms were chosen to obtain a heterogeneous and representative mix of computing architectures, including ARM-based CPUs, FPGA-based accelerators, and embedded systems equipped with NPUs and GPUs. Each platform offers unique capabilities and optimization opportunities for the deployment of machine learning models, which makes them suitable candidates for comparative analysis. The primary goal of this section is to provide a detailed description of the hardware used.

3.3.1. Kria KV260

The KV260 is a hardware platform built for the implementation of artificial vision systems with convolutional neural networks. The board consists of two different parts: the central system-on-module (SoM) and the carrier card.

The K26 SoM is optimized for edge vision applications that require flexibility to adapt to changing requirements. It is based on the Zynq UltraScale+ MPSoC (XCK26) architecture, which consists of two parts: the processing system (PS), which includes, among other components, a Quad-core Cortex-A53, a Dual-core Cortex-R5F, and an Arm Mali-400 MP2, and the Programmable Logic (PL), which contains a specifically designed FPGA for the K26 SoM.

For neural network inference acceleration, a Xilinx B4096 DPU is implemented in the SoC's FPGA. The DPU has a DPUCZDX8G_ISA1_B4096 architecture and a clock speed of 300 MHz. This DPU communicates with the PS part of the SoC via AXI and can achieve 1.2 TOPS.

3.3.2. i.MX93

The Variscite VAR-SOM-MX93 Development Kit is used to work with the i.MX93. This hardware features a VAR-SOM-MX93 SoM as its central unit, which integrates the NXP i.MX93 processor. This embedded processor consists, among other components, of an Arm Cortex-A55 MPCore cluster and an Arm Cortex-M33 platform, providing a balance between high-performance computing and low-power real-time processing for embedded applications.

Additionally, the i.MX93 integrates the Arm Ethos-U65 (256) NPU, which is designed to efficiently accelerate machine learning workloads in AI-on-the-edge applications. This NPU can perform at 1GHz up to 512 GOPS [37].

3.3.3. Jetson Orin Nano

For the GPU implementation, a Jetson Orin Nano is used. It has an Arm® Cortex®-A78AE v8.2 64-bit CPU, which serves as the central processing unit. This six-core, 64-bit v8.2 processor provides an optimal balance between computational performance and energy efficiency.

To accelerate neural network inference, the Jetson Orin Nano integrates an NVIDIA Ampere-based GPU with 1024 CUDA cores and 32 Tensor cores. This GPU architecture is designed to optimize parallel computation and deep learning workloads, being able to achieve a maximum performance of 67 TOPS at INT8 precision [38].

3.4. Conversion and Quantization

To deploy the model on the selected hardware platforms, it was necessary to convert the original model into compatible formats for each architecture. This process involved both model quantization and format conversion to optimize performance while maintaining detection accuracy. In this study, post-training quantization (PTQ) was used across all platforms, as it provides a toolchain-compatible and hardware-agnostic solution. However, we acknowledge that more advanced model-aware techniques, such as quantization-aware training (QAT), could help reduce accuracy degradation by better adapting the model during training to low-precision arithmetic. The following subsections describe the methodology used for each platform to transform the YOLOv5n model into an implementable model.

3.4.1. Kria KV260

To run the YOLOv5n network on the KRIA KV260, the model must first be quantized. For this process, the vai_q_pytorch tool is used, a Python package from the Vitis-AI Quantizer designed as an extension for PyTorch. This package allows the quantized model to be exported in various formats, including XMODEL, PyTorch, and ONNX. However, since XMODEL cannot be run on a PC to calculate the accuracy of the quantized model, ONNX is used instead.

Once quantized, the model must be compiled into a format that can be executed on the FPGA's DPU. Since there are different types of DPUs and they are configurable, the model compilation must take into account the specific configuration of the target DPU. This compilation step uses the vai_c_xir package, another Python package from Vitis-AI, which uses the provided DPU fingerprint and the quantized XMODEL to generate a new XMODEL with executable operations tailored to the specific DPU configuration.

3.4.2. i.MX93

The model must first be exported to TFLite format to run the YOLOv5n network on the i.MX93 NPU. Additionally, the model needs to be quantized to utilize uint8 data types, ensuring compatibility with hardware acceleration. The YOLOv5 framework is used to export and quantify the model. Finally, the quantized model must be converted into a format supported by the Ethos-U NPU integrated into the i.MX93. This conversion is performed using the Vela optimization tool, which exports a new TFLite with operations that are executable on the NPU.

TensorFlow 2.12 has been observed to provide optimal inference performance with YOLOv5n and was specifically chosen after detecting incorrect bounding box outputs when using more recent versions such as 2.15 and 2.17. This issue occurred during inference after converting the model from PyTorch or ONNX format to TensorFlow format, resulting in visibly inaccurate detection results that affected model reliability. Although a systematic benchmark across TensorFlow versions was not conducted, switching to version 2.12 consistently resolved the problem, producing correct and stable outputs. This behavior has also been reported by other developers in community forums [39], which further supports the selection of TensorFlow 2.12 as a stable baseline for deployment on the i.MX93 platform.

3.4.3. Jetson Orin Nano

To run a model on the NVIDIA Jetson Orin Nano GPU, the model must first be exported to ONNX, then quantized and exported to the engine format. Although the GPU supports various formats such as PT or ONNX, the engine format has been used, as it is NVIDIA's optimized format for executing models on the GPU.

The FP32 model provides higher precision. However, it results in a larger network size, which may result in longer inference time or higher power consumption. Conversely, the INT8 model reduces network size and may provide faster inference times, but it can also lead to a decrease in precision. For the comparison in this article, the INT8 network was chosen, as the other platforms require network quantization.

To export the model to ONNX, the YOLOv5 framework is used. Then, the polygraphy library is used to quantize and convert the network to engine format. For this task, a representative dataset is used for image calibration in the quantization process.

3.5. Model Accuracy Analysys

Precision is a key metric for evaluating the performance of object detection models such as YOLOv5n. In real-world applications, a high-precision score ensures that detected objects are correctly classified, minimizing false positives. This is particularly important in scenarios where misclassification can have significant consequences, such as fire detection. The mAP is the standard metric for assessing object detection models, as it provides a comprehensive measure of both precision and recall.

Precision measures the proportion of correctly detected objects among all detections, indicating how reliable the model is in minimizing false positives. It is defined as:

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} \tag{1}$$

where TPs (true positives) are correctly detected objects, and FPs (false positives) are incorrect detections. Recall, on the other hand, quantifies the model's ability to detect all relevant objects, reducing false negatives, and is given by:

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}} \tag{2}$$

where FNs (false negatives) are objects that were missed by the model. High precision ensures fewer false alarms, while high recall ensures that most objects are detected.

The mAP is computed by averaging the average precision (AP) across all classes, integrating the precision–recall curve. A higher mAP indicates a better balance between accurate detections and minimal false positives, ensuring the model performs consistently across different object categories.

To evaluate the performance of our YOLOv5n model implementations on each platform, we used PyCocoTools, a library designed to compute object detection metrics following the COCO evaluation standard. PyCocoTools provides a standardized method to compute precision, recall, and mAP at different intersection over union (IoU) thresholds, ensuring a robust assessment of model performance. We calculate mAP@50 (mAP at IoU = 0.5) and mAP@50:95 (mean AP over IoU thresholds from 0.5 to 0.95 in steps of 0.05) in three phases: with the trained model, the quantized model, and the model implemented on each hardware platform.

For the first two phases (trained and quantized models), we use the built-in YOLO libraries to compute mAP. However, to evaluate performance on hardware platforms, we developed a custom application using PyCocoTools to measure mAP. During inference on the hardware platforms, detection results are saved as txt files in YOLO format, containing the predicted bounding boxes. These predictions are then compared to the ground truth annotations to compute the final mAP, allowing for an objective assessment of the model's performance across different deployment environments.

3.6. Power Consumption Analysys

Power consumption is a critical factor in embedded AI platforms, as these devices often operate under strict energy restrictions in edge computing environments. To evaluate the power efficiency of these three embedded AI platforms—Xilinx Kria KV260, NXP i.MX93, and NVIDIA Jetson Orin Nano—a rigorous and reproducible framework is established to quantify both idle and active power consumption during inference execution. A controlled experiment was conducted, in which each device performed 1000 inference operations while real-time power measurements were logged. This approach ensured consistency, minimized variability, and provided a direct, data-driven comparison of energy efficiency across the platforms evaluated.

