# Comparison of edge computing platforms for hardware acceleration of AI: Kria KV260, Jetson Nano and RTX 3060

Master of Science in Technology Thesis
University of Turku
Department of Computing
Robotics and Autonomous Systems
Turku Intelligent Embedded and Robotic Systems (TIERS) Lab
2024
Sergio Aranda Lizano

Supervisors:
MSc. Minh Nguyen
Prof. Tomi Westerlund

UNIVERSITY OF TURKU
Department of Computing

SERGIO ARANDA LIZANO: Comparison of edge computing platforms for hardware acceleration of AI: Kria KV260, Jetson Nano and RTX 3060

Master of Science in Technology Thesis, 51 p.
Robotics and Autonomous Systems
Turku Intelligent Embedded and Robotic Systems (TIERS) Lab
May 2024

As edge computing platforms become more extense and newer companies join the field, it becomes harder to know which platform to use in any specific case. These systems are often packed with a broad array of different computation architectures and different hardware acceleration technologies, this can be confusing at the moment of the election to integrate them as hardware accelerators in larger designs. Due to the efficiency of these platforms, they often enable creative problem-solving approaches to robotics and other fields where computation on the edge was not common that long ago. This thesis delves into leading hardware accelerators, analyzing the performance and power usage of three platforms: Kria KV260, Jetson Nano and RTX 3060. Experiments were conducted with two neural network models-ResNet-50 and YOLO-trained for image identification tasks. Our findings highlight the FPGA-based platform's superior efficiency in terms of inference speed per watt compared to the other platforms.

Keywords: Edge Computing, Hardware Acceleration, IoT, ASIC, FPGA, AI, DNN, GPU

# Contents

# List of Figures

# List of Tables

# List Of Acronyms

| | |
|---|---|
| **APU** | Application Processing Unit |
| **ASIC** | Application-Specific Integrated Circuit |
| **BRAM** | Block Random Access Memory |
| **CPU** | Central Processing Unit |
| **CUDA** | Compute Unified Device Architecture |
| **DNN** | Deep Neural Network |
| **FPGA** | Field Programmable Gate Array |
| **FPS** | Frames Per Second |
| **GPU** | Graphical Processing Unit |
| **HDL** | Hardware Description Language |
| **HLS** | High-Level Synthesis |
| **LUT** | Look Up Tables |
| **FF** | Flip Flop |
| **MUX** | Multiplexer |
| **SSI** | Small Scale Integration Chip |
| **PROM** | Programmable Read-Only Memory |
| **FA** | Full Adder |
| **DSP** | Digital Signal Processor |
| **SM** | Streaming Multiprocessors |
| **PLD** | Programmable Logic Device |

| | |
|---|---|
| **YOLO** | You Only Look Once |
| **ResNet** | Residual Learning Model |
| **CMOS** | Complementary Metal-Oxide-Semiconductor |
| **PCI** | Peripheral Component Interconnect |
| **RTX** | Ray Tracing |
| **CRT** | Cathode-Ray Tube |
| **INT8** | 8-bit integer |

# 1 Introduction

The Internet of Things has evolved and nowadays more and more devices are being connected to the network and have small computing capabilities, these devices usually leverage the use of cloud computing, using cloud services to do the computation of the data they receive from their sensors. But a lot of new applications that are being deployed need the computation to be done at the edge, the connection between the network and the real world, this need created edge computing.

Edge computing also focuses on the efficiency of processing the data at the edge of the network instead of cloud computing, an example of this need was [1]. There are multiple needs that edge computing solves, faster speed while processing the data rather than using broadband to send the data to the cloud and wait for the compute. For example, an autonomous vehicle generates a lot of data each second and it needs it to be processed in real-time which means we cannot access cloud services for its processing [2].

Edge computing also solves the change from data consumer to producer, as more people generate new data and not only consume it on their devices. In Fig 1.1, we describe in a simple image the edge computing paradigm.

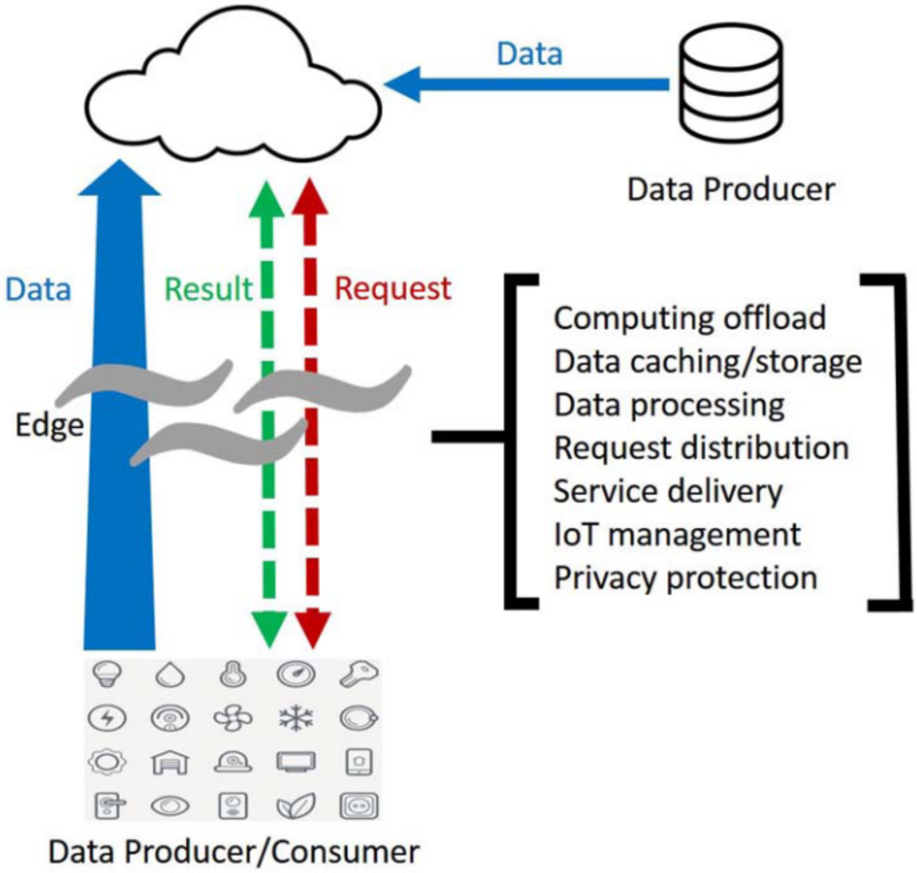


Figure 1.1: Edge computing paradigm [3]

The new advent of AI has also created the need for new hardware to process and train the new AI models and also to acquire the data needed for this [4]. A lot of the new models and autonomous systems require the computation to be done by edge computing, so the new hardware has to comply with the edge computing paradigm and constraints such as low power consumption, small size, and passive cooling.

With the advent of AI another field saw its rise, autonomous robots, we can see that the creation of new AI models has helped robots to navigate and interpret the environment faster and with better accuracy, with most models being able to be executed in real-time. For this, we need the use of hardware accelerators for AI in edge computing.

This thesis aims to assess and contrast the new hardware intended for processing and training AI models in both edge computing and for autonomous robots, such as Field Programmable Gate Arrays (FPGAs) and Graphic Processing Units (GPUs). Our base criteria of evaluation focus on power consumption, efficacy, development process, time of development and temperature of operation. In our work, we are going to use multiple models to evaluate given that their intended application is in autonomous robotics and their complexity.

## 1.1 Significance and Motivation

This thesis holds significance due to the need to acquire empirical data and evidence that will support which hardware accelerator between the Kria KV260, Jetson Nano and RTX 3060 is best for the models in question and why it is better for those models, this will be achieved with rigorous testing and evaluation of such platforms. The motivation of this thesis was based on the high volume of research and development of edge computing and autonomous robotics, and we will be addressing the next research questions:

1. What type of hardware accelerator to choose in a new development?
2. Can FPGAs perform better than GPUs in these types of tasks, based on our criteria of evaluation?
3. How efficient are the three platforms in their tasks, can the difference in efficiency modify the election between them?

## 1.2 Related works

The literature shows several studies that have mentioned or used different hardware accelerator platforms, including the work of a colleague [5], studies such as [6] which employ different platforms than the ones for this thesis and [7] which employ similar platforms in a different context.

Studies use tools dedicated to edge computing such as [8], which are tools we aim to use in our research. Despite using the same tools their research is based on optimization, and our main research point is the evaluation of the platforms and their efficiency with some DNN models.

The platforms used in this thesis have appeared in multiple research papers with a similar usage as our topic, for example, [9] shows the usage of the Kria KV260 as a hardware accelerator for the random forest algorithm. In another instance [10] we observe how the Jetson Nano performs real-time inference of the You Only Look Once (YOLO) model, one of the models we will implement. In addition, we can see the usage of the RTX 3060 platform on a case of real-time inference in [11], which provides measurements similar to our thesis.

## 1.3 Contribution

The purpose of this document is to detail the implementation of hardware accelerators for AI in a way that is both instructive and delivers value to the reader. This research provides the exploration of different edge computing platforms with different technologies and different DNN models that were used extensively in the research community. In particular, we compare performance, power consumption, temperature during operation, efficiency, performance-to-cost ratio, and more between three different hardware: a Jetson Nano, a Kria KV260, and an RTX 3060.

