cryptography

MDPI

Article

Benchmark Analysis of YOLO Performance on Edge Intelligence Devices

Haogang Feng [1,2], Gaoze Mu [1,2], Shida Zhong [1,2,*], Peichang Zhang [1] and Tao Yuan [2,3]

[1] College of Electronics and Information Engineering, Shenzhen University, Nanhai Avenue 3688, Shenzhen 518060, China; fenghaogang@email.szu.edu.cn (H.F.); 2060432051@email.szu.edu.cn (G.M.); pzhang@szu.edu (P.Z.)
[2] Guangdong Provincial Mobile Terminal Microwave and Millimeter Wave Antenna Engineering Research Center, College of Electronics and Information Engineering, Shenzhen University, Shenzhen 518060, China; yuantao@szu.edu.cn
[3] Guangdong-Hong Kong Joint Laboratory for Big Data Imaging and Communication, Shenzhen 518048, China
* Correspondence: shida.zhong@szu.edu.cn

Abstract: In the 5G intelligent edge scenario, more and more accelerator-based single-board computers (SBCs) with low power consumption and high performance are being used as edge devices to run the inferencing part of the artificial intelligence (AI) model to deploy intelligent applications. In this paper, we investigate the inference workflow and performance of the You Only Look Once (YOLO) network, which is the most popular object detection model, in three different accelerator-based SBCs, which are NVIDIA Jetson Nano, NVIDIA Jetson Xavier NX and Raspberry Pi 4B (RPi) with Intel Neural Compute Stick2 (NCS2). Different video contents with different input resize windows are detected and benchmarked by using four different versions of the YOLO model across the above three SBCs. By comparing the inference performance of the three SBCs, the performance of RPi + NCS2 is more friendly to lightweight models. For example, the FPS of detected videos from RPi + NCS2 running YOLOv3-tiny is 7.6 times higher than that of YOLOv3. However, in terms of detection accuracy, we found that in the process of realizing edge intelligence, how to better adapt a AI model to run on RPi + NCS2 is much more complex than the process of Jetson devices. The analysis results indicate that Jetson Nano is a trade-off SBCs in terms of performance and cost; it achieves up to 15 FPSs of detected videos when running YOLOv4-tiny, and this result can be further increased by using TensorRT.

Keywords: intelligence edge; edge computing; accelerator-based SBCs; YOLO network; Internet of Things (IoT)

Citation: Feng, H.; Mu, G.; Zhong, S.; Zhang, P.; Yuan, T. Benchmark Analysis of YOLO Performance on Edge Intelligence Devices. *Cryptography* **2022**, *6*, 16. https://doi.org/10.3390/cryptography6020016

Academic Editors: Cong Ling, Shanxiang Lyu, Ling Liu and Jiabo Wang

Received: 1 March 2022
Accepted: 28 March 2022
Published: 1 April 2022

1. Introduction

Recently, with the rapid development of 5th-Generation mobile networks (5G), communication technologies and Artificial Intelligence (AI), the Internet of Things (IoT) and edge computing have brought a new application revolution to our daily life. More specifically, 5G provides three broad use cases for meeting different service requirements: enhanced mobile broadband (eMBB), ultra-reliable low-latency communications (uRLLC) and massive machine-type communication (mMTC) [1]. At the same time, due to the great progress of IoT and radio access network (RAN), IoT devices have more computing resources and assume the role of the main executor of 5G core applications. In key emerging mMTC application scenarios such as smart cities, wide-area disaster monitoring and wireless factory automation, the reliability and latency of seamless operations are key issues [2]. In addition, corresponding to the three 5G application scenarios above, it is predicted that first, by the end of 2026, 44% of cellular IoT connections will be broadband IoT; secondly, the first batch of use-case modules for the critical IoT will be deployed in 2021; and finally the number of massive IoT connections in 2020 has reached 200 million, which is twice the number of

connections in 2019 [3]. Therefore, there will be more and more IoT devices with sufficient performance that can directly support eMBB and uRLLC applications, which means that massive amounts of data can be generated locally and consumed locally. As computing and storage resources are critical to enable the three 5G application scenarios, edge computing, which handles the computation-intensive and latency-critical tasks at the edge servers and devices directly, plays a significant role in 5G [4–6]. Compared with the limitations of cloud computing in real-time smart environments, edge computing brings computing resources closer to users, which can reduce communication latency and overhead and better protect the privacy of individuals' personal data. Figure 1 presents the architecture of an IoT-assisted edge computing system.

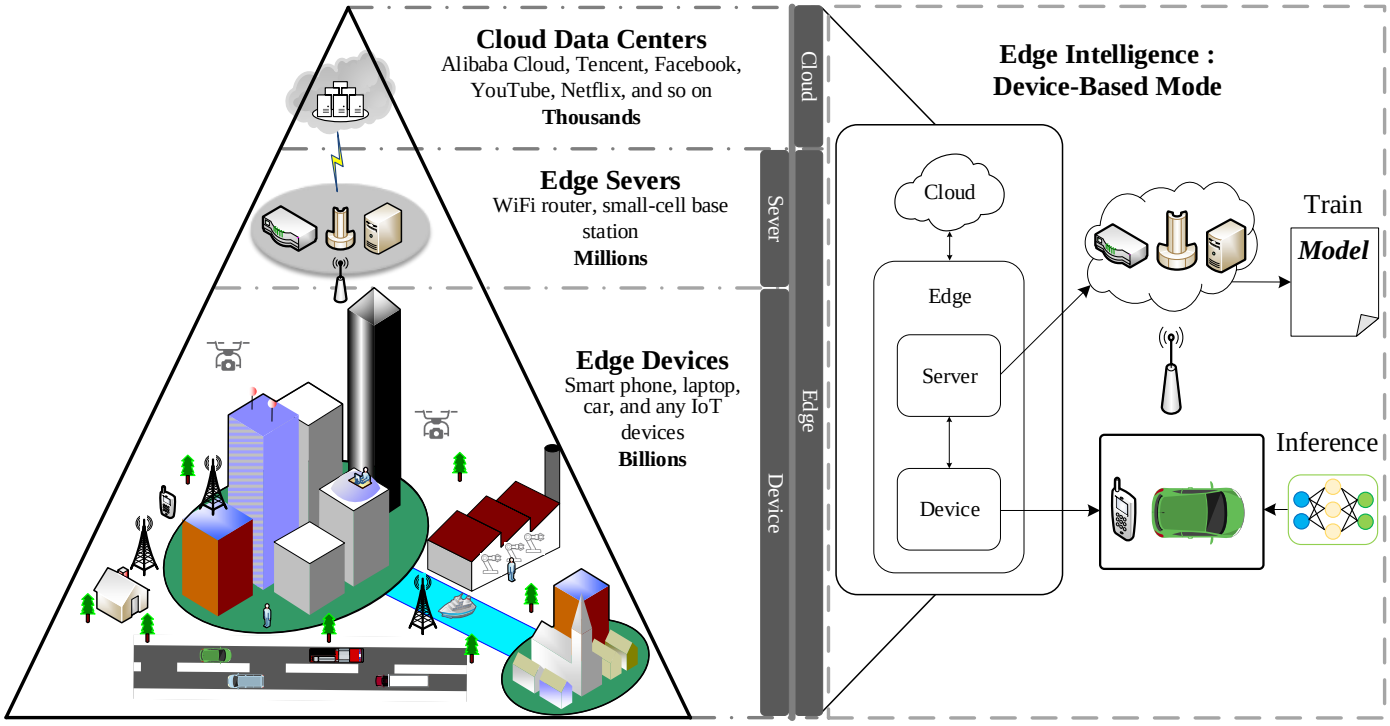


Figure 1. The structure of edge computing.