To ensure consistent and accurate power measurements, we employed an Agilent Technologies N6705B DC Power Analyzer equipped with an N6752A high-performance DC power module. The embedded devices were powered directly from the analyzer, allowing for precise monitoring of real-time current (I) in Amperes. Power (P) and energy consumption values were then computed based on the logged current and voltage data. It is important to note that the measurements include the total power consumption of the embedded system, comprising both the system on module (SoM) and the carrier board. As such, the reported power values reflect the consumption of the entire device rather than isolating the SoC or accelerator core. This approach provides a realistic view of the power requirements under typical deployment conditions but may differ from studies that isolate specific power rails or components.

The analyzer was configured to log real-time current (I) in Amperes, which made it possible to calculate power and energy values. The power consumption was calculated using the relationship below, where $P(t)$ is the instantaneous power, V is the known supply voltage, and $I(t)$ is the measured current at time t.

$$P(t) = V \cdot I(t) \tag{3}$$

The total energy consumption over time was then obtained by integrating the power over the execution period, where Δt represents the sampling interval of the power analyzer:

$$E = \sum P(t) \cdot \Delta t \tag{4}$$

The concept of energy per inference E_{i} measures the average amount of energy required to complete a single inference. In this context, the total energy consumption (E) includes the inference stage, as well as the associated pre-processing and post-processing steps. To calculate the energy per inference, the total measured energy is divided by the total number of inferences performed (N_{i}):

$$E_{\mathrm{i}} = \frac{\sum P(t) \cdot \Delta t}{N_{\mathrm{i}}} = \frac{E}{N_{\mathrm{i}}} \tag{5}$$

Before running the AI inference task, we measured the baseline (idle) power consumption to establish a reference. Next, we activate the inference process and record the power consumption under load (active state). By comparing these two measurements, we isolate the power consumed by the AI workload itself. This procedure is repeated for all 1000 inferences to compute average and total energy values.

For each device, the following key metrics were extracted from the logged data:

- **Execution Time** (Δt, s): Total duration of the inference task.
- **Total Energy Consumption** (J): Energy consumed during the entire execution period.
- **Energy per Inference** (J): Average energy required for a single inference.
- **Active Power Consumption** (W): Average power drawn during inference execution.
- **Idle Power Consumption** (W): Average power drawn when no inference is running.
- **Peak Power Consumption** (W, max & min): Maximum and minimum observed power spikes during execution.

These metrics allow a comparative analysis of energy efficiency across the three embedded platforms.

4. Results

To compare the performance of the YOLOv5n model across the three selected hardware platforms, we analyze results based on key metrics: detection accuracy, inference time, and power consumption. As the test conditions were standardized across all devices, the following subsections present a detailed breakdown of each performance dimension, highlighting trade-offs between computational efficiency and energy use.

4.1. Precision Metrics

Due to the different quantization and conversion methods applied to adapt the YOLOv5n models for each hardware platform, variations in precision have been observed. These differences stem from how each platform handles numerical representation, optimization techniques, and computational constraints, which can impact the accuracy of object detection.

The precision metrics of the YOLOv5 model will be evaluated in three phases for each model on each hardware platform. The first phase assesses the trained model, the second evaluates the quantized model, and the final phase measures the performance of the converted model. Both trained and quantized models are executed on the PC's CPU using the YOLO framework, while the converted models are executed on each target hardware using the corresponding framework. However, for the Jetson Orin Nano, since its adaptation process simultaneously quantizes and converts the model, only the trained model can be

tested on the PC's CPU, while the quantized and converted model is evaluated directly on the hardware.

To measure accuracy in mAP, the YOLO framework is used for models executed on the computer, while a custom application that uses pycocotools is used for models executed on the embedded platforms. These differences in the execution and the mAP measurement methods between the PC and the embedded boards may cause slight discrepancies between the results obtained on the PC and those obtained on the platforms.

Table 5 presents the precision metrics (mAP) of the YOLOv5 model evaluated in three stages: trained, quantized, and converted. The results show that quantization leads to a reduction in mAP, which is expected because of the lower numerical precision of weights and activations. The impact of quantization varies, but in all cases, the converted model retains a similar performance to the quantized model, indicating that adaptation to the target hardware does not introduce significant additional losses. The mAP@50:95 values show a more pronounced decrease compared to mAP@50, highlighting the increased difficulty of maintaining high precision across all IoU thresholds after quantization and conversion. Overall, these results confirm that quantization affects model precision, but the conversion process maintains the performance achieved after quantization.

Table 5. Object detection performance across different platforms: Comparison of mAP in trained, quantized, and converted models.

		Trained (PC)	Quantized (PC)	Converted (HW)
KV260	mAP@50	68.3	64.8	64.8
	mAP@50:95	35.7	30.8	31.0
i.MX93	mAP@50	68.3	62.3	63.6
	mAP@50:95	35.7	30	29.3
Jetson Orin Nano	mAP@50	68.3	–	66.3
	mAP@50:95	35.7	–	32.8

4.2. Inference Time

The inference time will be considered solely as the time during which the neural network is executed. The times for image pre-processing and post-processing (Non-Max Suppression and drawing bounding boxes) will be excluded. This choice was made to isolate the pure inference latency and ensure a fair, hardware-level comparison across platforms, since pre- and post-processing steps can vary significantly depending on the specific deployment framework, optimization strategies, and hardware capabilities. However, we acknowledge that in real-time applications, total end-to-end latency is critical and will be addressed during deployment testing.

Since running the model on the FPGA required removing the last layer of the detection head and implementing it in software, the execution times will be measured separately for two cases: the full process, which includes both the DPU execution and the additional layer running on the CPU, and the standalone execution time of the DPU alone. Table 6 presents the inference time comparison across different embedded platforms.

Table 6. Inference time comparison across different embedded platforms.

Processor	Min (ms)	Max (ms)	Avg (ms)	Warm-Up (ms)
DPU	22.5	25.3	22.7	23.8
DPU + CPU	51.8	56.7	52.1	53.5
NPU	32.2	34.6	32.4	32.8
GPU	2.5	4.7	3.1	107.4

The results reveal notable inference time differences between platforms. The GPU demonstrates the lowest execution time, with an average of just 3.1 ms. However, its warm-up time is significantly higher at 107.4 ms, indicating that while the GPU excels in continuous inference, it may not be optimal for applications requiring intermittent execution. The DPU, when running standalone, offers a balanced execution time of 22.7 ms, which increases to 54.7 ms when the additional CPU processing is included.

The NPU shows intermediate performance with an average execution time of 32.4 ms, making it slower than the standalone DPU but faster than the full FPGA (DPU+CPU) inference. In both, the warm-up time is relatively low compared to the GPU, which could be advantageous in scenarios where fast initialization is required.

4.3. Consumption

The power consumption results for the evaluated platforms revealed notable differences in energy efficiency, execution time, and power consumption during the AI inference process. Measurements include total execution time, total energy consumed, energy per inference, average active and idle power consumption, and peak power consumption. The following sections examine each platform individually before presenting overarching conclusions at the end.

4.3.1. Kria KV260

During the execution of 1000 AI inferences, the KV260 recorded an execution time of 151.29 s, with a total energy consumption of 1323.53 J, resulting in an energy per inference of 1.32 J. The system consumed an average of 8.75 W while active and 8.12 W when idle. Peak power measurements ranged from 7.95 W (minimum) to 13.33 W (maximum), highlighting the platform's dynamic power fluctuations during inference execution. The Table 7 presents the Xilinx KV260 power consumption metrics.

Table 7. Xilinx Kria KV260 power consumption metrics.