## 1.4 Structure

The thesis follows this organization:

- Chapter 2 introduces the necessary knowledge and relevant literature on the topic.
- In chapter 3, we describe in detail the design followed for this process, going into

higher detail on the specialities of the FPGA platform and how it differs from the rest of the hardware to evaluate.

- In chapter 4, we explain the implementation of the system.
- Chapter 5 presents the data re-collected during the experiment with ease-to-follow graphs and the analysis of the collected data.
- And lastly, chapter 6 presents the conclusion and future work directions.

# 2 Background

The advance in edge computing has been incredible, with leaps in performance, but battery technologies have not been able to keep up, this has created a need for power-efficient and specialised hardware, in our case in robotics, we have actuators, sensors and processors, we are going to dive into the solutions that specialise in robotics systems in this context.

## 2.1 Hardware accelerators

Hardware accelerators are platforms designed to assist a specific purpose, improving the performance in some parts of a process. Usually, these accelerators are designed to be the target of offloading intensive computation and performing it in parallel and concurrently. In this field different types of solutions have sprouted during the research, the most used are GPUs, ASICs and FPGAs [12]. We will overview these different solutions in this next section.

## 2.2 FPGAs

Programmable devices have been in digital design since the beginning of computing technologies, an example is a Programmable Read-Only Memory (PROM), the last evolution of programmable device is called FPGA, which work in a variety of applications such as the simulation of ASIC, implementation of Random Logic, Replacement of Small Scale

Integration (SSI) Chips for Random Logic, prototyping and in specific compute engines for FPGAs [13]. The implementation process of FPGAs follows different steps, as shown in Fig 2.1.

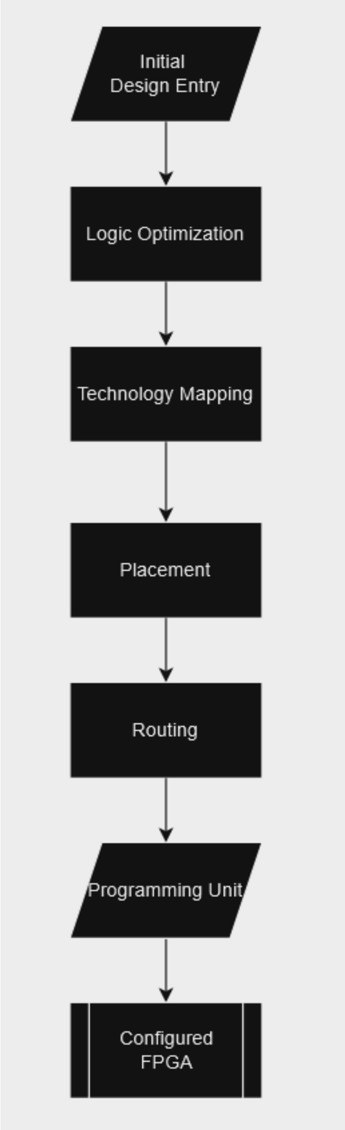


Figure 2.1: FPGAs implementation process [13]

*Copyright © 1992, Springer Nature Switzerland AG. Part of Springer Nature.*

The first step is to design the initial logic entry circuit, this can be done either with a program to draw electronic schematics or with hardware design layer (HDL) programming languages, such as VHDL or Verilog. After that, usually, the code expressions are optimized by different tools which reduce the area and improve the speed of the circuit. Subsequently, it gets transformed into a circuit of FPGA logic blocks, this step is called technology mapping. Last, we generate the placement of the logic blocks inside the board and route the wires between logic blocks [13]. The routing depends on the different types of architecture inside the FPGA, the most common ones are illustrated in figure 2.2, namely, Symmetrical array, Row-based, Sea-of-gates and Hierarchical Programmable Logic Device (PLD).

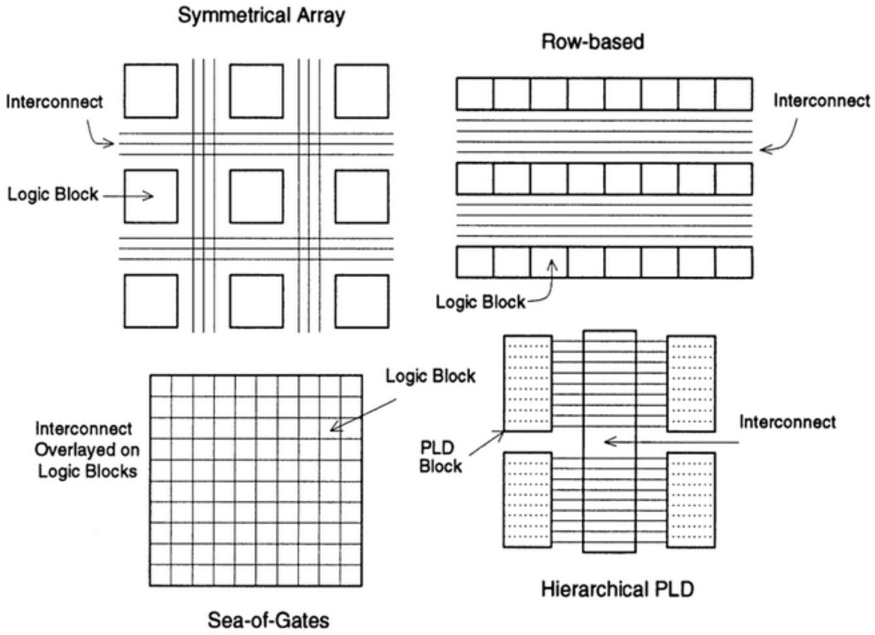


Figure 2.2: FPGAs different architectures [13]

Within the logic blocks, there are various components, as illustrated in figure 2.3. Specifically, it consists of Look-up Table (LUT), Flip-Flop (FF) to store the output of the LUT, Multiplexer (MUX) to control signal routing, and specialized hardware technologies that enhance arithmetic operation throughput, such as a Full Adder (FA).

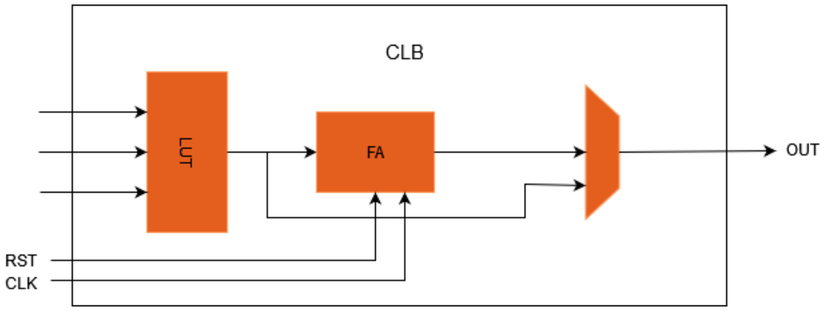


Figure 2.3: Internal structure of FPGA's Control logic block

Modern FPGAs not only include logic blocks, they also include other components such as different I/O communication chips, for example USB ports, Serial protocols; digital signal processors (DSPs); Analog-digital converters; Block Random Access Memory (BRAMs).

## 2.3 Graphical Processing Units

GPUs have a long history in computing architectures. At first, they were created to help the system produce images and output them into cathode-ray tube (CRT) displays, as older CPUs did not contain graphical units inside the silicon die. Since the display technology became more advanced, GPUs needed to compute more advanced operations. With the advances in architecture design, GPU can now support a wider range of operations[14]. Hence, it has provided engineers with the opportunity to employ those computation resources for other tasks. For instance, it was thought that by offloading small and repetitive tasks to the GPU, we can help relieve the stress on the CPU if those computations. Moreover, it could offset the latency between components using the highly parallelism capability of the GPUs.

This created the idea of General Purpose Computation using GPUs, in the beginning of this idea, programming into a GPU was hard and tedious. Nowadays, there are a lot of new platforms that provide ease of access to this domain. One of the most prominent vendor of GPU technology is Nvidia, which also provide a set of toolkits and libraries for developers who use their graphic cards.

We are going to explain the architecture of the GPUs and the reason for their performance in parallel computing.

### 2.3.1 GPUs architecture

The architecture of the GPU is constructed based on scalable arrays of Streaming Multiprocessors (SM) and the reason hardware parallelism is conquered is because the replication of this architectural block [15]. An SM contains a lot of components, as illustrated in the figure 2.4: Compute Unified Device Architecture (CUDA) core, registers and shared memory.

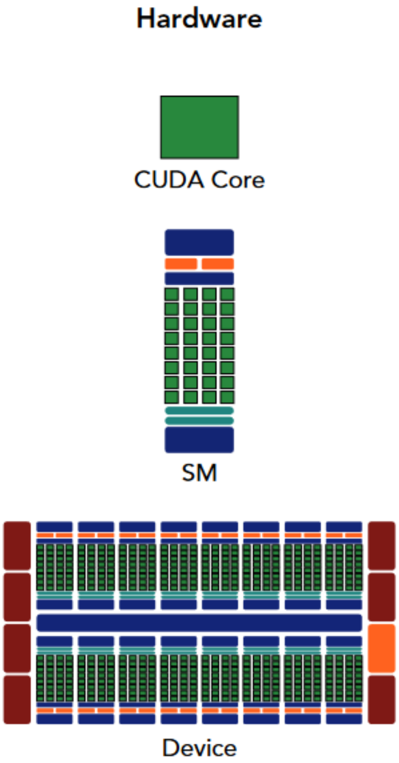


Figure 2.4: Insides of an SM [15]

*This figure is reproduced with permission from the book Professional CUDA C Programming by Cheng et al., publish by John Wiley & Sons*

Every SM within a GPU is designed to manage the simultaneous execution of hundreds of threads. Generally, there are multiple SMs per GPU. Upon launching a kernel grid, the thread blocks within that grid are allocated across the available SMs for execution. A thread block is formed by a specific number of threads, usually defined by the specific architecture of the GPU, as shown in Fig 2.5.