Just like the novel contributions of 5G, IoT and edge computing, the development of Deep Learning (DL) has impacted every aspect of people's lives, such as object recognition and natural language processing. Nowadays, the emergence of various applications has increasingly pushed the inference part of DL models to edge devices such as IoT devices, smart phones and wearable devices [7]. When the inference environment of DL algorithms transferred from the core cloud to the edge cloud, the massive data generated at the edge can be processed in real time and efficiently, which means that the high latency, privacy leakage and other issues will be fundamentally resolved. The integration of AI and edge computing, named edge intelligence (EI), will bring unlimited possibilities to our lives [8]. More importantly, the realization of EI—which brings DL closer to people and data—requires intelligence edge devices to execute the inference part of the DL models [9]. However, these intelligent edges running DL algorithms must have the characteristics of energy efficiency, miniaturization and low cost [10], which are essential for the IoT, driverless and wearable devices, robotics, etc. Two aspects must be met to realize the intelligent edge. Firstly, there must be an edge device or auxiliary device with sufficient performance to run the DL model, such as a single-board computer with a mobile GPU or a coprocessor such as a Google Coral Accelerator and Intel Neural Compute Stick [11,12]. Secondly, there must be DL models adapted to these devices to meet the requirements of the corresponding application scenarios on the inference performance including inference

speed, accuracy, etc. [13]. More and more companies are participating in the design of neural network chips for inference to support the edge-computing paradigm and promote DL acceleration on resource-limited IoT devices. The hardware upgrade of edge devices has also injected vitality into AI. However, despite the rapid development of the current EI field, there are huge deficiencies—many AI models for low-performance edge devices are only developed at an overly broad theoretical level. Therefore, it is of great practical significance to explore the ability and bottleneck of specific different types of edge intelligence devices to run high-performance AI models, so as to provide specific design reference both for algorithm and hardware development. In this article, we mainly focus on the inference of AI models on edge devices because the limited computing resources and storage resources of edge devices mean AI models are more trained in edge server or core cloud.

Against this background, in this paper: (1) four You Only Look Once (YOLO) models are investigated and benchmarked in three different edge intelligence devices, which are NVIDIA Jetson Nano with TensorFlow GPU, NVIDIA Jetson Xavier NX with TensorFlow GPU and Raspberry Pi (RPi) 4B with Intel Neural Compute Stick2 (NCS2). (2) The methodology and steps of the benchmark performance on different edge intelligence devices are introduced and analyzed. (3) Empirical recommendations and potential problems based on analytical results in term of frames-per-second, memory usage, CPU usage, and energy consumption of the three single-board computers (SBCs) are given for deploying AI applications on these types of intelligence edges.

The rest of this paper is given as follows. Section 2 will provide a survey with related research projects. YOLO networks, three kinds of intelligence edge and their working principles will be introduced in Section 3. In Section 4, the benchmark performance of the intelligence edges for the YOLO models will be analyzed, and the conclusions are given in Section 5.

2. Related Work

Due to the character of low power consumption, tiny volume and high performance, single-board computers are widely used in IoT scenarios and have become the main carrier of edge accelerators such as AI chips. As the carrier of the edge AI inference model, when the SBCs are combined with various edge accelerators, many AI applications can be implemented on them, uncovering exciting opportunities for building powerful applications with complicated learning objectives and demanding computations. Generally speaking, there are three ways to combine single board computers and edge accelerators: (1) GPU-based SBCs, which are suitable for general-purpose applications because of their good compatibility, such as Jetson Nano and JeVois A33 [14]; (2) Application Specific Integrated Circuit (ASIC)-based SBCs, which specialize in deep neural network applications, such as Intel's vision processing unit (VPU) [15], Google's tensor processing unit (TPU) [16] and Sipeed Maix Bit with K210 [14,17]; and (3) Field Programmable Gate Array (FPGA)-based SBCs, which are more energy efficienct [18]. Our work will be carried out on two kinds of GPU-based SBCs (NVIDIA Jetson Nano and NVIDIA Xavier NX) and a ASIC-based SBC (Raspberry Pi 4B with Intel Neural Compute Stick2).

NVIDIA Jetson Nano is the entry-level device from NVIDIA for embedded IoT applications. For our benchmark, we choose another NVIDIA's embedded computing device named NVIDIA Xavier NX, which is more advanced. Compare with JeVois A33 and other GPU-based SBCs, the Jetson Series offers solutions that meet specific performance and budget needs for large state-owned enterprises, small and medium-sized enterprises and individual researchers. All of these solutions use the same architecture and SDK to achieve a code-based and seamless deployment across the product portfolio. However, JeVois A33 has advantages only when it completes machine-vision-related functions. Therefore, we chose two more universal and high-performance Jetson series devices for the benchmark analysis of this paper. Additionally, the Intel Neural Compute Stick2 (NCS2) is a plug-in development kit for AI inferencing and is perfectly supported by Raspberry Pi hardware. When we talk about Intel NCS2, its role in sharing computing power is irreplaceable for

Sipeed Maix Bit with K210 and other ASIC-based SBC. For an Unmanned Aerial Vehicle (UAV) that is already controlled by RPi, we can offload the face or gesture recognition function onto NCS2, which makes it very easy to further develop the UAV's function; for an unmanned vessel, unmanned vehicle or robot, we only need to add one or multiple NCS2s to them to complete the recognition or computing complement. Moreover, the combination of RPi + NCS2 can run more kinds and larger deep learning algorithm models than Sipeed Maix Bit, so we chose the Jetson series and RPi + NCS2 as our benchmark test devices. The details of the above three devices will be introduced in Section 3.2.

In recent years, more and more attention has been paid to the analysis of the edge accelerator-based SBCs [19–23] and the implementation of intelligence applications by using these intelligence edges [24–26]. Most notably, Basulto-Lantsova et al. [19], Jo et al. [20] and Süzen et al. [21] all analyzed the performance of the Jetson Nano and Jetson TX2, which are both GPU-based SBCs, when implementing intelligence models. Although they have shown the relevant performance evaluations of the two devices in detail, the TX2 has been sold for more than 3 years and has an excessively large size. Thus, we used the brand new Jetson Xavier NX, which is an upgraded version of Jetson TX2 and maintains the same price, but has stronger computing power. In addition, Süzen et al. [21] trained and tested an image-classification 2D-CNN model on Raspberry Pi 4 for benchmark performance comparison with Jetson Nano, but the Raspberry Pi (RPi) is not equipped with an edge accelerator to co-process data with CPU. Moreover, Aleksandrov [22] verified the necessity of Intel NCS2 for low-performance edge devices by evaluating the face recognition performance on a video, which is very enlightening for our work.

In terms of intelligence applications, Srinivasan et al. [25] built a real-time emotions-classification system by implementing a DNN model on a RPi with NCS and evaluated several DNN models on the RPi. The FPS of multiple DNN models performed on the RPi was around two, which is too slow to satisfy the demand of real-time emotion detection even if using NCS on RPi to get around 6 FPS. Additionally, Sahu et al. [24] implemented the inference part of a skin lesion image-classification DNN model, which is offline-trained on the NVIDIA GPU, on the RPi with a NCS to build a hand-held assistant. The speed to process 224 × 224 skin images of using RPi with NCS, i.e., 0.65 s/image, was five times faster than using RPi, i.e., 3.3 s/image. Jung et al. [26] implemented a person-recognition system, in which a four-channel real-time camera's data were processed in one image frame to detect four directions' areas and succeeded further with recognizing a person by adopting a YOLOv3 network to Jetson AGX Xavier on autonomous tractors. They used a YOLOv3 network, which was trained on their own dataset that collected at the LSMtron driving test field, to perform 'only person detection'. The detection results showed up to 15.7 FPS and 86.19% precision, which showed 2.3 FPS faster and 0.71% higher precision than the YOLOv3 network trained on the COCO dataset that can perform 80-class detection.