Metric	Value
Execution Time (Δt, s)	151.29
Total Energy Consumption (J)	1323.53
Energy per Inference (J)	1.32
Active Power Consumption (W)	8.74
Idle Power Consumption (W)	8.12
Max Peak Power Consumption (W)	13.33
Min Peak Power Consumption (W)	7.95

It is important to note that, for consistency across platforms, the fan of the Kria KV260 was disconnected during energy measurements. This decision was made to isolate and accurately benchmark the power consumption of the SoC and the carrier board, which are the key components under evaluation. Including the fan would have introduced additional variability unrelated to computational performance, potentially obscuring meaningful comparisons between platforms. While this setup provides a fair and controlled environment for benchmarking, it does not fully represent typical deployment conditions, where the fan is normally active. The fan's power draw—measured separately at approximately 0.63 W—should be considered in practical use cases. Nonetheless, it is worth noting that the Kria KV260's thermal performance could be improved through alternative passive cooling solutions, such as enhanced heat sinks or heat-pipe-based designs. These strategies could eliminate the need for active cooling in certain embedded scenarios, making the platform more viable for energy-constrained deployments. We encourage system designers to weigh these thermal and power trade-offs based on the target use case.

Consumption data have been captured throughout the inference process, including idle states before and after algorithm execution, as well as detailed expanded views of power fluctuations over shorter time periods.

The Figure 1 represents the complete inference process, showing a transition from idle to active execution and back to idle. Power consumption starts at around 8.1 W during the idle phase and increases to an average of 8.75 W during inference, with frequent power spikes reaching up to 13.33 W. This fluctuation indicates dynamic workload variations as the FPGA processes different inference tasks.

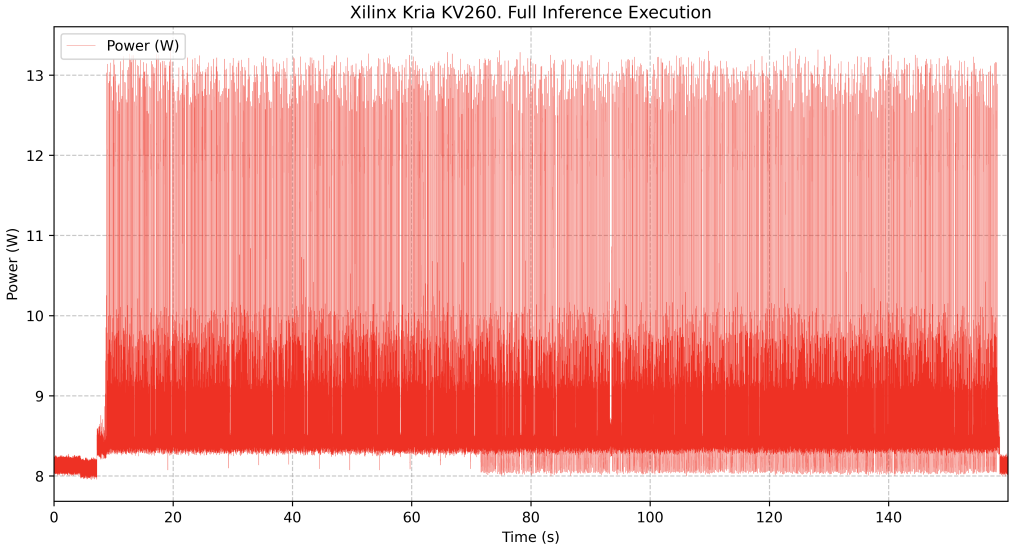

Figure 1. Power consumption of the Xilinx Kria KV260 during the full inference execution process.

The Figure 2 provides a closer look at a 1-s interval within the inference execution. Power consumption shows rapid fluctuations, with frequent spikes above 12 W. These oscillations suggest that power consumption varies significantly at the millisecond level, surely influenced by variations in computational load during individual inferences.

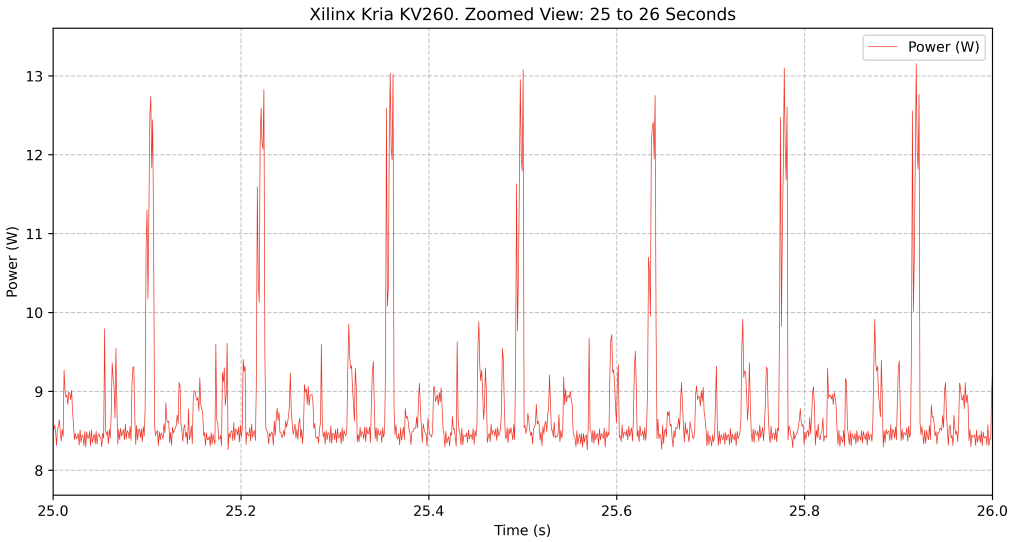

Figure 2. Zoomed-in power consumption of the Xilinx Kria KV260 within a 1-s interval during inference execution.

The Figure 3 further zooms in on a 200 ms segment, offering a high-resolution perspective of the short-term power variations. Here, individual power peaks become more distinguishable, highlighting the rapid transitions between lower-power and peak-power

states. This fine-grained view provides insight into the FPGA's workload scheduling and power management during inference execution.

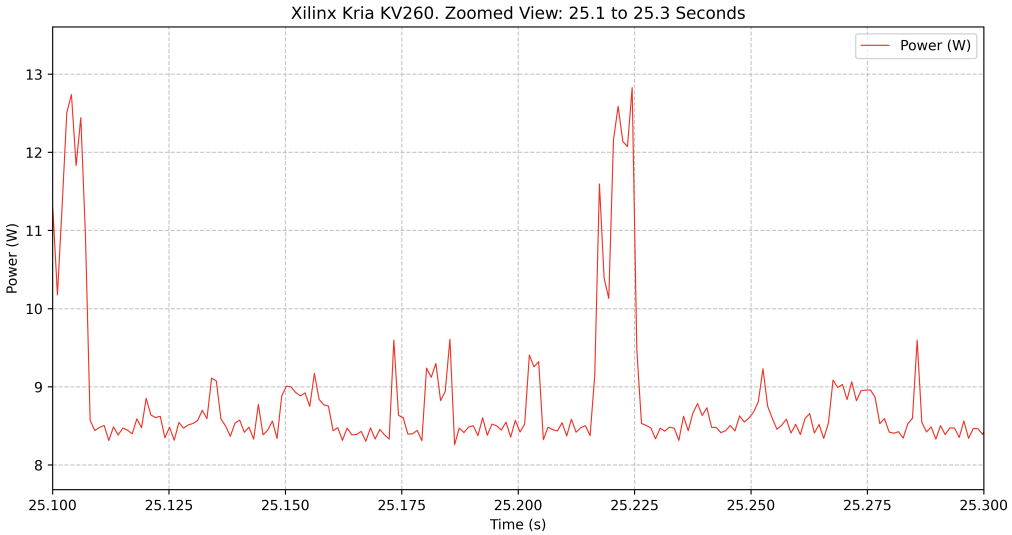

Figure 3. Detailed power consumption of the Xilinx Kria KV260 over a 200 ms interval.

4.3.2. i.MX93

The same test of 1000 AI inferences was performed on the NXP i.MX93, which recorded an execution time of 110.55 s, with a total energy consumption of 236.82 J, resulting in an energy per inference of 0.237 J. The system consumed an average of 2.14 W when active and 1.65 W when idle. Peak power measurements ranged from 1.55 W (minimum) to 3.17 W (maximum), demonstrating a low power profile compared to the other evaluated platforms. The Table 8 presents the NXP i.MX93 power consumption metrics.

Table 8. NXP i.MX93 power consumption metrics.