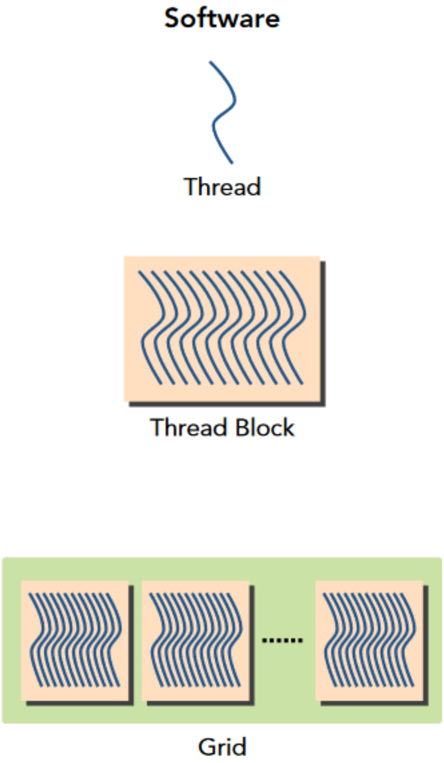


Figure 2.5: Thread architecture of CUDA [15]

*This figure is reproduced with permission from the book Professional CUDA C Programming by Cheng et al., publish by John Wiley & Sons*

CUDA follows a memory hierarchy, as demonstrated in the figure 2.6, where the CUDA memory models create unity between separate host and device memory systems. Additionally, the memory hierarchy can be modified for excellent performance. The levels in the hierarchy of the memory model are:

1. *Registers*: Registers take action when a variable is declared in a kernel. These registers are private to each thread. They are scarce resources and they have small hardware

limits.

2. *Local memory*: Variables that could be inside registers but do not fit anymore are stored in local memory. An example of those variables could be large arrays.

3. *Shared memory*: Shared memory is on-chip and has a faster response time than local memory. There is a small amount of shared memory for each SM. It is the base of inter-thread communication, the access to it has to be synchronized.

4. *Constant memory*: Constant memory is part of the device memory and can be cached to an SM constant cache. It is read-only and is usually utilized for mathematical formulas with constants.

5. *Texture memory*: Texture memory is part of the device memory and can be cached to SM read-only cache. It is optimized for 2D spatial locality.

6. *Global memory*: Global memory is part of the device memory and it contains other parts such as:

- L1 Cache, is individual per each SM.

- L2 Cache, is shared by all SM.

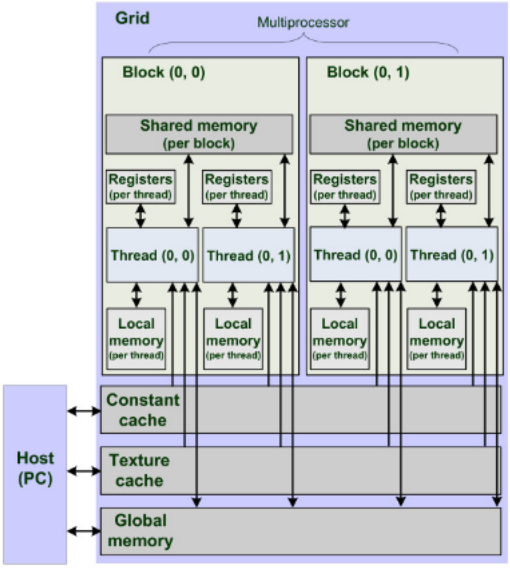


Figure 2.6: Memory hierarchy of CUDA [16]

*Copyright CC-2.0*

## 2.4 ASICs

ASIC is an IC designed for a particular application or end-use[17]. Different technologies are applied in the development of an ASIC, as illustrated in the figure 2.7.

1. *Full custom design*: Each logic cell inside the ASIC is designed from the ground up, starting at the transistor level to the higher level. In this case, the designer can exploit the custom design by wiring areas between logic cells to create compact functions using techniques such as over-the-cell routing.

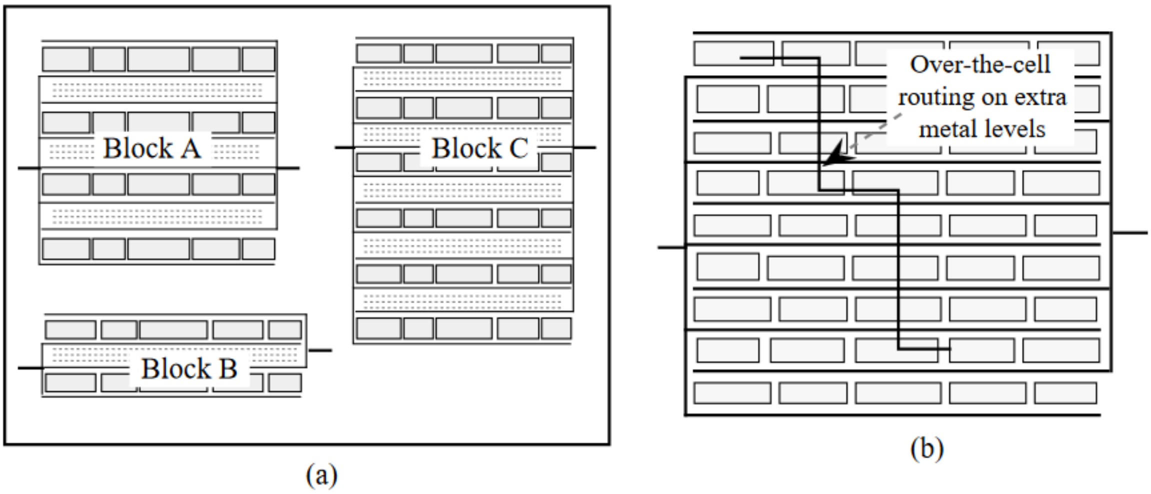


Figure 2.7: (a) Construction of larger blocks with custom cells. (b) Over-the-cell routing [17]

*Copyright © 1996 West Virginia University*

2. *Standard-cell ASIC technology*: This process uses predefined logic cells to create the ASIC, the final quality of the cells will depend on the library used [17].

3. *Gate array ASIC Technology*: The gate array technology [18] uses prefabricated wafers with simple gate cells, and the ASIC designer specifies the final metalization layer added to customize the gate array. An evolution of this process is the sea-of-gates technology, in the sea-of-gates more waves and layers are used, creating a sea-of-gates.

4. *Complementary Metal-Oxide-Semiconductor (CMOS) Circuits*: With the increase

of complexity in ASIC design, standard high-level functions have become more popular, such as Peripheral Component Interconnect (PCI) interfaces, RAM arrays, etc. Using this high-level functions to create a circuit takes advantage standard functions for an easier desing process [17]. A case of modern ASIC design for AI processing and modelling using the described technologies is the Tensor Processing Unit (TPU). In Fig 2.8 we can see the latest TPU architecture, inside it contains a tensor core, this tensor core contains multiple units:

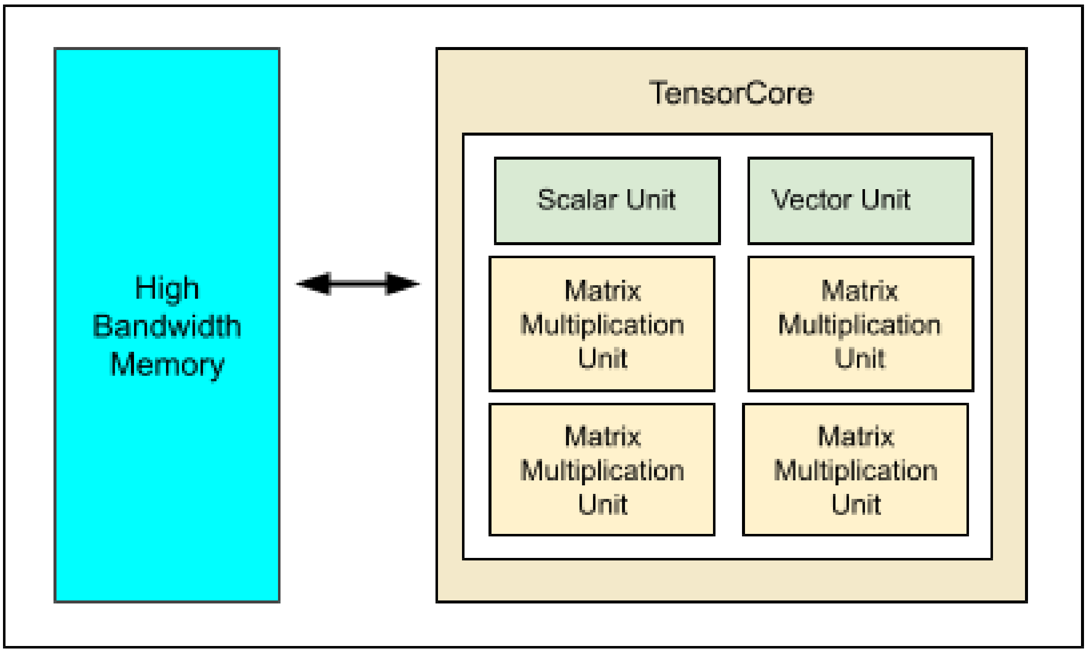

High Bandwidth Memory
TensorCore
Scalar Unit
Vector Unit
Matrix Multiplication Unit
Matrix Multiplication Unit
Matrix Multiplication Unit
Matrix Multiplication Unit