Object recognition is the key to many 5G EI applications, such as intelligent agriculture [27,28] and intelligent transportation [29–31]. Among many object recognition models, the YOLO network is one of the most popular CNN-based algorithms that can implement high-precision detection in real-time scenes [32,33]. However, it is difficult for both device and edge-based methods to fully support many real-time intelligent applications with strict latency requirements because of edge devices' limited computing resources. Therefore, how to deploy AI models and perform the inference part on edge devices is a very critical issue. The enabling technologies that improve the performance of EI model inference include, but are not limited to, model compression, model partition, input filtering, model early exit, results caching and so on. Deng et al. [34] defined the research direction of 'how to run AI models on edge' as 'AI on edge', and divided the related research works into three categories: (1) model adaptation; (2) framework design; and (3) processor acceleration. Zhou et al. [8] defined four architecture modes for the inference part of DNN models on edge devices, namely edge-based, device-based, edge device and edge cloud. As shown in the right part of Figure 1, a device-based model refers to transmitting the DNN model that trained at the edge server to the edge device and directly preforming the model inference

on the local device, which is the same as this benchmark analysis. In the device-based mode, model adaptation and frame design require additional attention. Li et al. [35] explored a typical method where the trained DNN model was split into two parts, just like the edge-device mode. The first part of the network layers that were computation intensive and would produce less data were placed on the edge server, and the other part was placed on the edge device. The edge device user sent the data that needed to be processed to the edge server, and the edge server sent the processing results of the previous part of the model to the edge device to complete the rest part, and the final result was presented to the user directly. This process required a trade-off between computational overhead and communication overhead.

Therefore, exploring the operating performance of the YOLO networks on the current popular universal edge accelerator-based SBCs is of great significance for research work and industrial applications in the field of object recognition edge computing.

3. Backgrounds and Methodology

In this section, the YOLO networks and the SBCs that used in the benchmarks are discussed in detail.

3.1. YOLO Networks

You Only Look Once (YOLO) v3, invented and proposed in 2018, has become a very classic one-stage algorithm in object detection, including the darknet-53 network structure, anchor frame, feature pyramid networks and other excellent structures. As the backbone of YOLOv3 for feature extraction, darknet-53 introduces the residual module on the basis of darknet-19 (contains 19 convolutional layers), and further deepens the network. It mainly contains 53 successive 1×1 and 3×3 convolutional layers, so it is named darknet-53 [32]. YOLOv3-tiny is a simplified version of YOLOv3. The main difference is that the backbone network of YOLOv3-tiny uses a seven-layer combination of convolutional and pooling layers which is similar to darknet-19, and the graft network uses a detection network with 13×13 and 26×26 resolutions. As shown in Table 1, the scale of YOLOv3-tiny's feature extraction network is much smaller than that of YOLOv3. Therefore, YOLOv3-tiny is unable to extract higher-level semantic features, which means the detection accuracy has decreased. However, the smaller network model of YOLOv3-tiny also leads to a greater increase in detection speed compared to YOLOv3 [36,37]. YOLOv4 and YOLOv4-tiny were proposed by Bochkovskiy in 2020 [33]. YOLOv4 has optimized and improved every part of YOLOv3. The main optimization is to use CSPDarknet-53 as its backbone network for extracting feature. Moreover, the framework of YOLOv4-tiny's feature extraction network is similar to that of YOLOv3, except that three tricks are added to it to make the network easier to tain. The difference between YOLOv4-tiny and YOLOv4 is that the tiny version only has two YOLO heads at the end. All of the YOLO networks divide every picture into $S \times S$ grid cells as the input. The input size S of the networks is not fixed, and can be modified as needed in actual projects, such as 608×608, 416×416, and 320×320, which is generally a multiple of 32. The larger the S value, the better the detection effect of small targets, but the amount of video memory occupied will be higher and the computation time for inference will be correspondingly longer, which needs to be weighed.

Table 1. Comparison of the scale of the backbone between different YOLO models.

YOLO	Total Layers of Backbone
YOLOv3	106
YOLOv3-tiny	24
YOLOv4	161
YOLOv4-tiny	38

3.2. Hardware Platforms

Two different GPU-based SBCs and one ASIC-based SBC are considered in this study; their hardware specifications are given in Table 2. Table 2 shows that NVIDIA Jetson Xavier NX has the superior AI performance, but comes with higher price. RPi connects NCS2 with USB 3.0, and together they have the lowest power consumption.

Table 2. Hardware specifications used in the benchmarks.

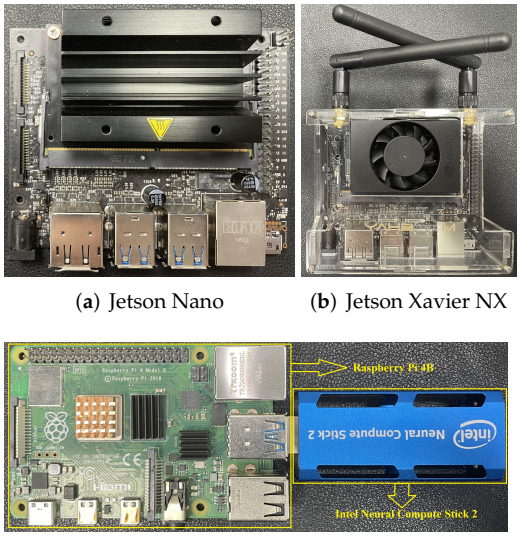
	NVIDIA Jetson Nano	**NVIDIA Jetson Xavier NX**	**Raspberry 4B + Intel NCS2**
Edge Accelerator	128-core NVIDIA Maxwell GPU	384-core NVIDIA Volta GPU with 48 Tensor Cores	Intel Movidius Myriad X VPU
AI Performance	0.5 TFLOPs	1.3 TFLOPs	1 TFLOPs
CPU	Quad-core ARM Cortex-A57 MPCore processor	6-core NVIDIA Carmel ARM v8.2 64-bit CPU 6 MB L2 + 4 MB L3	Quad-core ARM Cortex-A72
Memory	4 GB 64-bit LPDDR4 25.6 GB/s	8 GB 128-bit LPDDR4x 51.2 GB/s	4 GB LPDDR4
Edge Accelerator Interface	PCIe	PCIe	USB 3.0
Dimensions	69.6 mm × 45 mm	69.6 mm × 45 mm	85 mm × 56 mm +72.5 mm × 27 mm
Nominal Power Envelope	5 W–10 W	10 W–15 W	3 W–6.25 W +1 W
Price	USD 89	USD 399	USD 55 + USD 69

3.2.1. Jetson Nano and Jetson Xavier NX

As shown in Figure 2a,b, CPU-GPU heterogeneous architecture is the most basic feature of Jetson Nano and Jetson Xavier NX [38,39]. As the latest device in the NVIDIA Jetson ecosystem, Jetson Xavier NX integrates the NVIDIA Xavier System on a Chip (SoC) in the same size module as Jetson Nano, just like the official introduction of NVIDIA: "The World's Smallest AI Supercomputer for Embedded and Edge Systems". It should be noted that in order to show the best performance of Jetson Xavier NX, we chose the six-core 15 W model. In addition, Jetson Nano and NX are both running operating systems derived from Ubuntu. In order to benchmark with RPi, we did not consider TensorRT but considered using GPU to compile TensorFlow.

3.2.2. Intel Neural Compute Stick2

Intel NCS2 is a USB 3.1 stick that hosts the Intel Movidius Myriad X VPU [15]. As a powerful and low-power coprocessor, Intel NCS2 allows the SBC to offload part or all of the DNN inference. NCS2 is supported by Ubuntu 16.04.3, Windows 10, Rsapbian and other operating systems via the open-source distribution of OpenVINO [40]. As shown in Figure 2c, we used RPi 4B as the SBC for benchmark. Similarly, we first converted the YOLO model to a TensorFlow model, and then used the OpenVINO ToolKit to convert the TensorFlow model to an Intermediate Representation (IR) model which is a pair of .xml and .bin files that can run on the NCS2, which are presented in detail in Section 3.3.



(**a**) Jetson Nano (**b**) Jetson Xavier NX

(**c**) Raspberry Pi 4B with NCS2

Figure 2. Hardware platforms used in this study. (**a**) Jetson Nano. (**b**) Jetson Xavier NX. (**c**) Raspberry Pi 4B with NCS2.