Metric	Value
Execution Time (Δt, s)	110.55
Total Energy Consumption (J)	236.81
Energy per Inference (J)	0.23
Active Power Consumption (W)	2.14
Idle Power Consumption (W)	1.64
Max Peak Power Consumption (W)	3.16
Min Peak Power Consumption (W)	1.54

The Figure 4 represents the complete inference process, showing a transition from idle to active execution and back to idle. The power consumption starts at around 1.65 W in the idle phase and increases to an average of 2.14 W during inference. The power fluctuations are relatively minor compared to the Kria KV260, with occasional peaks reaching 3.17 W. This steady power profile suggests that the i.MX93 prioritizes low-energy operation over peak performance.

The Figure 5 provides a closer look at a 1-s interval within the inference execution. Power consumption exhibits modest variations, fluctuating around 2.2–2.6 W, with occasional peaks slightly exceeding 2.8 W. Unlike the Kria KV260, which showed rapid and frequent spikes, the i.MX93 maintains a more consistent power consumption pattern, likely due to its efficient power management.

The Figure 6 further zooms in on a 200 ms segment, revealing short-term power variations. While there are small fluctuations, the power profile remains relatively stable, with fewer sharp peaks compared to other platforms. This suggests that the i.MX93

efficiently manages its power consumption, avoiding significant spikes. Unlike the Xilinx case, it is difficult to analyze when inferences occur.

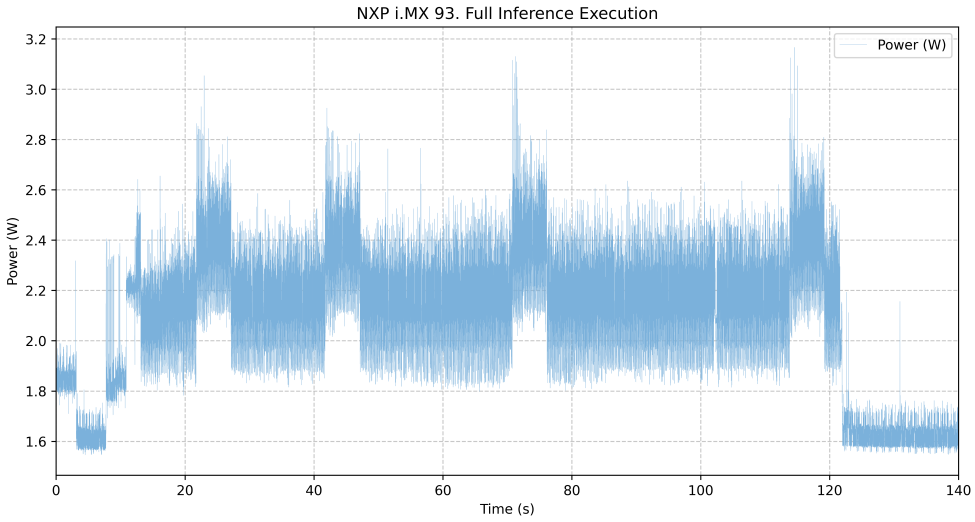

Figure 4. Power consumption of the NXP i.MX93 during the full inference execution process.

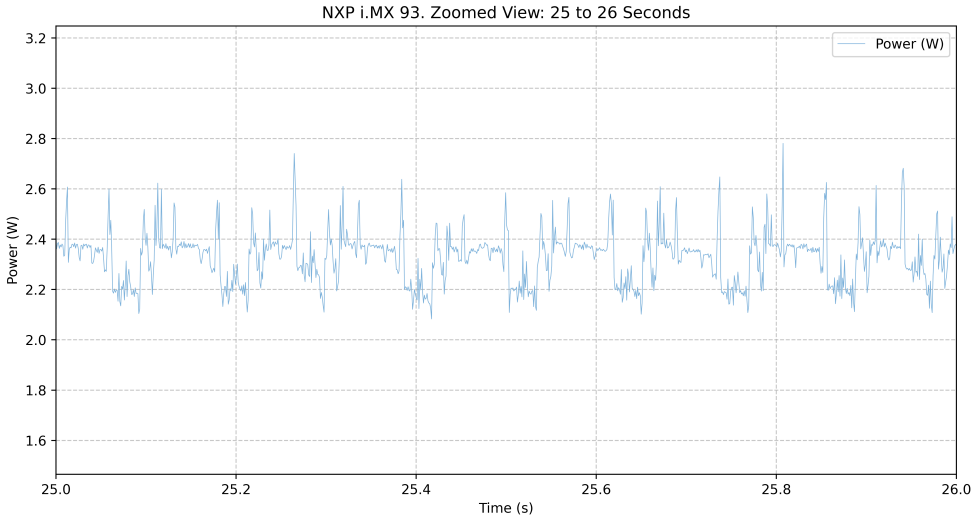

Figure 5. Zoomed-in power consumption of the NXP i.MX93 within a 1-s interval during inference execution.

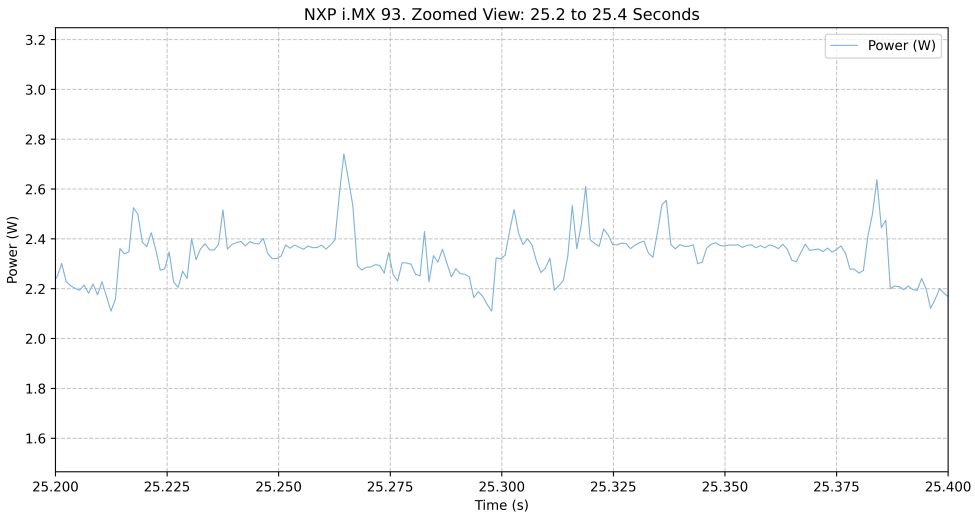

Figure 6. Detailed power consumption of the NXP i.MX93 over a 200 ms interval.

4.3.3. Jetson Orin Nano

When executing 1000 AI inferences, the NVIDIA Jetson Orin Nano took 35.72 s to complete, consuming 223.86 J of total energy, resulting in 0.224 J per inference. This result aligns closely with the findings of [33], who reported an energy consumption of 0.185 J per inference for the YOLOv8n model on the same hardware platform. The system exhibited an average active power consumption of 6.27 W and an idle power consumption of 3.77 W. During execution, power usage ranged from 3.57 W to a peak of 11.44 W, indicating significant variations in power demand during inference. The Table 9 presents the NVIDIA Jetson Orin Nano power consumption metrics.

Table 9. NVIDIA Jetson Orin Nano power consumption metrics.

Metric	Value
Execution Time (Δt, s)	35.72
Total Energy Consumption (J)	223.85
Energy per Inference (J)	0.22
Active Power Consumption (W)	6.26
Idle Power Consumption (W)	3.76
Max Peak Power Consumption (W)	11.43
Min Peak Power Consumption (W)	3.57

Consumption data have been recorded throughout the inference process, including idle states before and after execution, as well as detailed zoomed-in views of power fluctuations over shorter time periods.

The Figure 7 represents the complete inference process, showing a transition from idle to active execution and back to idle. Power consumption starts at around 3.77 W in the idle phase and increases to an average of 6.27 W during inference, with frequent spikes reaching up to 11.44 W.