Figure 2.8: V5-TPU architecture [19]

*Copyright CC-4.0*

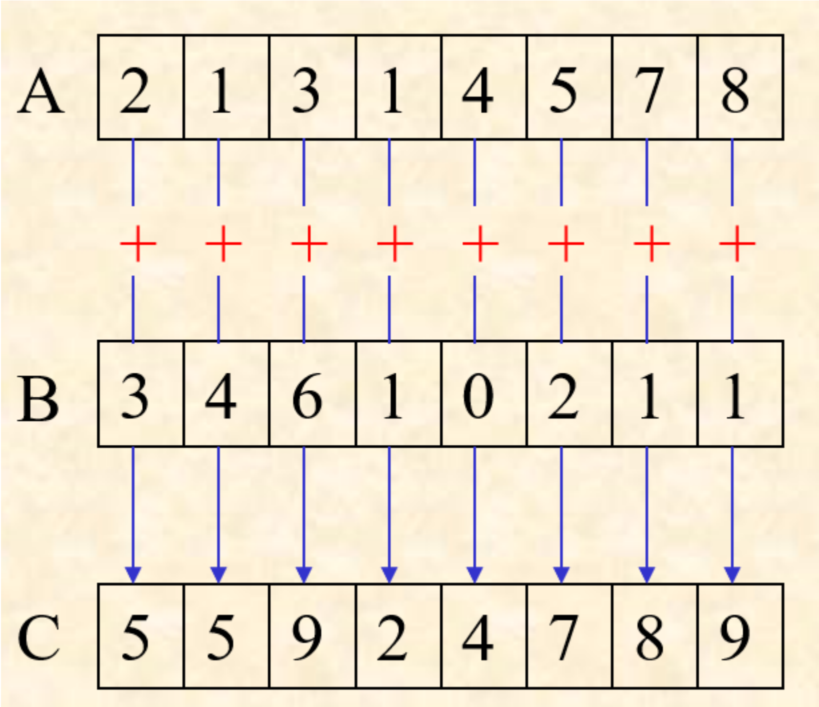
*Portions of this page are reproduced from work created and shared by Google and used according to terms described in the Creative Commons 4.0 Attribution License.*

1. *Scalar Unit*: Inside are performed scalar calculations, such as 2.1

$$a \cdot b = |a|\,|b| \cos\theta \tag{2.1}$$

2. *Vector Unit*: A vector unit is a structure that computes vector-specific operations, an

example in the figure 2.9. The most important part about vector units is that they operate on the whole vector in one cycle, speeding up the computation.



|   |   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|---|
| A | 2 | 1 | 3 | 1 | 4 | 5 | 7 | 8 |
|   | + | + | + | + | + | + | + | + |
| B | 3 | 4 | 6 | 1 | 0 | 2 | 1 | 1 |
| C | 5 | 5 | 9 | 2 | 4 | 7 | 8 | 9 |

Figure 2.9: Basic vector unit operation

3. *Matrix multiplication unit*: The matrix multiplication units follow the same idea as a vector unit, but instead of containing multiple operations it focuses only on matrix multiplication, which makes it fast and efficient.

## 2.5 Deep Neural Network Models (DNN)

DNNs are artificial neural networks engineered by adding a combination of layers of neurons in different patterns of connection [20]. The layers are organized in a structure that introduces the possibility of having an input layer, and an output layer. The different layers take the raw data and extract relevant attributes. The neurons in the layers utilize mathematical functions on the raw data to be able to extract those relevant attributes. An example of one of these functions is combination (2.2).

$$Combination = \sum_{i=1}^{n}((Inputs_i \times Weights_i) + Bias) \tag{2.2}$$

After the processing on each layer, it has an activation method, which is needed to have non-linearity. There are different methods, some are SoftMax, Sigmoid, and Rectified Linear Units (ReLU) [21]. The sigmoid (2.3) function uses exponentiation, which is quite resource-heavy. In the equation of ReLU (2.4), activation goes in a different direction, opting for a less resource-consuming operation.

$$Sigmoid(x) = \frac{1}{1 + e^-x} \tag{2.3}$$

$$ReLU(x) = max(0, x) \tag{2.4}$$

After all layers have produced their output, we apply backpropagation to correct the errors. There exists different types of backpropagations, in here we show its basic procedure in figure 2.10 . Usually, we configure how much we want to depend on backpropagation and how much we want to keep the output by using different weights during training.

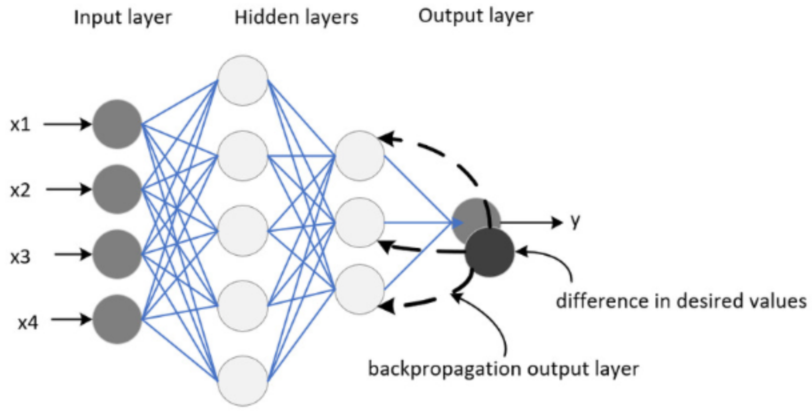


Figure 2.10: Basic backpropagation [22]

*Copyright CC-4.0*

# 3 Design overview

The objective of this chapter is to outline the design of the systems intended for evaluating the hardware accelerators. For the purpose of comparison across different platform, we implemented on there different setup: (1) the first system will use a Nvidia RTX-3060, which is a general purpose GPU; (2) the second system will use a Jetson Nano; and (3) the third system will use a AMD Xilinx Kria KV260 platform. We have chosen these platforms for three reasons: (1) The Kira KV260 and the Jetson Nano are both edge computing platforms that were specialised for acceleration, with the first being an FPGA and the second a GPU-based platform; (2) the difference between both is what motivates us to make the comparison; and furthermore (3), we chose the RTX 3060 as a comparison between a consumer platform against the other specialised platforms.

## 3.1 FPGA preparation

We use the unified software platform for FPGAs to assist with the development of the KV260. That unified software platform is called Vitis, and it has been developed by Xilinx.

This platform has multiple components:

1. *Vitis accelerated libraries*: Xilinx provides them to support the implementation, some of these libraries are Solver, DSP, Sparse, and Basic Linear Algebra Subroutines (BLAS)[23].

2. *Vitis Core Development Kit*: This kit includes tools required for compiling, analyz-

ing and debugging.

3. *Xilinx Runtime library*: The XRT allows communication between applications on host machine and accelerator.

4. *Vitis AI*: This tool 3.1 is used for AI inference development. This encompasses DPUs uniquely optimized for neural network computations and accelerating deep learning.

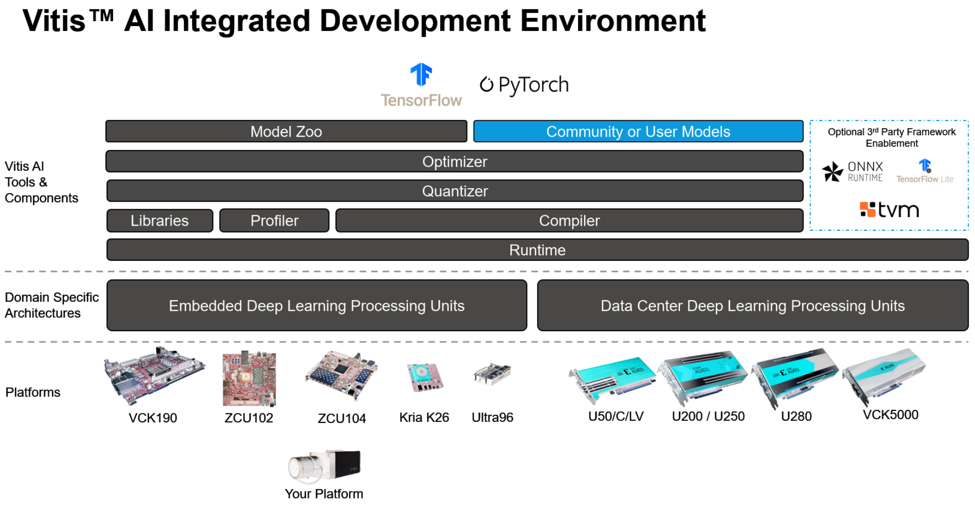

Vitis™ AI Integrated Development Environment
TensorFlow
PyTorch
Vitis AI Tools & Components
Model Zoo
Community or User Models
Optional 3rd Party Framework Enablement
ONNX RUNTIME
TensorFlow Lite
tvm
Optimizer
Quantizer
Libraries
Profiler
Compiler
Runtime
Domain Specific Architectures
Embedded Deep Learning Processing Units
Data Center Deep Learning Processing Units
Platforms
VCK190
ZCU102
ZCU104
Kria K26
Ultra96
U50/C/LV
U200 / U250
U280
VCK5000
Your Platform

Figure 3.1: Vitis AI [23]

*© 2024 Advanced Micro Devices, Inc.*

The Kria KV260 speciality is the possibility of generating a Deep Learning Processor Unit (DPU), shown in figure 3.2, as the FPGA instead of designing it from scratch using HDL. Inside the DPU we can find different elements similar to the elements that you could find inside an FPGA design, such as DPS, BRAM, UltraRAM, LUTs, and FFs. This speciality enables importing an AI engine from AMD sources and connecting it to the DPU design.