3.3. Compilation Workflow

In this paper, we used the pre-trained weights files of YOLO models from the Darknet website [41]. Figure 3 presents the compile process of YOLO models on the SBCs, which require different constraints, to fully exploit the SBCs for model inference. In this section, we will introduce the related process for compilation.

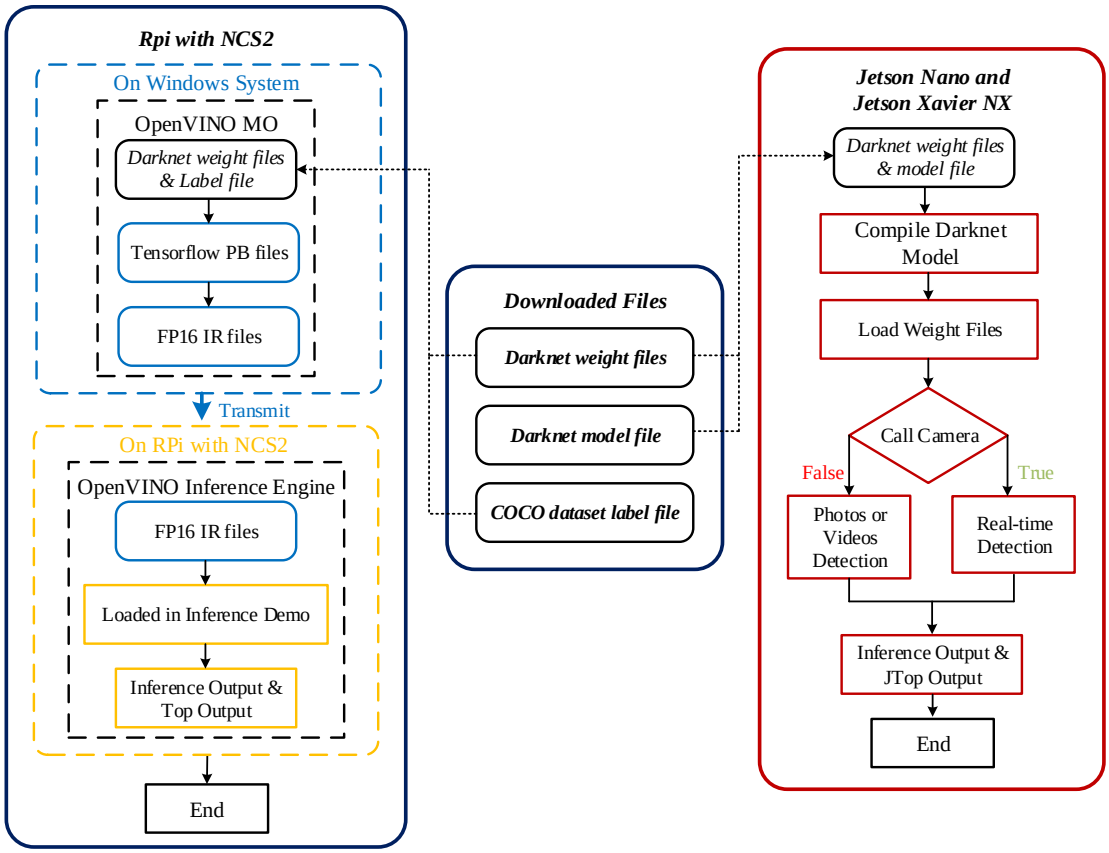


Figure 3. Compilation workflow on different hardware platforms.

3.3.1. Jetson Nano and Jetson Xavier NX

In the first step, we downloaded Darknet code on the two GPU-based SBCs. Then the Darknet code was compiled on them. The pre-trained weights file, which is trained on COCO dataset (a large-scale object detection, segment, and captioning dataset [42]) in advance, was loaded into the compiled model. You can also train your own dataset based on this weight, but for us, this pre-trained weight was enough to complete our benchmark work. The YOLO model can call the camera for real-time detection through preset instructions and can also detect video files in the device. Since real-time detection is not easy for performance comparison, we let different devices detect a same video. We tested three normalized input box sizes (320×320, 416×416, 608×608) of the YOLO model. It should be noted that Jetson Xavier NX's GPU supports Int8 and half-precision (FP16) computation and Intel NCS2 supports single-precision (FP32) and FP16 computation; however, Jetson Nano only supports FP16 computation. Therefore, in configuring the benchmark setting, FP16 models were used for the benchmark works.

3.3.2. RPi and Intel Neural Compute Stick2

Object detection with YOLO on Raspberry Pi does not require downloading the Darknet model. Running YOLO on RPi only needs weights files and the the label file. We converted the weights files into tensorflow protocol buffer (PB) files first, and then converted them to FP16 IR files. All the above work was performed on a Windows system by using OpenVINO toolkit's model optimizer (MO). Then we installed the OpenVINO toolkit, which facilitated the deployment by using the inference engine onto Intel hardware [43], in Raspberry Pi and compiled the previously converted files on the device. Acceleration with the NCS2 requires only a simple configuration of the NCS USB driver and then it is plug-and-play. The inference engine Application Programming Interface (API) ran the model over the Movidius X VPU with 16 programmable shave cores and a neural compute engine.

4. Benchmark Results and Analysis

In this section, we will analyse the benchmark results to measure the inference performance and capabilities of accelerator-based SBCs to uncover the feasibility of implementing AI model inference and applications on these accelerator-based SBCs.

A video sequence of 1596 frames with a frame size of 768×436 (Video1) and a video sequence of 960 frames with a frame size of 1920×1080 (Video2) were downloaded from gitee [44] and used as the test dataset for the object detection to evaluate the comparative inference performance when inputting different size of data. More specifically, a half-length portrait was shown in Video1, and the person in the video rotated his head at multiple angles. A classroom with multiple tables and chairs and four people were shown in Video2. These four people in Video2 performed different actions such as standing up, sitting down and turning. We considered the frames-per-second (FPS), memory usage, CPU usage, and energy consumption as the basic performance metrics throughout the analysis process. To benchmark the memory usage, CPU usage and energy consumption, JTOP software was used in Jetson devices and TOP command and a USB dynamometer were used on RPi.

The analysis process was carried out from two aspects. Firstly, each accelerator-based SBC performed inferences by running YOLOv3 and YOLOv3-tiny, with the value of S (The input resize of YOLO network) at 416, to investigate their resource characteristics. Secondly, to further compare the inference performance, we ran YOLOv4 and YOLOv4-tiny with $S = 416$ on Jetson Nano, and set the $S = 320$ and $S = 608$ of YOLOv3 and YOLOv3-tiny to run on the Jetson Nano and Jetson Xavier NX for further inference to analyze the affect of YOLO model optimisation parameters. It is important to note that the data compared in Sections 4.1 and 4.2 are the average result of 10 inference processes. For example, to obtain the average Mean Confidence of Jetson Nano when it ran YOLOv3 to detect Video1, we first calculated the average detection Mean Confidence of Video1's 960 frames, then repeated the process 10 times to obtain a more accurate average.

4.1. Benchmarking Jetson Nano, Jetson Xavier NX and RPi with Intel Neural Compute Stick2

The benchmark results of inference performance on accelerator-based SBCs by running YOLOv3 and YOLOv3-tiny with $S = 416$ are shown in Table 3. The next following subsections will discuss the results in Table 3 in details from three aspects: FPS, memory usage and energy consumption.

Table 3. Benchmarking Jetson Nano, Jetson Xavier NX and RPi with NCS2. (Note: Video1 has 1596 frames with a frame size of 768 × 436, Video2 has 960 frames with a frame size of 1920 × 1080).