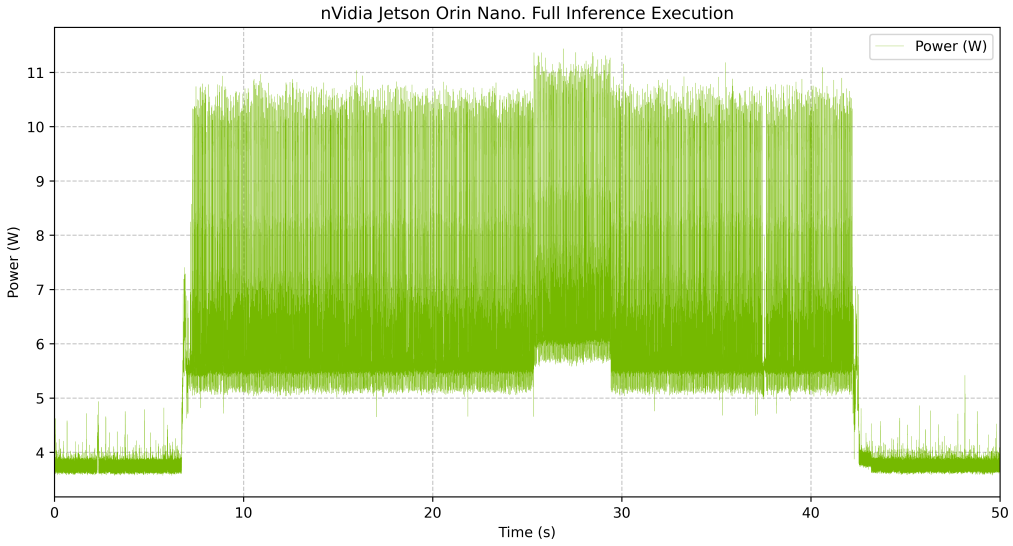

Figure 7. Power consumption of the NVIDIA Jetson Orin Nano during the full inference execution process.

The Figure 8 provides a closer look at a 1-s interval within the inference execution. The power draw exhibits high-frequency oscillations, with multiple peaks exceeding 10 W. This suggests a highly dynamic power profile, likely due to rapid bursts of computation when processing individual inferences.

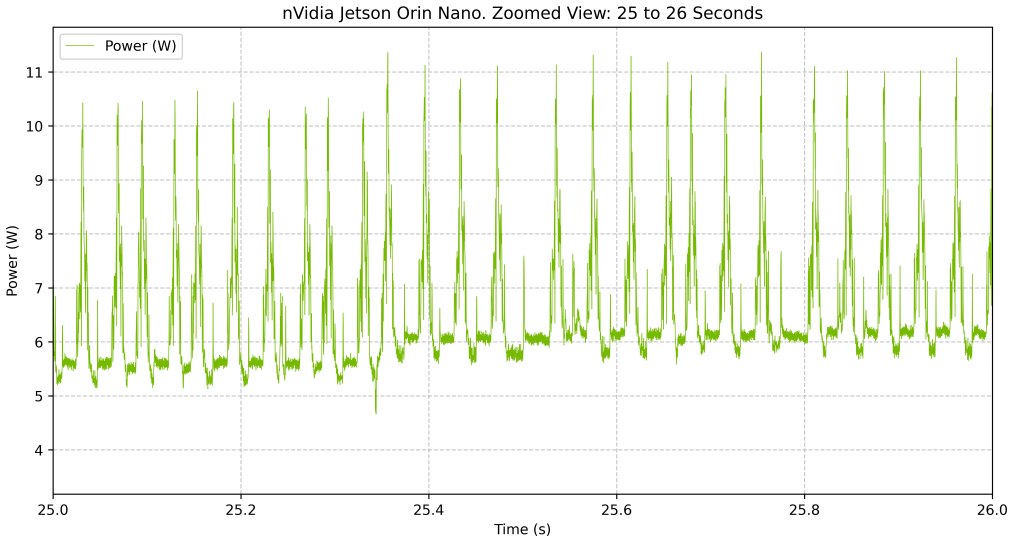

Figure 8. Zoomed-in power consumption of the NVIDIA Jetson Orin Nano within a 1-s interval during inference execution.

The Figure 9 further zooms in on a 200 ms segment, providing insight into the short-term power variations. Unlike the NXP i.MX93, which demonstrated a stable power profile, the Jetson Orin Nano exhibits frequent and sharp spikes. This behavior suggests an aggressive power management approach; the device dynamically scales power consumption based on real-time processing needs. Unlike i.MX93, this model makes it very easy to identify when an inference is taking place and to distinguish between pre-processing and post-processing computing.

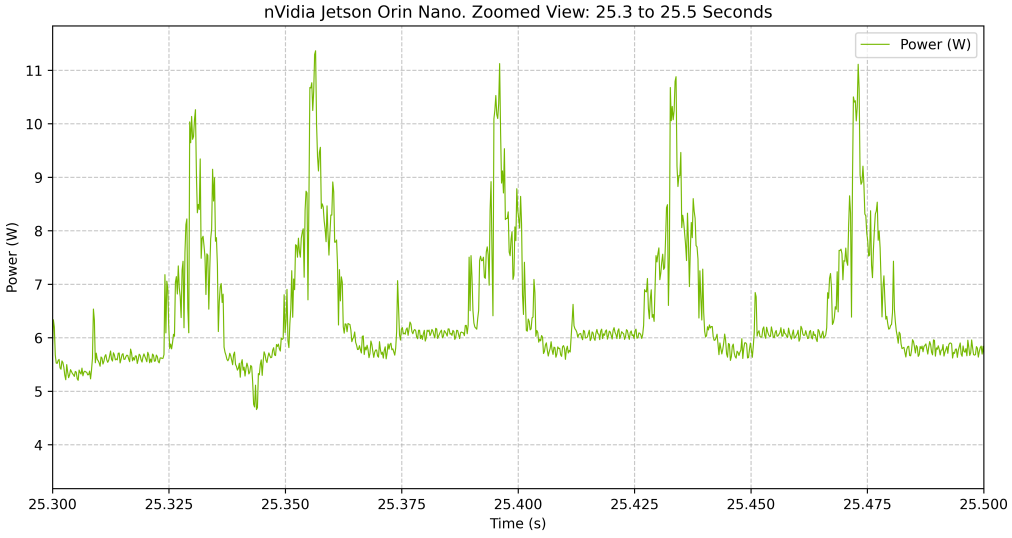

Figure 9. Detailed power consumption of the NVIDIA Jetson Orin Nano over a 200 ms interval.

4.3.4. Comparison and Conclusions

The power consumption characteristics of the Xilinx Kria KV260, NXP i.MX93, and NVIDIA Jetson Orin Nano are summarized Table 10.

Figure 10 provides a global view of the power consumption for each platform throughout the entire inference execution. The Xilinx Kria KV260 has the highest power consumption, averaging 8.75 W in active mode, with peaks exceeding 13 W. The NVIDIA Jetson Orin Nano exhibits a lower average power draw of 6.27 W, yet frequent fluctuations and peaks

up to 11.44 W suggest aggressive power scaling. In contrast, the NXP i.MX93 maintains the lowest energy profile, averaging 2.14 W with minimal variations.

Table 10. Power consumption comparison metrics.

Metric	Xilinx	NXP	NVIDIA
Execution Time (Δt, s)	151.29	110.55	35.72
Total Energy Consumption (J)	1323.53	236.81	223.85
Energy per Inference (J)	1.32	0.23	0.22
Active Power Consumption (W)	8.74	2.14	6.26
Idle Power Consumption (W)	8.12	1.64	3.76
Max Peak Power Consumption (W)	13.33	3.16	11.43
Min Peak Power Consumption (W)	7.95	1.54	3.57

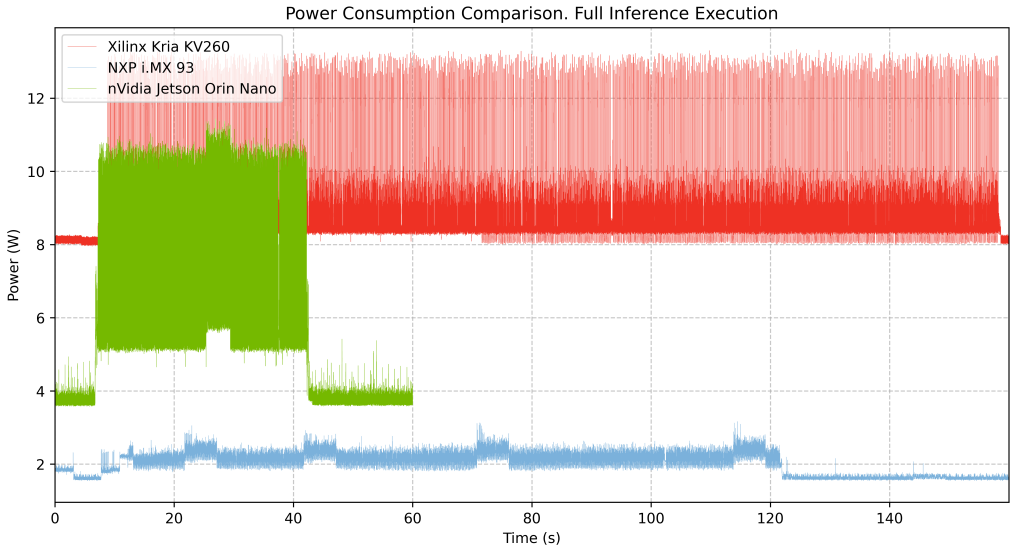

Figure 10. Power consumption comparison of all platforms during full inference execution.