The platform also makes it available to have multiple DPU instances and transforms the size of the DPUs.

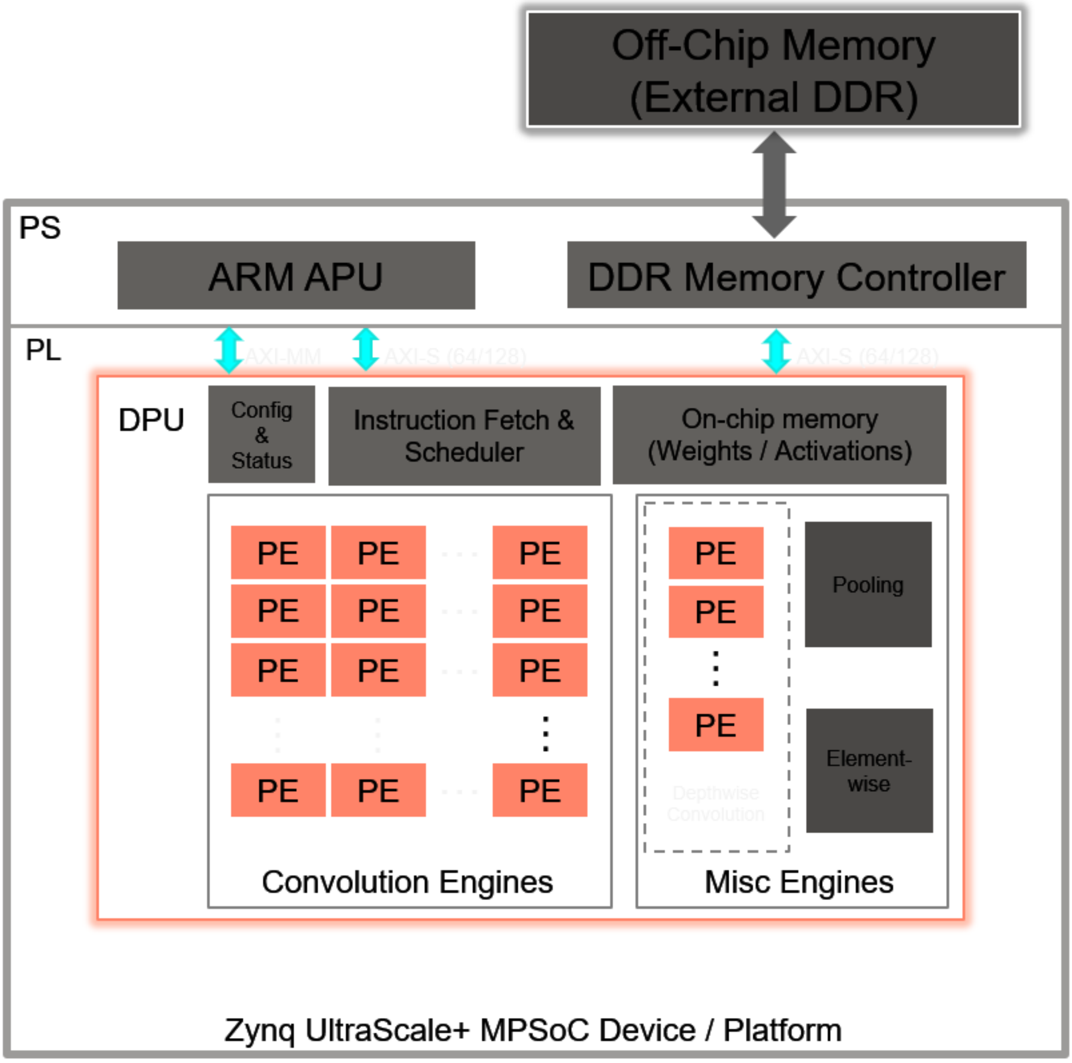


Figure 3.2: DPU for Zynq UltraScale+ [23]

*© 2024 Advanced Micro Devices, Inc.*

Both edge computing platforms are developed and optimized for implementing DNN models, however the RTX-3060 is optimized for game and video processing performance. The FPGA board, on the other hand, has extra optimization that we can implement due to their reconfigurability. To achieve the optimization the FPGA will have multiple processes used [23].The multiple steps that we followed are:

1- Pruning:

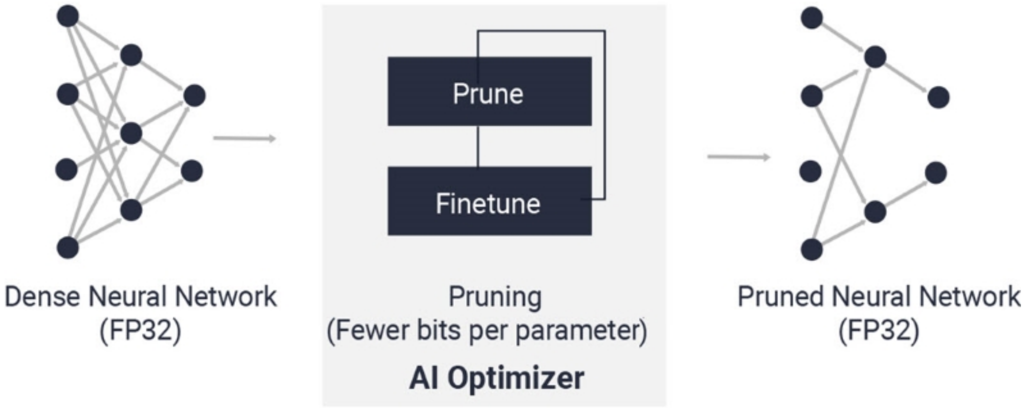


Figure 3.3: Pruning [23]

*© 2024 Advanced Micro Devices, Inc.*

The AMD's AI-Optimizer will obtain a neural network and use a pruning process [24], as depicted in Fig 3.3. Initially, the optimizer conducts a sensitivity analysis aimed at determining the extent to which each convolution kernel in every layer influences the overall output [25]. Next, the kernel weights that are going to be pruned, i.e., zeroed. This is made possible by evaluating the pre-trained model before removing any element. Then, We adjusted the weights through several training epochs to regain accuracy. It usually performs pruning in many iterations, for each, we can examine the state of the neural network, the loss in accuracy and the number of selected neurons to prune. Depending on the state, we can decide when to stop the pruning or roll back as needed [23].

After the last phase of the process, which erases the neurons elected for pruning, we have a model with a smaller number of neurons and, thus, less computation. For instance, a layer that previously necessitated computing 128 channels may now only need to compute 87 channels after pruning [25]. Figure 3.4 shows a graphical idea of the flow of pruning.

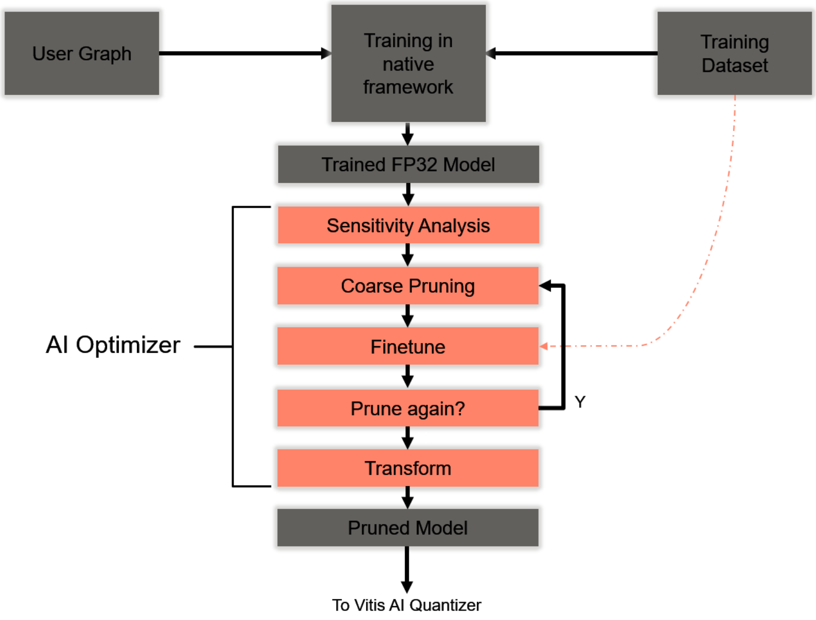


Figure 3.4: Pruning process flow [25]

*Copyright 2024 Advanced Micro Devices, Inc*

Figure 3.5 shows the accuracy and parameter reduction modification during the loop of pruning.