	Models	Accelerator -Based SBCs	Mean Confidence (%)	FPS	CPU Usage (%)	Memory Usage (GB)	Power (W)	Time (s)
Video1	**YOLOv3**	RPi + NCS2	99.3	2.5	4.3	0.33	6.0	690
		Nano	99.7	1.7	26.5	1.21	7.9	967
		NX	99.7	6.1	22.5	1.51	15.2	256
	YOLOv3-tiny	RPi + NCS2	0	18.8	15.5	0.11	6.5	121
		Nano	59.7	6.8	28.8	1.00	7.2	236
		NX	59.7	41.1	30.5	1.33	13.5	46
Video2	**YOLOv3**	RPi + NCS2	85.8	2.5	9.8	0.41	6.2	496
		Nano	71.5	1.6	28.8	1.36	8.0	587
		NX	71.5	5.9	26.8	1.69	15.2	162
	YOLOv3-tiny	RPi + NCS2	61.5	19.0	24.8	0.18	6.8	162
		Nano	54.1	6.6	37.3	1.16	7.4	152
		NX	54.1	35.6	55.5	1.47	13.2	31

4.1.1. Inference Performance

The FPS of the model inference is a key metric for evaluating the video detection performance, which reflects the processing speed of an accelerator-based SBC to input data. Figure 4 summarizes the inference results of the FPS performance shown in Table 3, that measured across the YOLOv3 and YOLOv3-tiny running on accelerator-based SBCs.

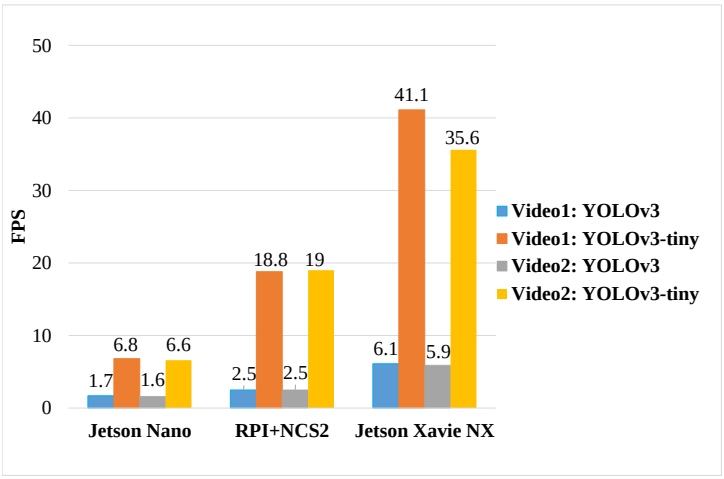


Figure 4. FPS inference performance on different accelerator-based SBCs.

It can be easily concluded from Figure 4 that the YOLO model is the main factor affecting inference performance for each accelerator-based SBC, i.e., the more complex the

model is, the slower the inference is. At the same time, it can be seen that the influence of the size of input data on inference performance is quite slight, even though the frame sizes of Video1 and Video2 are quite different. For example, the FPS of YOLOv3 on RPi + NCS2 for processing Video1 and Video2 were both 2.5. Besides that, Jetson Xavier NX outperformed other two accelerator-based SBCs no matter the size of data and the YOLO model used for inference and Jetson Nano showed the worst inference performance in terms of FPS. However, as shown in Table 3, the mean confidence, which refers to the average accuracy of all correctly identified objects throughout the inference process, of YOLO v3-tiny on RPi + NCS2 was 0%, whereas it was 57.9% on Jetson Nano and Jetson Xavier NX. On the contrary, the mean confidence of each YOLO network was the same between Jetson Nano and Jetson Xavier NX. This indicates that during the compilation process of the NCS2, the YOLO networks were changed by the MO to adapt YOLO models to NCS2's architecture. More specifically, as shown in Section 3.1, YOLOv3 and YOLOv3-tiny's feature extractor was a Darknet framework, which means that their branches at the end must end with the YOLO Region layer in to perform detection at different scales. The egion layer was first introduced in the Darknet framework. Other frameworks, including TensorFlow, do not have the Region implemented as a single layer, so every author of public YOLOv3 and YOLOv3-tiny model creates it using simple layers. For this reason, the main idea of YOLOv3 and YOLOv3-tiny model conversion to IR is to cut off these custom Region-like parts of the model and use the region layer to complete the model when necessary, which leads to the different inference performance of RPi + NCS2 when compared with Jetson Nano and Jetson Xavier NX. Therefore we can conclude that the inference performance of a accelerator-based SBC mainly depends on the AI model. For an intelligence application developed on a accelerator-based SBC, we not only have to choose a platform with suitable performance and price, but also consider the adaptation of the model to it, especially for ASIC-based SBC.

4.1.2. Memory Usage

As the key resource of SBCs, memory usage can reflect the AI model carrying capacity of a SBC. As shown in Figure 5, the memory usage of YOLOv3-tiny is about 0.2GB less than that of YOLOv3 no matter whether it is on Jetson Nano, Jetson Xavier NX or RPi + NCS2 due to the fact that the feature extraction scales of YOLOv3-tiny are smaller than that of YOLOv3.

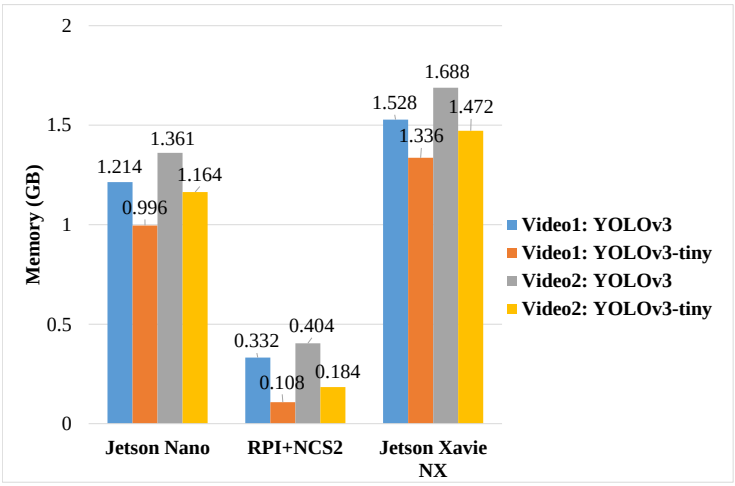


Figure 5. Memory usage on SBCs.

Moreover, since the RPi + NCS2 architecture will choose to offload the inference part of the model to the on-chip memory of NCS2 as much as possible to ensure low latency access, when we compare the memory usage of RPi + NCS2 with Jetson Nano or Jetson Xavier NX, we conclude that the memory usage of GPU-based SBC is much bigger than the

memory usage of ASIC-based SBC when using the same AI model, especially when using YOLOv3-tiny to detection Video1, the memory usage of Jetson Nano and Jetson Xavier NX is 9 times and 12 times larger than the memory usage of RPi + NCS2, respectively.

On the other hand, the GPU-based SBCs, i.e., Jetson Nano and Jetson Xavier NX, process various tasks through the cooperation of CPU and GPU, and share memory resources between CPU and GPU. When the model inference occupies more memory resources, other tasks such as user interface operations will be difficult to perform. In contrast, NCS2 is more friendly to some low-performance SBCs that need to run other processes, especially when multiple NCS2s can collaborate to complete inference tasks.

4.1.3. Energy Consumption

Energy consumption is a key performance metric for intelligence edges and intelligence applications such as a rescue UAV cruising on the beach and thousands of smart cameras in the city. The energy consumption of each SBC can be obtained by multiplying Power and Time in Table 3.

Idle power of the accelerator-based SBCs was measured before all the benchmark tests. As shown in Table 4, compared with the GPU-based SBCs, RPi + NCS2 takes more power when idle, and the idle power of NCS2 is 0.64 W, which is much higher than that of GPU.

Table 4. Idle power of accelerator-based SBCs (W).