Beyond power consumption, execution time varies widely between platforms. The Jetson Orin Nano completes the 1000 inferences in 35.72 s, significantly outperforming both the i.MX93 (110.55 s) and Kria KV260 (151.29 s). This difference directly affects total energy consumption, where the KV260 expends 1323.53 J, nearly six times the 223.86 J required by the Jetson Orin Nano for the same workload. The NXP i.MX93, despite its lower power draw, consumes 236.82 J, slightly more than the Jetson Orin Nano due to its longer execution time.

Energy per inference is a key metric for evaluating efficiency, which quantifies how much energy is required for each AI computation. NVIDIA Jetson Orin Nano proves to be the most energy-efficient, consuming only 0.22 J per inference, closely followed by NXP i.MX93 at 0.24 J. Xilinx Kria KV260, however, consumes significantly more energy per inference (1.32 J), highlighting its higher energy cost per operation.

Figure 11 presents a zoomed-in comparison over a 1-s interval, revealing power fluctuations at a finer scale. The KV260 maintains a high baseline power level with moderate fluctuations, while the Jetson Orin Nano exhibits frequent, high-amplitude power spikes. This suggests a dynamic power scaling strategy where power consumption is rapidly adjusted based on computational demand. NXP i.MX93, in contrast, remains remarkably stable, exhibiting small variations in power draw.

A further high-resolution view of a 200 ms interval is provided in Figure 12. The Jetson Orin Nano continues to exhibit frequent sharp peaks, with power draw exceeding 10 W at regular intervals. The Kria KV260 remains relatively stable at a high-power level,

with occasional fluctuations, while the NXP i.MX93 maintains a consistently low power profile, reinforcing its suitability for energy-constrained applications.

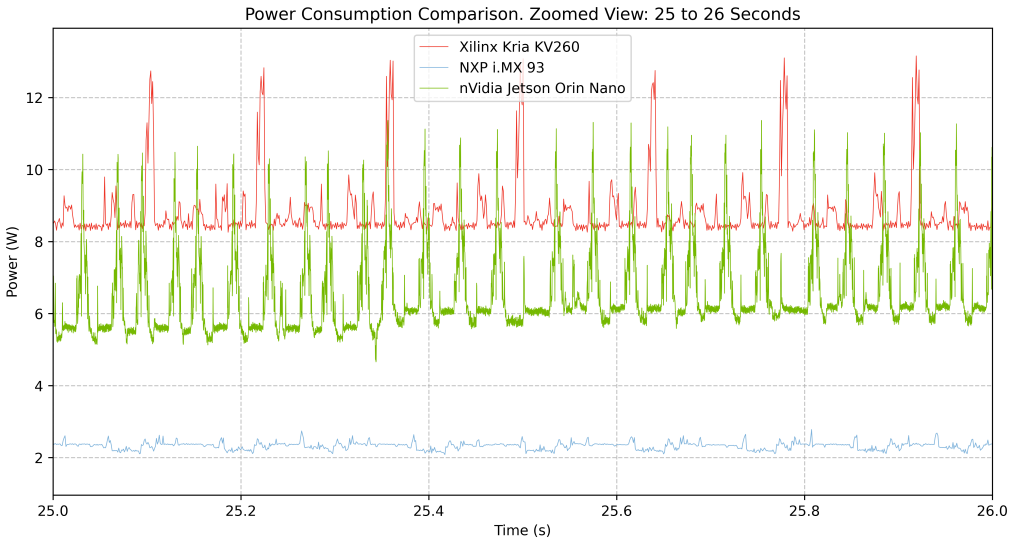

Figure 11. Zoomed-in power consumption comparison over a 1-s interval.

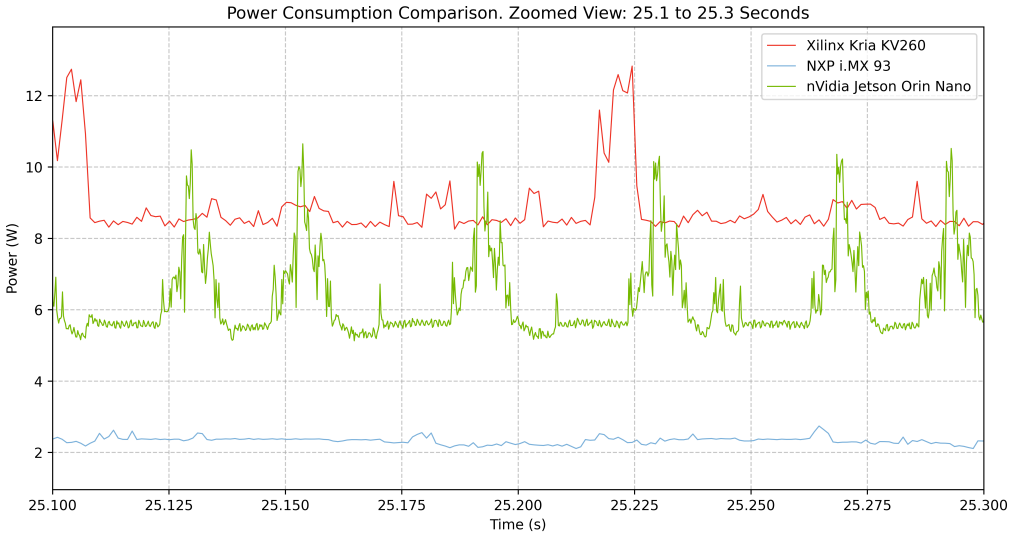

Figure 12. Detailed power consumption comparison over a 200 ms interval.

These results highlight the trade-offs between performance, power consumption, and energy efficiency across the evaluated platforms. The Xilinx Kria KV260, while delivering FPGA-based acceleration, operates with the highest power demand and energy cost per inference. The NVIDIA Jetson Orin Nano is a high-performance solution that completes inference faster while maintaining competitive power efficiency. The NXP i.MX93, although slower than the Jetson, demonstrates exceptional energy efficiency, making it an optimal choice for ultra-low-power applications.

5. Discussion

The findings of this study provide valuable insights into the deployment of the YOLOv5n deep learning model for fire and smoke detection on embedded edge devices, highlighting the trade-offs between detection accuracy, inference time, and power consumption. The study underscores the importance of selecting hardware based on the specific constraints and requirements of the target application.

One of the key observations is the impact of quantization on detection accuracy. The evaluation of precision metrics across different stages of the model adaptation process shows that quantization is the main factor contributing to a reduction in accuracy. This decrease is expected, as quantization reduces the numerical precision of weights and activations to improve computational efficiency. However, once the model has been quantized, the subsequent conversion to the target hardware does not introduce significant additional losses. This suggests that the optimization frameworks used for hardware adaptation effectively preserve the performance of the quantized model. These methods could help improve the post-quantization performance of models deployed on NPUs and DPUs, bridging the gap between precision and computational efficiency.

Another important factor is power consumption. The i.MX93 demonstrated exceptional energy efficiency, consuming only a mean active power consumption of 2.14W, making it an ideal candidate for applications requiring long-term operation in battery-powered systems. In contrast, the KV260 exhibited the highest mean active power consumption (8.74 W). It is important to note that the power consumption values reported in this study correspond to the evaluation boards with their respective carrier cards.