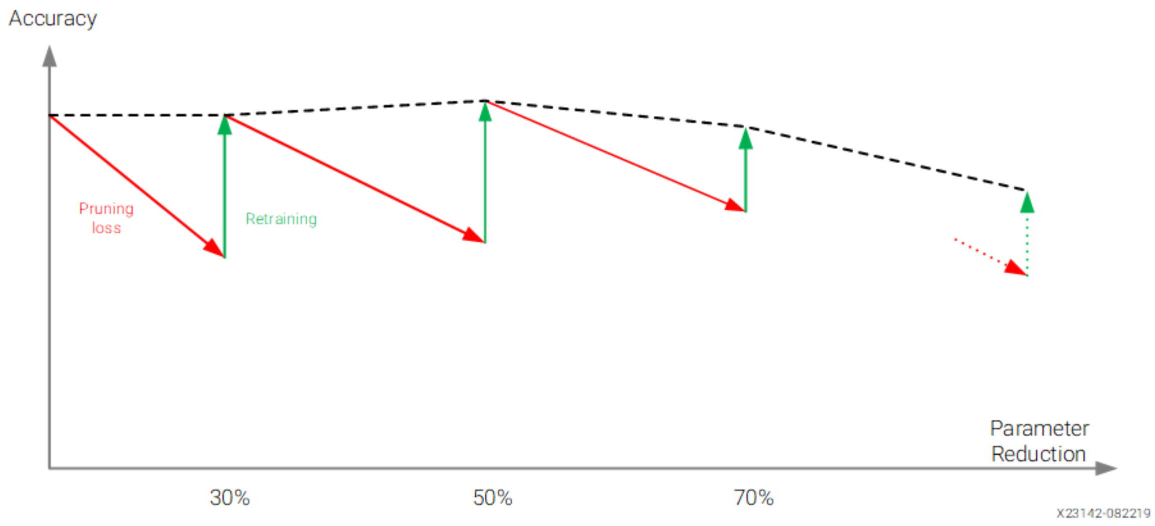


Figure 3.5: Evolution of accuracy and parameter reduction over iterations in the pruning process [23]

*© 2024 Advanced Micro Devices, Inc.*

2 - Quantization [26]:

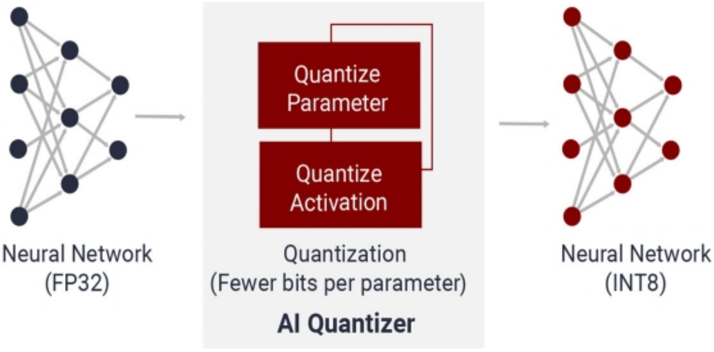


Figure 3.6: Quantization process [23]

*© 2024 Advanced Micro Devices, Inc.*

This method enhances deployment efficiency by employing integer quantization to minimize data path bandwidth, energy consumption, and memory usage [26].

In this case, we leverage INT8-quantization of the previously pruned network. The Vitis AI Quantizer first realizes a calibration step, where it takes a small part of the training samples, and then pushes them through the network to observe the activation of each channel. The weights and activations were then quantized as 8-bit integers. We usually call this step Post-Training Quantization [27].

In Fig 3.7 we can observe the flow of the quantization process.

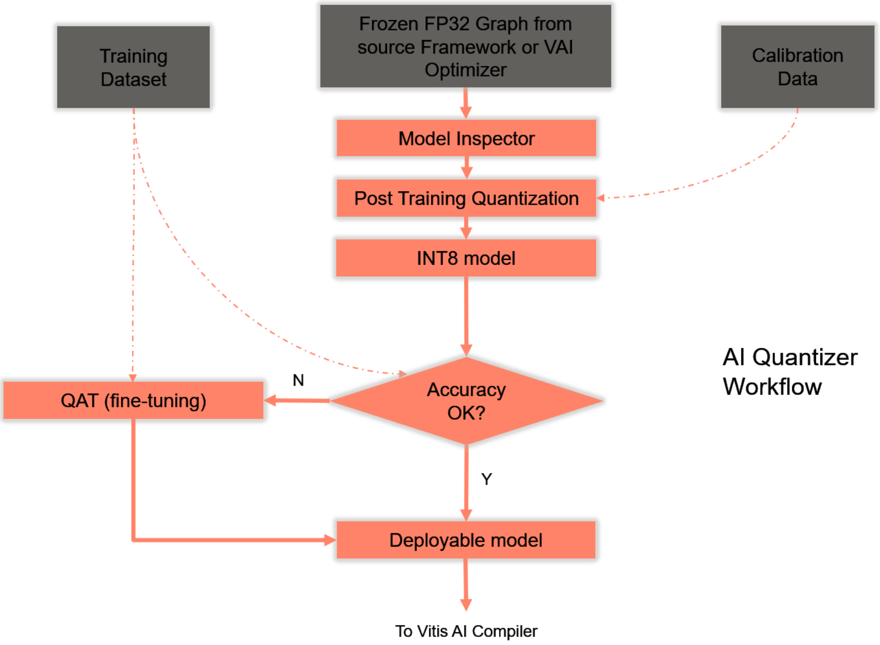


Figure 3.7: Quantization flow [23]

*© 2024 Advanced Micro Devices, Inc.*

3. Compilation:

After the quantization, the compilation process is needed. The compiler is employed to implement various optimizations; for example, batch normalization operations are fused with convolution when the convolution operator precedes the normalization operator. Given that the DPU accommodates multiple dimensions of parallelism, effective instruction scheduling is crucial for harnessing the innate parallelism and maximizing data reuse potential within the graph [25].

Figure 3.8 shows the compiler flow:

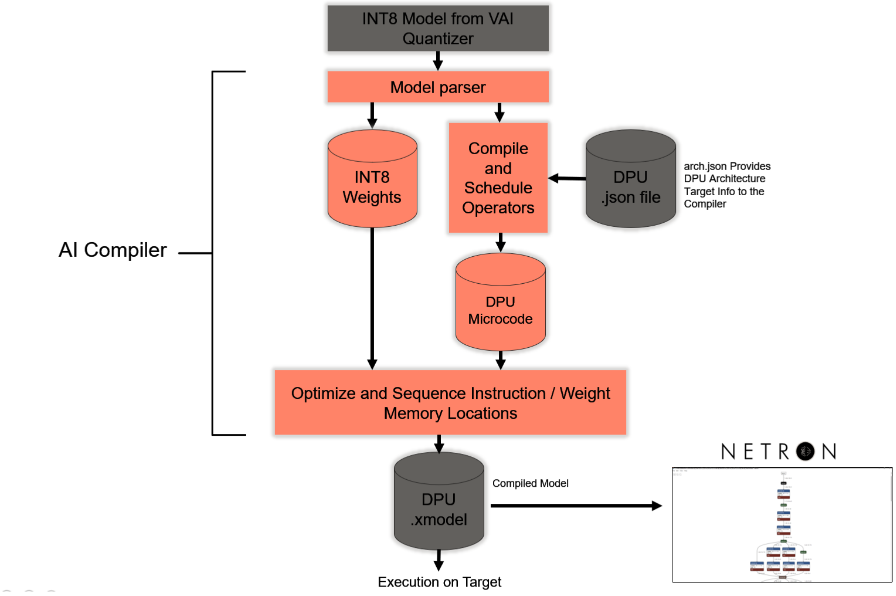

INT8 Model from VAI Quantizer
Model parser
INT8 Weights
Compile and Schedule Operators
DPU .json file
arch.json Provides DPU Architecture Target Info to the Compiler
AI Compiler
DPU Microcode
Optimize and Sequence Instruction / Weight Memory Locations
DPU .xmodel
Compiled Model
NETRON
Execution on Target

Figure 3.8: Compilation flow [23]

*© 2024 Advanced Micro Devices, Inc.*

The Jetson Nano is provided with an image prepared for running inference manifesting the ease of use of the platform. On the other hand, to be able to perform inference in the RTX 3060 the Nvidia toolkit is necessary, it prepares models and runs them in consumer graphic cards. Figure 3.9 we can see how the compilation process works for any .cu file that we are going to use in development.