Accelerator-Based SBC	Idle Total	Idle CPU	Idle GPU/NCS2
RPi + NCS2	3.16	-	0.64
Jetson Nano	2.60	0.4	0.04
Jetson Xavier	3.15	0.7	0.08

Interestingly, we found that changing the model used and the size of the input data for inference had little effect on the average power of the accelerator-based SBC. Jetson Nano has lower FPS, which leads to a longer inference time. Therefore, as shown in Figure 6, Jetson Nano always consumes more energy for the model inference. In contrast, though Jetson Xavier NX has higher average power (reaches 15.2 W when running YOLOv3), its extremely high performance enables it to complete inference work in a very short time, so Jetson Xavier NX always consumes lower energy for model inference. In terms of RPi + NCS2, it always has lower average power, no matter which model is running for inference, but its FPS is higher than Jetson Nano, which means it has better energy consumption performance than Jetson Nano.

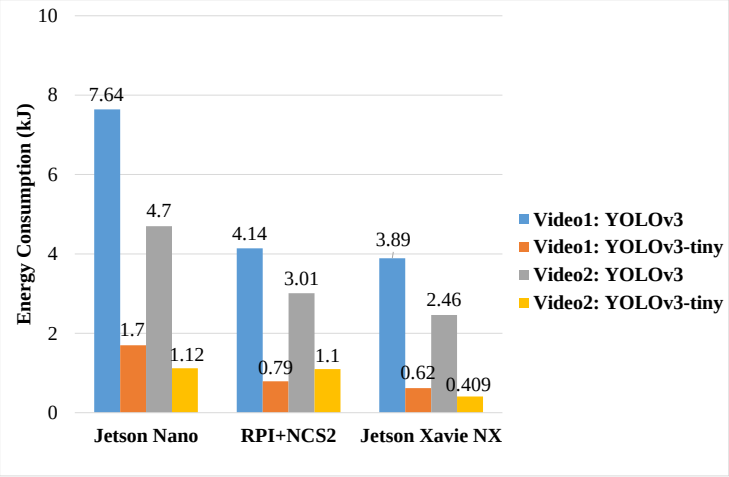


Figure 6. Energy consumption of different accelerator-based SBCs.

4.2. Performance Comparison

To further discuss the inference performance comparison, Jetson Nano was selected to run with more benchmarks analysis. Jetson Nano was selected because it is cheaper and it meets the requirement of edge intelligence SBCs. Two different benchmark analyses were conducted on Jetson Nano: (1) as shown in Table 5, the *S* value of YOLOv3 and YOLOv3-tiny were changed to evaluate the influence of different resize windows of YOLO to inference performance; (2) advanced versions of YOLO, i.e., YOLOv4 and YOLOv4-tiny, were run on Jetson and the benchmark results are shown in Table 6.

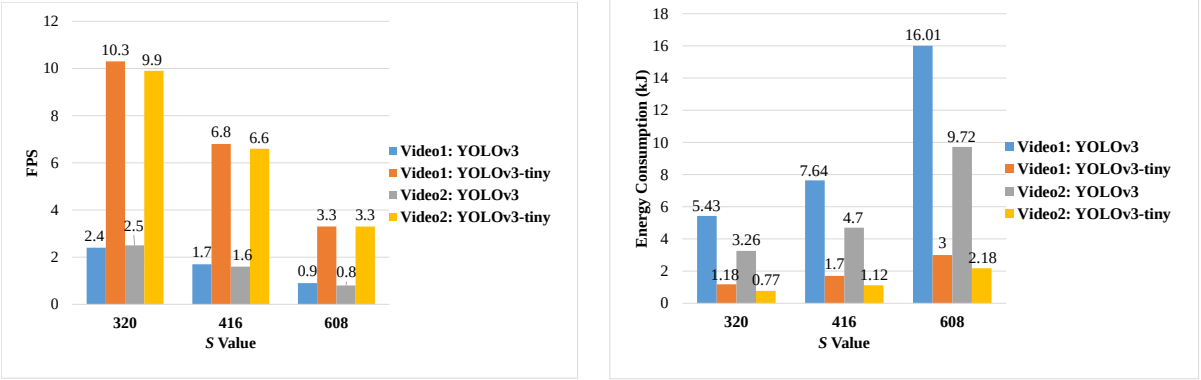
Table 5. Inference performance of YOLOv3 and YOLOv3-tiny on Jetson Nano when the *S* value changes.

	Models	*S* Value	FPS	CPU Usage (%)	Memory Usage (GB)	GPU Power (W)	Energy Consumption (kJ)
Video1	**YOLOv3**	320	2.4	26.8	0.86	3.7	5.43
		608	0.9	25.5	1.19	3.9	16.01
	YOLOv3-tiny	320	10.3	30.5	1.00	3.7	1.18
		608	3.3	27.0	1.01	3.9	3.00
Video2	**YOLOv3**	320	2.5	30.8	1.31	3.3	3.26
		608	0.8	26.8	1.31	3.4	9.72
	YOLOv3-tiny	320	9.9	41.3	1.13	3.3	0.77
		608	3.3	31.8	1.13	3.4	2.18

The results in Table 5 show the inference performance of YOLOv3 and YOLOv3-tiny with $S = 320$ and $S = 608$ when running on Jetson Nano. During the detection of each group of Video1 or Video2, as the resize window became smaller, i.e., the *S* value became smaller, the FPS increased, resulting in an increase in the number of frames for the CPU to read and process in per unit time, and an increase in CPU usage, which posed a great challenge to the performance of SBC's CPU. Furthermore, there is an interesting phenomenon: when we put our focus on the GPU Power in Tables 5 and 6, the influence of model change on GPU Power, which also reflects the GPU usage in some degree, was obviously less than that of input data size. We can conclude that larger frame size does not increase the use of GPU, but greatly increases the use of CPU and memory. Therefore, if it is necessary to use GPU to perform many tasks other than inference, minimizing the size of input data is an important method to keep the GPU-based SBCs running stably. At the same time, as shown in Figure 7, with the increase in *S* value, inference time increases while FPS decreases, which leads to a sharp increase in energy consumption, especially for YOLOv3. This means that when using AI models to deploy intelligence applications on edge devices, we must make a rigorous trade-off between inference accuracy and inference speed; YOLOv4-tiny is a perfect example.

By comparing the results in Tables 3 and 6, the performance of YOLOv4-tiny has caught our attention. Although the mean confidence of YOLOv4-tiny is not as high as that of YOLOv4 and YOLOv3 running inference on Jetson Nano, the FPS and mean confidence of YOLOv4-tiny are greatly improved compared to YOLOv3-tiny with a slight increase in energy consumption. As introduced in Section 3.1, YOLOv4-tiny is a simplified version of YOLOv3. It has a more optimized structure than YOLOv3-tiny, but at the same time has a smaller model size than YOLOv3. Furthermore, when using TensorRT for GPU acceleration, the inference speed on GPU-based devices of YOLO networks can be increased to more than twice the original amount [45]. In this respect, for SBCs with limited resources, it is important to choose a model with a more optimized structure to better realize edge intelligence on the chosen edge devices. Han et al. [46] proposed scaleable convolutional

blocks to address the problem of limiting the maximum number of kernels for real-time object detection on edge computing devices. At first, they chose three edge devices to determine the maximum number of kernels on the convolutional layer. Then they used the proposed scaleable convolutional blocks to design three Scalable and Fast YOLO (SF-YOLO) models, which had two times faster processing speed compared with YOLOv3-tiny, but the same accuracy.



(**a**) FPS Performance (**b**) Energy Consumption

Figure 7. FPS and energy consumption of different S value. (**a**) FPS Performance; (**b**) energy consumption.

Table 6. Inference performance of YOLOv4 and YOLOv4-tiny with $S = 416$ runs on Jetson Nano.