Regarding the Kria KV260, it is worth noting that the fan was disconnected during measurements to avoid distortion in energy comparisons. While this setup reflects an idealized case, the thermal requirements of the platform could be addressed through improved passive cooling, potentially avoiding the need for active components in embedded scenarios.

Inference speed is a critical factor in real-time fire detection systems, and the results reveal significant differences between the evaluated platforms. The Jetson Orin Nano achieved the fastest execution time, with an average of 3.1 ms per inference, making it highly suitable for latency-sensitive applications. However, its high warm-up time (107.4 ms) may limit its efficiency in scenarios where sporadic or event-driven inference is required. The i.MX93, while significantly slower (32.4 ms average), demonstrated more consistent performance and low energy consumption, making it a viable option for battery-powered applications where power efficiency is prioritized over speed. The Kria KV260 FPGA, when inference was performed solely on the DPU, achieved an average execution time of 22.7 ms, but when the required CPU processing was included, the total time increased to 54.7 ms. The entire YOLOv5n model could not be executed entirely on the FPGA and required additional processing on the CPU.

Although the primary evaluation in this study focused on the YOLOv5n model due to its lightweight architecture and suitability for low-power deployment, we also tested the YOLOv5s variant. However, it required more than twice the inference time for only a 2.4% gain in mAP, which reinforced the decision to select YOLOv5n as the most balanced model for this benchmarking. The decision to focus on a single-model architecture was intentional, aiming to isolate the influence of hardware characteristics under consistent application conditions. Unlike other networks such as MobileNet or EfficientDet, YOLOv5 was selected for its compatibility and deployability across all three hardware platforms without major structural modifications, ensuring a fair and reproducible comparison.

The model has been trained only with the D-Fire dataset. This dataset is sufficient to compare the effectiveness of the three platforms, but if the model were to be deployed in a real use case, its detection capabilities would need to be improved. The accuracy data do not reflect how the model would perform in a real-world environment. Accuracy could be improved by adding more datasets or generating custom datasets. The most effective approach would be to create custom datasets with images that closely resemble the environment and conditions in which the model would operate.

Despite the fact that the system has not yet been tested with real tunnel footage, a deployment in a controlled real-world environment is planned as part of this project, in

collaboration with a public entity responsible for road infrastructure in the Basque Country. Due to the critical nature of these facilities, further details cannot be disclosed, but the tests will provide valuable validation under realistic conditions.

6. Conclusions

This study evaluated the implementation of the YOLOv5n neural network for fire and smoke detection in critical infrastructures between three embedded edge platforms: Xilinx Kria KV260 (DPU), NXP i.MX93 (NPU), and NVIDIA Jetson Orin Nano (GPU). The comparative analysis focused on key performance metrics, including detection accuracy, inference time, and power consumption, to assess the trade-offs between computational efficiency and energy efficiency.

The results highlight significant differences in platform performance. The NVIDIA Jetson Orin Nano achieved the best mAP@50 accuracy and the highest inference speed, with an average execution time of 3.1 ms, making it the most suitable choice for very fast real-time applications. In contrast, the Xilinx Kria KV260 demonstrated competitive accuracy and inference time, but exhibited the highest mean active power consumption (8.74W), making it less efficient for low-power applications. The NXP i.MX93, while slower than the Jetson Orin Nano, proved to be the most energy-efficient platform, consuming only 2.14W in active mode, reinforcing its suitability for battery-powered or resource-constrained environments.

These results suggest that the optimal platform choice depends on the specific application constraints. The Jetson Orin Nano is preferable for real-time, high-performance scenarios, while the i.MX93 is ideal for energy-constrained environments. If we consider the specific use case of fire and smoke detection using battery-powered embedded systems, the best option among the three platforms would be the i.MX93. This is mainly because it is the most power-efficient.

Although the Jetson Orin Nano achieves better results in terms of accuracy and speed, these factors are not the most critical. If the goal is to detect and alert the presence of fire in tunnels, a few milliseconds will not make a significant difference, nor will having extremely precise localization of the fire. However, the ability of the system to operate for extended periods of time without draining the battery could be a deciding factor.

While this study focused on benchmarking embedded platforms for fire and smoke detection, several critical aspects remain for future exploration to support real-world deployment. Model life-cycle management—particularly in the context of emerging regulations such as the European Cyber Resilience Act—will be addressed by integrating secure, lightweight update mechanisms like SWUpdate. Additionally, we aim to assess the model's robustness under challenging visual conditions, including occlusion, noise, and low lighting, using targeted robustness metrics. Beyond fire detection, the proposed benchmarking methodology is applicable to other infrastructure monitoring scenarios, such as intrusion or structural anomaly detection, leveraging the same energy—accuracy–latency trade-off framework to guide broader edge AI adoption. These directions will help bridge the gap between controlled evaluation and deployable, resilient, and scalable solutions.

Author Contributions: Methodology, J.A., A.C. and J.Á.G.; Software, J.A. and A.C.; Writing – original draft, J.A., A.C. and J.Á.G.; Supervision, J.M. and K.B.; Project administration, J.M. All authors have read and agreed to the published version of the manuscript.

Funding: TESTUDO project has received funding from the European Union's Horizon Europe research and innovation programme under grant agreement No 101121258. Views and opinions expressed are, however, those of the author(s) only and do not necessarily reflect those of the European

Un-ion or the European Commission. Neither the European Union nor the granting authority can be held responsible for them.

Data Availability Statement: Data are contained within the article.

Acknowledgments: ChatGPT-4o was used to help with academic writing and paraphrasing.