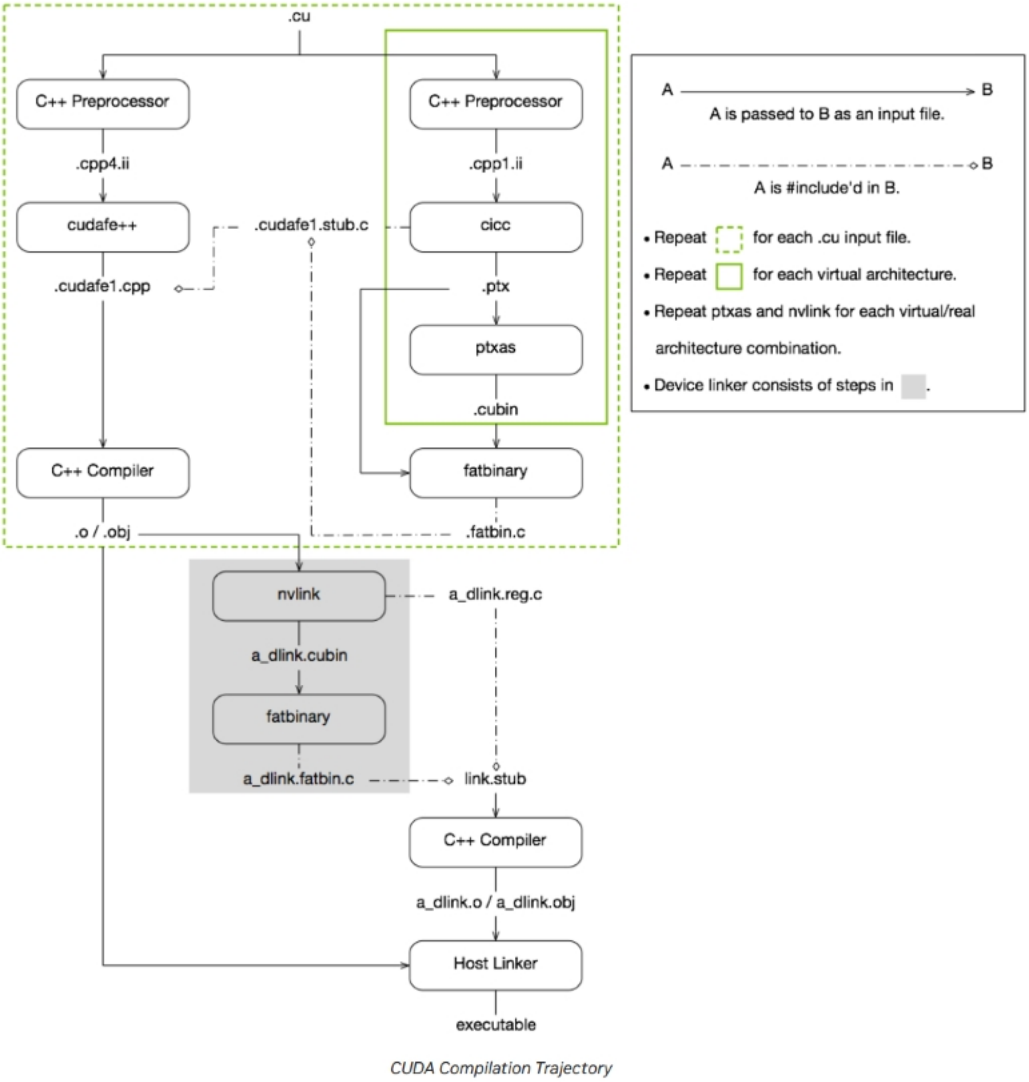


Figure 3.9: CUDA compilation flow

*© Copyright 2024, NVIDIA*

## 3.2 AI Models

We have discussed all preparations for running models. In this section, we discuss the models that were implemented to evaluate.

### 3.2.1 You only look once (YOLO)

Real-time object detection became a core piece on multiple applications, such as autonomous vehicles, robotics, video surveillance, and augmented reality.

There have been a lot of different object detection algorithms, the You Only Look Once (YOLO) model became popular for being a middle ground between performance and precision [28]. The model has had multiple iterations improving performance or adding new features as shown in Fig 3.10. The common dataset used for training this model is the Microsoft Common Objects in Context (COCO) [29].

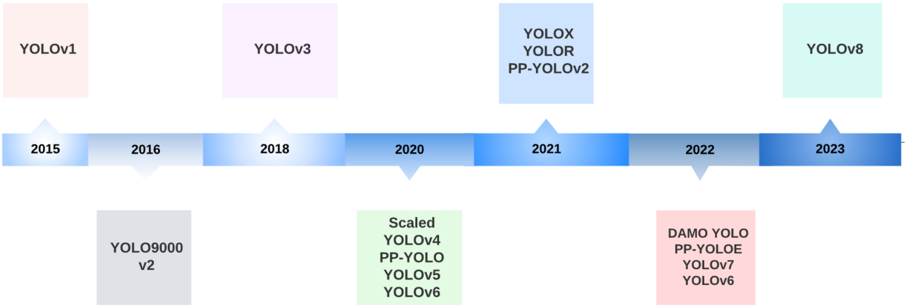


Figure 3.10: YOLO version history [30]

The architecture of YOLO has been improved for each iteration, we include the YOLOv3 architecture 3.11 as an example of the YOLO architecture.

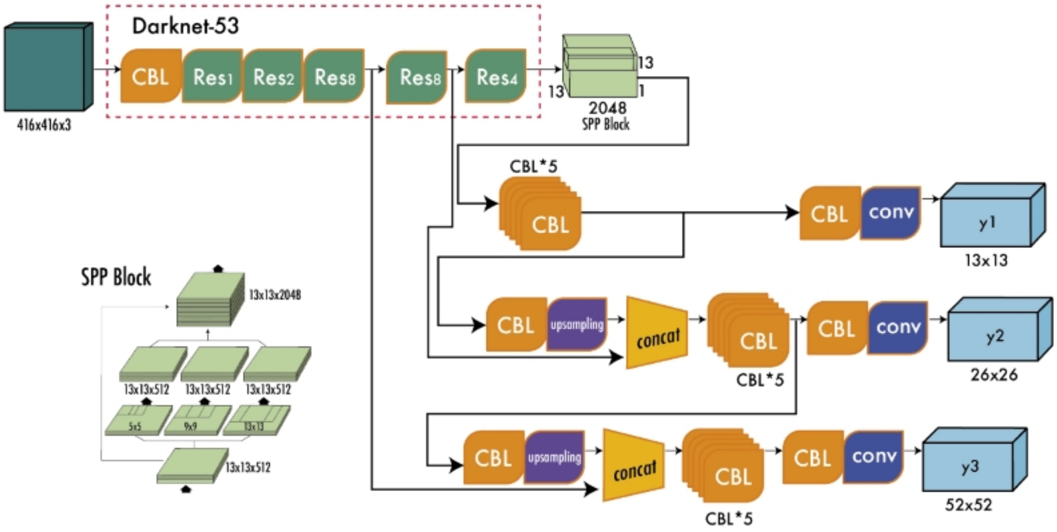


Figure 3.11: YOLOv3 architecture [30]

*© Copyright 2024, ACM*

### 3.2.2 ResNet

With the implementation of larger and larger deep learning models, previous research has found more errors in training and loss of accuracy, to prevent this, residual learning was created. As shown in Fig 3.12, residual learning improves inter-connectivity in the layers and prevents degradation on the deep layers of the model [31].

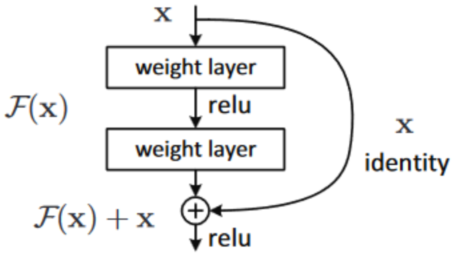

x
weight layer
relu
weight layer
$\mathcal{F}(\mathbf{x})$
x
identity
$\mathcal{F}(\mathbf{x}) + \mathbf{x}$
relu

Figure 3.12: Residual learning [32]

*Copyright © 2015 arXiv*

There are multiple models based on Resnet as shown in the table 3.1, the difference between them is the number of layers and the dimension of their filters, we have chosen to show the layer conv4_x as it is the one with the biggest changes between versions. We elected ResNet-50 as the model to use in the experiment due to it being the first to have a bit more complexity in its layers yet not a lot of layers in comparison with other versions such as ResNet-101 or bigger.

| Layer | ResNet-18 | ResNet-34 | ResNet-50 | ResNet-101 | ResNet-152 |
|---|---|---|---|---|---|
| conv4_x | $2 \times \begin{bmatrix} 3 \times 3, & 256 \\ 3 \times 3, & 256 \end{bmatrix}$ | $6 \times \begin{bmatrix} 3 \times 3, & 256 \\ 3 \times 3, & 256 \end{bmatrix}$ | $6 \times \begin{bmatrix} 1 \times 1, & 256 \\ 3 \times 3, & 256 \\ 1 \times 1, & 1024 \end{bmatrix}$ | $23 \times \begin{bmatrix} 1 \times 1, & 256 \\ 3 \times 3, & 256 \\ 1 \times 1, & 1024 \end{bmatrix}$ | $36 \times \begin{bmatrix} 1 \times 1, & 256 \\ 3 \times 3, & 256 \\ 1 \times 1, & 1024 \end{bmatrix}$ |

Table 3.1: ResNet versions [33]

*Copyright CC-4.0*

# 4 Implementation and Platform Characteristics

This system aims to identify the objects in front of the camera. We have used the COCO dataset as explained before for training and calibration, steps previously presented in the last chapter. This dataset contains 123.287 images which are already annotated and with all the information prepared for training [34]. Moreover, this dataset is used plenty of times in research endeavours. Subsequently, the models were trained using 100,000 images, while the remaining 23,287 were allocated for the calibration process conducted during the quantization procedure.