	Models	Mean Confidence (%)	FPS	CPU Usage (%)	Memory Usage (GB)	GPU Power (W)	Energy Consumption (kJ)
Video1	**YOLOv4**	94.2	1.6	26.8	1.22	4.0	7.92
	YOLOv4-tiny	67.1	15.0	34.3	0.98	3.9	0.86
Video2	**YOLOv4**	75.6	1.6	29.5	1.36	3.5	3.78
	YOLOv4-tiny	61.9	14.2	54.5	1.10	3.4	0.57

5. Conclusions

In this study, we benchmark and analyze the performance of three accelerator-based SBCs for the YOLO models to provide useful enlightenment for the development of intelligent applications and intelligence edge devices. Before we choose a suitable platform to build up a intelligence application, we must consider the following two aspects. Firstly, ASIC accelerators are low-performance and SBC friendly. Intel NCS2, especially with the feature that allows multiple NCS2s to collaborate to complete inference tasks, can make a low-computing-power SBC capable of running a high-performance CNN model while occupying very few board resources. However, adapting an AI model to the architecture of the ASIC-accelerator SBC is the most important step in the inference process of running the AI model. A typical example is that the reason why the mean confidence of the YOLOv3-tiny when inferencing Video1 on RPi + NCS2 is 0%, while on other devices it is 57.9%, is that the Openvino toolkit's MO only connects Region layers that are cut off in the PB model when necessary. Moreover, if the AI model we used is not available in Intel NCS2's Open Model Zoo, the workload of the intelligence application deployment process will be greatly increased by using Intel NCS2. Secondly, when implementing smart applications on GPU-based SBCs, due to the memory sharing between the CPU and GPU, the architecture and related parameters of the AI model must be carefully designed to make a trade-off between inference accuracy and inference speed. Otherwise, good inference performance cannot be obtained, and the high CPU and memory usage while

inferencing will not allow SBC to perform any tasks other than model inference. Besides this, the power of the two Jetson devices when inferencing is always higher than that of RPi + NCS2; for example: the average power of the Jetson Xavier NX is 2.5 times higher than that of the RPi + NCS2, which means these two GPU-based SBCs are not suitable for applications that require long-term work: they will create more energy consumption and their working stability cannot be guaranteed. In future work, YOLO networks will be implemented in FPGA-based SBCs to further compare the inference performance of different intelligence edges.

Author Contributions: Conceptualization, H.F.; data curation, G.M.; project administration, S.Z.; validation, P.Z.; investigation, T.Y.; writing—original draft, H.F. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded in part by the (Key Area) Project of Department of Education of Guangdong Province, China, under Project No.2021ZDZX4008, in part by the Guangdong Provincial Department of Science and Technology, China, under Project No.2020B1212030002, in part by the Shenzhen Science and Technology Innovation Commission, China, under Projects No.KQTD20180412181337494 and No.JSGG20201103095805016, in part by the National Key Research and Development Program, China, under Subject No.2019YFF0216602, and in part by the Graduate Innovation and Development Fund Project of Shenzhen University.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

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

Abbreviations

The following abbreviations are used in this manuscript:

SBCs	Single-board computers
AI	artificial intelligence
IoT	Internet of things
5G	5th Generation mobile networks
eMBB	enhanced mobile broadband
uRLLC	ultra reliable low latency communications
mMTC	Massive Machine Type Communication
EI	Edge Intelligence
ASIC	Application Specific Integrated Circuit
FPGA	Field Programmable Gate Array
NCS2	Neural Compute Stick2
UAV	Unmanned Aerial Vehicle
RPi	Raspberry Pi
YOLO	You Only Look Once
SoC	System on Chip
IR	Intermediate Representation
FP16	Half-Precision
FP32	Sigle-Precision
PB	Protocol Buffer
MO	Model Optimizer
API	Application Programming Interface
FPS	Frames-Per-Second

References

1. Redana, S.; Bulakci, O.; Mannweiler, C.; Gallo, L.; Kousaridas, A.; Navratil, D.; Tzanakaki, A.; Gutiwrrez, J.; Karl, H.; Hasselmeyer, P.; et al. 5G PPP Architecture Working Group—View on 5G Architecture, Version 3.0. 2019. Available online: https://zenodo.org/record/3265031#.Yj1fbDURXIU (accessed on 9 August 2021).
2. Pokhrel, S.R.; Ding, J.; Park, J.; Park, O.S.; Choi, J. Towards Enabling Critical mMTC: A Review of URLLC within mMTC. *IEEE Access* **2020**, *8*, 131796–131813. [CrossRef]

3. Ericsson. IoT Connections Outlook: In 2026, NB-IoT and Cat-M Technologies Are Expected to Make Up 45 percent of All Cellular IoT Connections. Available online: https://www.ericsson.com/en/mobility-report/dataforecasts/iot-connections-outlook (accessed on 9 August 2021).
4. Khan, L.U.; Yaqoob, I.; Tran, N.H.; Kazmi, S.M.A.; Dang, T.N.; Hong, C.S. Edge-Computing-Enabled Smart Cities: A Comprehensive Survey. *IEEE Internet Things J.* **2020**, *7*, 10200–10232. [CrossRef]
5. Artunedo Guillen, D.; Sayadi, B.; Bisson, P.; Wary, J.P.; Lonsethagen, H.; Anton, C.; de la Oliva, A.; Kaloxylos, A.; Frascolla, V. Edge Computing for 5G Networks—White Paper. 2020. Available online: https://zenodo.org/record/3698117#.Yj1fpDURXIU (accessed on 9 August 2021).
6. Naouri, A.; Wu, H.; Nouri, N.A.; Dhelim, S.; Ning, H. A Novel Framework for Mobile-Edge Computing by Optimizing Task Offloading. *IEEE Internet Things J.* **2021**, *8*, 13065–13076. [CrossRef]
7. Wang, F.; Zhang, M.; Wang, X.; Ma, X.; Liu, J. Deep Learning for Edge Computing Applications: A State-of-the-Art Survey. *IEEE Access* **2020**, *8*, 58322–58336. [CrossRef]
8. Zhou, Z.; Chen, X.; Li, E.; Zeng, L.; Luo, K.; Zhang, J. Edge Intelligence: Paving the Last Mile of Artificial Intelligence With Edge Computing. *Proc. IEEE* **2019**, *107*, 1738–1762. [CrossRef]
9. Wang, X.; Han, Y.; Leung, V.C.M.; Niyato, D.; Yan, X.; Chen, X. Convergence of Edge Computing and Deep Learning: A Comprehensive Survey. *IEEE Commun. Surv. Tutor.* **2020**, *22*, 869–904. [CrossRef]
10. Mittal, S. A Survey on optimized implementation of deep learning models on the NVIDIA Jetson platform. *J. Syst. Archit.* **2019**, *97*, 428–442. [CrossRef]
11. Kang, P.; Lim, S. A Taste of Scientific Computing on the GPU-Accelerated Edge Device. *IEEE Access* **2020**, *8*, 208337–208347. [CrossRef]
12. Mittal, S.; Vaishay, S. A survey of techniques for optimizing deep learning on GPUs. *J. Syst. Archit.* **2019**, *99*, 101635. [CrossRef]
13. Park, J.; Samarakoon, S.; Bennis, M.; Debbah, M. Wireless Network Intelligence at the Edge. *Proc. IEEE* **2019**, *107*, 2204–2239. [CrossRef]
14. Nair, D.; Pakdaman, A.; Plöger, P. Performance Evaluation of Low-Cost Machine Vision Cameras for Image-Based Grasp Verification. *arXiv* **2020**, arXiv:2003.10167.
15. Intel. Intel® Movidius™ Myriad™ X VPUs. Available online: https://www.intel.com/content/www/us/en/artificialintelligence/movidius-myriad-vpus.html (accessed on 9 August 2021).
16. LLC, G. Coral Dev Board Datasheet Version 1.3. Available online: https://coral.ai/static/files/Coral-DevBoard-datasheet.pdf (accessed on 9 August 2021).
17. Torres-Sánchez, E.; Alastruey-Benedé, J.; Torres-Moreno, E. Developing an AI IoT application with open software on a RISC-V SoC. In Proceedings of the 2020 XXXV Conference on Design of Circuits and Integrated Systems (DCIS), Segovia, Spain, 18–20 November 2020; pp. 1–6. [CrossRef]
18. Attaran, N.; Puranik, A.; Brooks, J.; Mohsenin, T. Embedded Low-Power Processor for Personalized Stress Detection. *IEEE Trans. Circuits Syst. II Express Briefs* **2018**, *65*, 2032–2036. [CrossRef]
19. Basulto-Lantsova, A.; Padilla-Medina, J.A.; Perez-Pinal, F.J.; Barranco-Gutierrez, A.I. Performance comparative of OpenCV Template Matching method on Jetson TX2 and Jetson Nano developer kits. In Proceedings of the 2020 10th Annual Computing and Communication Workshop and Conference (CCWC), Las Vegas, NV, USA, 6–8 January 2020; pp. 0812–0816. [CrossRef]
20. Jo, J.; Jeong, S.; Kang, P. Benchmarking GPU-Accelerated Edge Devices. In Proceedings of the 2020 IEEE International Conference on Big Data and Smart Computing (BigComp), Busan, Korea, 19–22 February 2020; pp. 117–120. [CrossRef]
21. Suzen, A.A.; Duman, B.; Sen, B. Benchmark Analysis of Jetson TX2, Jetson Nano and Raspberry PI using Deep-CNN. In Proceedings of the 2020 International Congress on Human-Computer Interaction, Optimization and Robotic Applications (HORA), Ankara, Turkey, 26–27 June 2020; pp. 1–5. [CrossRef]
22. Aleksandrova, O.; Bashkov, Y. Face recognition systems based on Neural Compute Stick 2, CPU, GPU comparison. In Proceedings of the 2020 IEEE 2nd International Conference on Advanced Trends in Information Theory (ATIT), Kyiv, Ukraine, 25–27 November 2020; pp. 104–107. [CrossRef]
23. Antonini, M.; Vu, T.H.; Min, C.; Montanari, A.; Mathur, A.; Kawsar, F. Resource Characterisation of Personal-Scale Sensing Models on Edge Accelerators. In Proceedings of the First International Workshop on Challenges in Artificial Intelligence and Machine Learning for Internet of Things (AIChallengeIoT'19), Harbin, China, 25–26 May 2019; Association for Computing Machinery: New York, NY, USA, 2019; pp. 49–55. [CrossRef]
24. Sahu, P.; Yu, D.; Qin, H. Apply lightweight deep learning on internet of things for low-cost and easy-to-access skin cancer detection. In *Medical Imaging 2018: Imaging Informatics for Healthcare, Research, and Applications*; Zhang, J., Chen, P.H., Eds.; International Society for Optics and Photonics, SPIE: Bellingham, DC, USA, 2018; Volume 10579, pp. 254–262. [CrossRef]
25. Srinivasan, V.; Meudt, S.; Schwenker, F. Deep Learning Algorithms for Emotion Recognition on Low Power Single Board Computers. In *Multimodal Pattern Recognition of Social Signals in Human-Computer-Interaction*; Schwenker, F., Scherer, S., Eds.; Springer International Publishing: Cham, Switzerland, 2019; pp. 59–70.
26. Jung, T.H.; Cates, B.; Choi, I.K.; Lee, S.H.; Choi, J.M. Multi-Camera-Based Person Recognition System for Autonomous Tractors. *Designs* **2020**, *4*, 54. [CrossRef]
27. Zheng, Y.Y.; Kong, J.L.; Jin, X.B.; Wang, X.Y.; Su, T.L.; Zuo, M. CropDeep: The Crop Vision Dataset for Deep-Learning-Based Classification and Detection in Precision Agriculture. *Sensors* **2019**, *19*, 1058. [CrossRef] [PubMed]