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

References

1. TESTUDO Consortium. TESTUDO: Autonomous Swarm of Heterogeneous Resources in Infrastructure Protection via Threat Prediction and Prevention. 2023. Available online: https://testudo-project.eu/ (accessed on 6 March 2025).
2. Mikhalevich, I.F.; Trapeznikov, V.A. Critical Infrastructure Security: Alignment of Views. In Proceedings of the 2019 Systems of Signals Generating and Processing in the Field of on Board Communications , Moscow, Russia, 20–21 March 2019; pp. 1–5.
3. Talaat, F.M.; ZainEldin, H. An improved fire detection approach based on YOLO-v8 for smart cities. *Neural Comput. Appl.* **2023**, *35*, 20939–20954. [CrossRef]
4. Castro Correa, J.A.; Sepúlveda Mora, S.B.; Medina Delgado, B.; Escobar Amado, C.D.; Guevara Ibarra, D. A forest fire monitoring and detection system based on wireless sensor networks. *Sci. Tech.* **2022**, *27*, 89–96. [CrossRef]
5. Lema, D.G.; Usamentiaga, R.; García, D.F. Quantitative comparison and performance evaluation of deep learning-based object detection models on edge computing devices. *Integration* **2024**, *95*, 102127. [CrossRef]
6. Laganà, F.; Bibbò, L.; Calcagno, S.; De Carlo, D.; Pullano, S.A.; Pratticò, D.; Angiulli, G. Smart Electronic Device-Based Monitoring of SAR and Temperature Variations in Indoor Human Tissue Interaction. *Appl. Sci.* **2025**, *15*, 2439. [CrossRef]
7. Arnautovic, A.; Teskeredzic, E. Evaluation of Artificial Neural Network Inference Speed and Energy Consumption on Embedded Systems. In Proceedings of the 2021 20th International Symposium INFOTEH-JAHORINA, INFOTEH 2021, East Sarajevo, Bosnia and Herzegovina, 17–19 March 2021; pp. 17–19.
8. Baller, S.P.; Jindal, A.; Chadha, M.; Gerndt, M. DeepEdgeBench: Benchmarking Deep Neural Networks on Edge Devices. In Proceedings of the 2021 IEEE International Conference on Cloud Engineering (IC2E), San Francisco, CA, USA, 4–8 October 2021; pp. 20–30.
9. Ismagilov, R. Performance Evaluation of the Rockchip Systems- on-Chip Through YOLOv4 Object Detection Model. In Proceedings of the 2023 IEEE Ural-Siberian Conference on Biomedical Engineering, Radioelectronics and Information Technology (USBEREIT), Yekaterinburg, Russia, 15–17 May 2023; pp. 241–243.
10. Elhence, A.; Panda, A.; Chamola, V.; Sikdar, B. FPGA-Accelerated YOLOX with Enhanced Attention Mechanisms for Real-Time Wildfire Detection on UAVs. *IEEE Trans. Instrum. Meas.* **2025**. [CrossRef]
11. Miglionico, G.C.; Ducange, P.; Marcelloni, F.; Vallati, C.; Di Rienzo, F. Performance Evaluation of YOLOv5 on Edge Devices for Personal Protective Equipment Detection. In Proceedings of the 2024 International Joint Conference on Neural Networks (IJCNN), Yokohama, Japan, 30 June–5 July 2024; pp. 1–8.
12. Rahman, S.; Jamee, S.M.H.; Rafi, J.K.; Juthi, J.S.; Sajib, A.A.; Uddin, J. Real-time smoke and fire detection using you only look once v8-based advanced computer vision and deep learning. *Int. J. Adv. Appl. Sci.* **2024**, *13*, 987. [CrossRef]
13. Norkobil Saydirasulovich, S.; Abdusalomov, A.; Jamil, M.K.; Nasimov, R.; Kozhamzharova, D.; Cho, Y.I. A YOLOv6-Based Improved Fire Detection Approach for Smart City Environments. *Sensors* **2023**, *23*, 3161. [CrossRef]
14. Zheng, H.; Duan, J.; Dong, Y.; Liu, Y. Real-time fire detection algorithms running on small embedded devices based on MobileNetV3 and YOLOv4. *Fire Ecol.* **2023**, *19*, 31. [CrossRef]
15. Zheng, H.; Wang, G.; Xiao, D.; Liu, H.; Hu, X. FTA-DETR: An efficient and precise fire detection framework based on an end-to-end architecture applicable to embedded platforms. *Expert Syst. Appl.* **2024**, *248*, 123394. [CrossRef]
16. Fernandez, P.; Jimenez, J.; Astarloa, A.; Idirin, M.; Salas, S. Accelerating Video Analytic Processing on Edge Intelligence. In Proceedings of the 2022 37th Conference on Design of Circuits and Integrated Circuits (DCIS), Pamplona, Spain, 16–18 November 2022; pp. 1–6.
17. Tatar, G.; Bayar, S. Real-Time Multi-Task ADAS Implementation on Reconfigurable Heterogeneous MPSoC Architecture. *IEEE Access* **2023**, *11*, 80741–80760. [CrossRef]
18. Feng, H.; Mu, G.; Zhong, S.; Zhang, P.; Yuan, T. Benchmark Analysis of YOLO Performance on Edge Intelligence Devices. *Cryptography* **2022**, *6*, 16. [CrossRef]
19. Das, A.; Patterson, S.; Wittie, M. EdgeBench: Benchmarking Edge Computing Platforms. In Proceedings of the 2018 IEEE/ACM International Conference on Utility and Cloud Computing Companion (UCC Companion), Zurich, Switzerland, 17–20 December 2018; pp. 175–180. .
20. Zhang, X.; Wang, Y.; Shi, W. pCAMP: Performance Comparison of Machine Learning Packages on the Edges. *arXiv* **2019**. [CrossRef]

21. Luo, C.; Zhang, F.; Huang, C.; Xiong, X.; Chen, J.; Wang, L.; Gao, W.; Ye, H.; Wu, T.; Zhou, R.; et al. AIoT Bench: Towards Comprehensive Benchmarking Mobile and Embedded Device Intelligence. In *Benchmarking, Measuring, and Optimizing. Bench 2018. Lecture Notes in Computer Science*; Springer: Cham, Switzerland, 2019; pp. 31–35.
22. Varghese, B.; Wang, N.; Bermbach, D.; Hong, C.H.; Lara, E.D.; Shi, W.; Stewart, C. A Survey on Edge Performance Benchmarking. *ACM Comput. Surv.* **2022**, *54*, 1–33. [CrossRef]
23. Aslanpour, M.S.; Gill, S.S.; Toosi, A.N. Performance evaluation metrics for cloud, fog and edge computing: A review, taxonomy, benchmarks and standards for future research. *Internet Things* **2020**, *12*, 100273. [CrossRef]
24. Martin, J.; Cantero, D.; González, M.; Cabrera, A.; Larrañaga, M.; Maltezos, E.; Lioupis, P.; Kosyvas, D.; Karagiannidis, L.; Ouzounoglou, E.; et al. Embedded Vision Intelligence for the Safety of Smart Cities. *J. Imaging* **2022**, *8*, 326. [CrossRef] [PubMed]
25. Chen, L.; Hu, J.; Li, X.; Quan, F.; Chen, H. Onboard Real-time Object Detection for UAV with Embedded NPU. In Proceedings of the 2021 IEEE 11th Annual International Conference on CYBER Technology in Automation, Control, and Intelligent Systems (CYBER), Jiaxing, China, 27–31 July 2021; pp. 192–197.
26. Liu, M.; Tang, L.; Li, Z. Real-Time Object Detection in UAV Vision based on Neural Processing Units. In Proceedings of the 2022 IEEE 6th Information Technology and Mechatronics Engineering Conference (ITOEC), Chongqing, China, 4–6 March 2022; pp. 1951–1955.
27. Jinguji, A.; Sada, Y.; Nakahara, H. Real-Time Multi-Pedestrian Detection in Surveillance Camera using FPGA. In Proceedings of the 2019 29th International Conference on Field Programmable Logic and Applications (FPL), Barcelona, Spain, 9–13 September 2019; pp. 424–425.
28. Uribe-Rios, J.; Castano-Londono, L.; Marquez-Viloria, D.; Morantes-Guzman, L. Vehicle Detection and Counting Framework in Aerial Images Based on SoC-FPGA. In *Applied Computer Sciences in Engineering, Proceedings of the 9th Workshop on Engineering Applications, WEA 2022, Bogotá, Colombia, 30 November–2 December 2022*; Figueroa-García, J.C., Franco, C., Díaz-Gutierrez, Y., Hernández-Pérez, G., Eds.; Communications in Computer and Information Science; Springer Nature: Cham, Switzerland, 2022; Volume 1685, pp. 473–484.
29. Nghi, H.P.; Thinh, T.N. A CNN-Based Vehicle Identification Solution in Parking System With Hardware Accelerator on FPGA. In Proceedings of the 2022 RIVF International Conference on Computing and Communication Technologies (RIVF), Ho Chi Minh City, Vietnam, 20–22 December 2022; pp. 518–523.
30. Han, K.; Luo, Y. Feasibility Analysis and Implementation of Adaptive Dynamic Reconfiguration of CNN Accelerators. *Electronics* **2022**, *11*, 3805. [CrossRef]
31. Farooq, M.A.; Shariff, W.; Corcoran, P. Evaluation of Thermal Imaging on Embedded GPU Platforms for Application in Vehicular Assistance Systems. *IEEE Trans. Intell. Veh.* **2023**, *8*, 1130–1144. [CrossRef]
32. Matic, I.; de Lemos, F.; Ihianle, D.I.K.; Adama, D.D.A.; Machado, P. Estimating the Power Consumption of Heterogeneous Devices When Performing Ai Inference. *SSRN Electron. J.* **2022** . [CrossRef]
33. 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. [CrossRef]
34. Cantero, D.; Esnaola-Gonzalez, I.; Miguel-Alonso, J.; Jauregi, E. Benchmarking Object Detection Deep Learning Models in Embedded Devices. *Sensors* **2022**, *22*, 4205. [CrossRef]
35. Jocher, G. Ultralytics /Yolov5: v6.0—YOLOv5n 'Nano' Models, Roboflow Integration, TensorFlow Export, OpenCV DNN Support. 2021. Available online: https://github.com/ultralytics/yolov5 (accessed on 6 March 2025).
36. de Venâncio, P.V.A.B.; Lisboa, A.C.; Barbosa, A.V. An automatic fire detection system based on deep convolutional neural networks for low-power, resource-constrained devices. *Neural Comput. Appl.* **2022**, *34*, 15349–15368. [CrossRef]
37. ARM. Ethos-U 65. Available online: https://developer.arm.com/Processors/Ethos-U65 (accessed on 11 March 2025).
38. NVIDIA. Jetson Orin Nano. Available online: https://docs.rs-online.com/4051/A700000009607470.pdf (accessed on 11 March 2025).
39. dos Santos, J.P. Problems with Quantized Model #12609. Available online: https://github.com/ultralytics/yolov5/issues/12609 (accessed on 21 February 2025).