28. Horng, G.J.; Liu, M.X.; Chen, C.C. The Smart Image Recognition Mechanism for Crop Harvesting System in Intelligent Agriculture. *IEEE Sens. J.* **2020**, *20*, 2766–2781. [CrossRef]
29. Ferdowsi, A.; Challita, U.; Saad, W. Deep Learning for Reliable Mobile Edge Analytics in Intelligent Transportation Systems: An Overview. *IEEE Veh. Technol. Mag.* **2019**, *14*, 62–70. [CrossRef]
30. Arabi, S.; Haghighat, A.; Sharma, A. A deep-learning-based computer vision solution for construction vehicle detection. *Comput.-Aided Civ. Infrastruct. Eng.* **2020**, *35*, 753–767. [CrossRef]
31. Huang, X.; Wang, P.; Cheng, X.; Zhou, D.; Geng, Q.; Yang, R. The ApolloScape Open Dataset for Autonomous Driving and Its Application. *IEEE Trans. Pattern Anal. Mach. Intell.* **2020**, *42*, 2702–2719. [CrossRef]
32. Redmon, J.; Farhadi, A. YOLOv3: An Incremental Improvement. *arXiv* **2018**, arXiv:1804.02767.
33. Bochkovskiy, A.; Wang, C.; Liao, H.M. YOLOv4: Optimal Speed and Accuracy of Object Detection. *arXiv* **2020**, arXiv:2004.10934.
34. Deng, S.; Zhao, H.; Fang, W.; Yin, J.; Dustdar, S.; Zomaya, A.Y. Edge Intelligence: The Confluence of Edge Computing and Artificial Intelligence. *IEEE Internet Things J.* **2020**, *7*, 7457–7469. [CrossRef]
35. Li, E.; Zhou, Z.; Chen, X. Edge Intelligence: On-Demand Deep Learning Model Co-Inference with Device-Edge Synergy. In Proceedings of the 2018 Workshop on Mobile Edge Communications (MECOMM'18), Budapest, Hungary, 20 August 2018; Association for Computing Machinery: New York, NY, USA, 2018; pp. 31–36. [CrossRef]
36. Adarsh, P.; Rathi, P.; Kumar, M. YOLO v3-Tiny: Object Detection and Recognition using one stage improved model. In Proceedings of the 2020 6th International Conference on Advanced Computing and Communication Systems (ICACCS), Coimbatore, India, 6–7 March 2020; pp. 687–694. [CrossRef]
37. Huang, R.; Pedoeem, J.; Chen, C. YOLO-LITE: A Real-Time Object Detection Algorithm Optimized for Non-GPU Computers. In Proceedings of the 2018 IEEE International Conference on Big Data (Big Data), Seattle, WA, USA, 10–13 December 2018; pp. 2503–2510. [CrossRef]
38. NVIDIA Jetson Nano. Available online: https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/jetson-nano (accessed on 9 August 2021).
39. NVIDIA Jetson Xavier NX. Available online: https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/jetson-xavier-nx/ (accessed on 9 August 2021).
40. Neural Compute Stick 2 Documentation. Available online: https://software.intel.com/content/www/us/en/develop/articles/get-started-with-neural-compute-stick.html (accessed on 9 August 2021).
41. AlexeyAB. Darknet. Available online: https://github.com/AlexeyAB/darknet#pre-trained-models (accessed on 9 August 2021).
42. COCO. Common Objects in Context. Available online: https://cocodataset.org (accessed on 9 August 2021).
43. Intel. Intel Distribution of OpenVINO Toolkit. Available online: https://docs.openvinotoolkit.org (accessed on 9 August 2021).
44. Lun, W. Sample Videos. Available online: https://gitee.com/ve2102388688/sample-videos/tree/master (accessed on 9 August 2021).
45. Jing, Y.; Wu, T.; Li, J.; Zhang, Z.; Gao, C. GPU acceleration design method for driver's seatbelt detection. In Proceedings of the 2019 14th IEEE International Conference on Electronic Measurement Instruments (ICEMI), Changsha, China, 1–3 November 2019; pp. 949–953. [CrossRef]
46. Han, B.G.; Lee, J.G.; Lim, K.T.; Choi, D.H. Design of a Scalable and Fast YOLO for Edge-Computing Devices. *Sensors* **2020**, *20*, 6779. [CrossRef] [PubMed]