In this experimental study, two deep neural network models, namely ResNet-50 and YOLO, were utilized. These selections were made with the intention of encompassing a spectrum of performance attributes and architectural complexities within the scope of the investigation. Each system was subjected to testing using all models within a consistent environment in three runs of 60 seconds, with longer runs of 20 minutes if anomalies occurred.

The experimental procedures involved the utilization of three different hardware accelerators, a single-board GPU-based Jetson Nano from NVIDIA, a Kria SOM KV260 FPGA-based from AMD and a mobile RTX-3060. All accelerators were employed both as edge computing devices and to run the OS necessary in each case. The reason we used them as stand-alone devices arises from the need to properly evaluate the power consumption and temperature of the devices without interference from other systems.

The only sensor deployed in this work was a webcam with a 1920x1080 resolution implemented as an RGB camera. This camera was connected to each platform through a USB port, the platform had to process the camera feed, realize the inference and display the results in the display. This display was connected via HDMI to the platforms. All platforms were AC-powered, all using barrel connectors, each with its specific voltage as shown in table 4.1.

| Platform | Voltage |
|---|---|
| Jetson Nano | 5V |
| Kria KV260 | 12V |
| RTX 3060 | 12V |

Table 4.1: Voltage of platforms

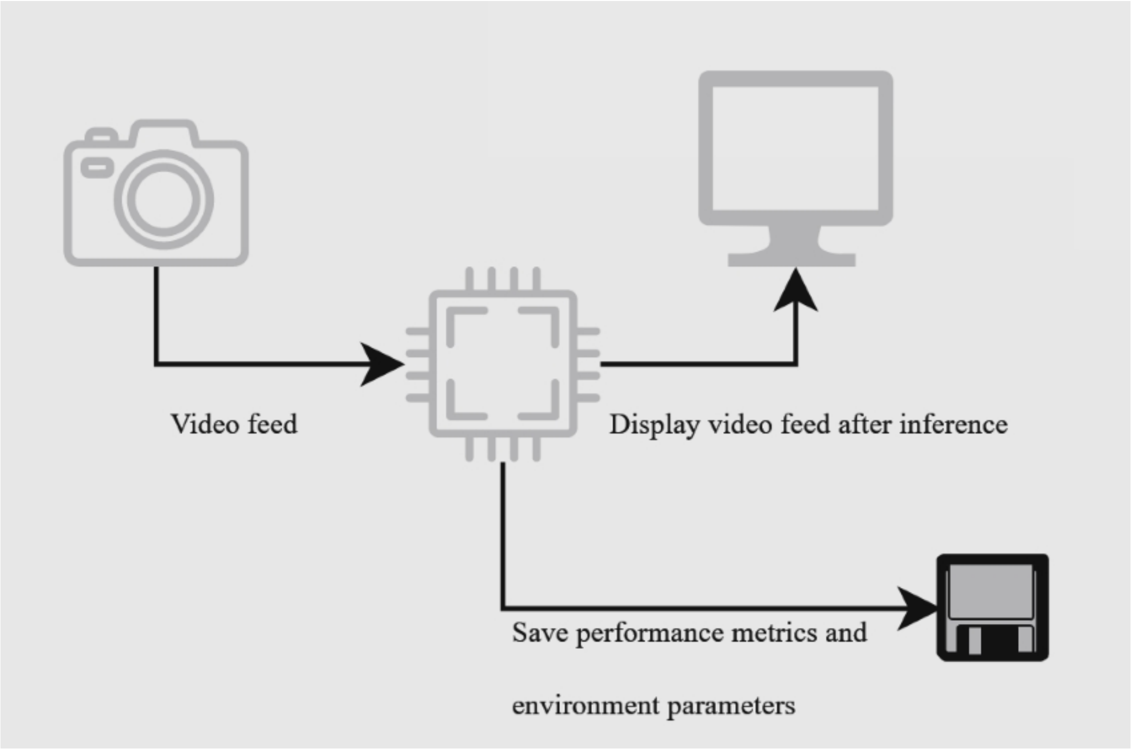

Video feed
Display video feed after inference
Save performance metrics and
environment parameters

Figure 4.1: System workflow

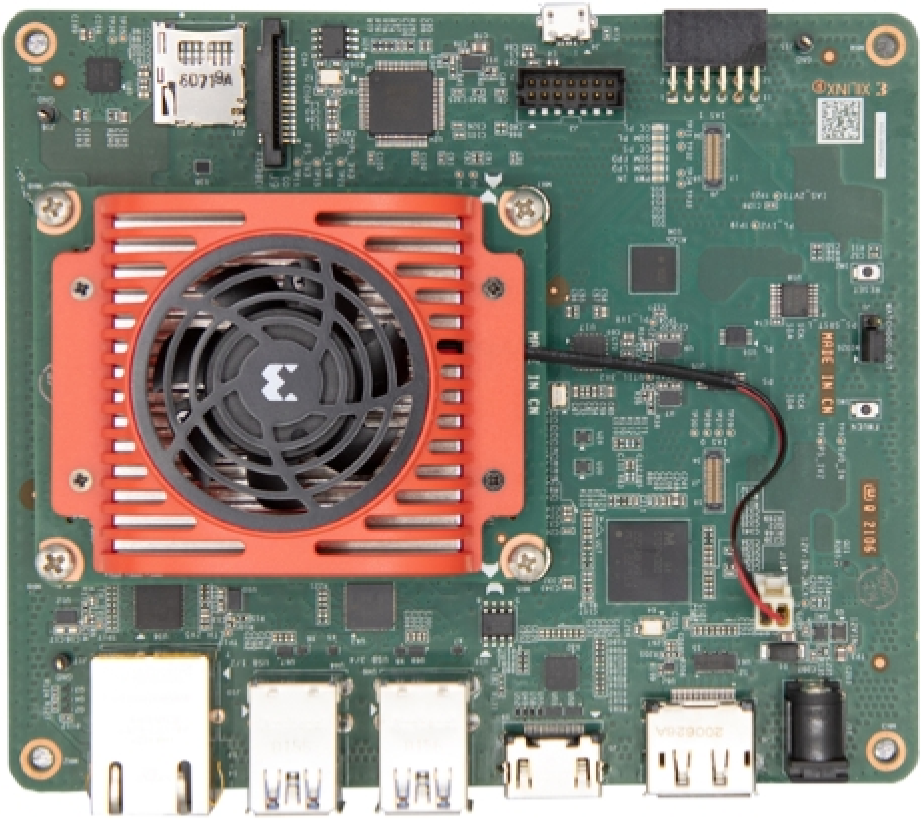


Figure 4.2: KV260 as edge computing

*© 2024 Advanced Micro Devices, Inc.*

The operating system installed on each of the platforms can be seen in the table 4.2.

| Platform | OS |
|---|---|
| Jetson Nano | Ubuntu 20.04 |
| Kria KV260 | Petalinux |
| RTX 3060 | Ubuntu 22.04 |

Table 4.2: Platform OS

Throughout the training phase, models previously trained on the extensive Microsoft COCO dataset were employed [29]. Therefore, for the Kria KV260 we needed to perform quantization employing Vitis AI. Essentially, within the Vitis AI environment, three distinct frameworks exist for deploying quantization: ONNX, TensorFlow, and PyTorch. PyTorch was our selection due to the vast support and documentation for the framework. Upon successfully deploying the PyTorch framework in the environment, we performed the quantization process explained in the last chapter. This resulting model was compiled within the corresponding environment for the DPU of KV260, whose name is DPUCZDX8G due to naming convention. A more intrinsical illustration of the inference process for Jetson Nano and KV260 is explained in Algorithms 1 and 2, the RTX 3060 follows the same process as the Jetson Nano.

**Algorithm 1** Jetson Nano inference

```
$model \leftarrow trained\_model$
$device \leftarrow cuda$
Load model to device.
Initialize pipeline.
Get camera input as source.
Connect to the device and start the pipeline.
while True do
    Convert the RGB frame to a PyTorch tensor.
    Load the tensor to the GPU.
    Perform inference using the model.
    Calculate FPS.
    if model == YOLO then
        Identify object and draw a box.
    else
        if model == ResNet-50 then
            Identify object.
        end if
    end if
end while
```

**Algorithm 2** Kria KV260 inference

```
Load Xmodel.
Get camera input as batch of images.
while True do
    Perform inference using the model.
    Calculate FPS.
    if model == YOLO then
        Identify object and draw a box.
    else
        if model == ResNet-50 then
            Identify object.
        end if
    end if
end while
```

To acquire experiment measurements such as temperature, power, and memory usage, we employed tools provided by the platforms. In the case of both Jetson Nano and Kria KV260, they contain specific ICs to measure these values and provide their API to allow the user to access them through the preinstalled tools directly. However, in the case of the RTX-3060, we used the NVIDIA docker which provides all the specific values that a GPU reports.

# 5 Experimental Results

The outcomes depicted in this chapter were acquired through executing the inference of the two DNN models on all platforms Jetson Nano, Kria KV260 and RTX 3060 using the previously described system implementation. The models chosen for this experience were Resnet-50 and YOLO and all the video sources have been inferred in 1920x1080 resolution. Fundamentally, we derived our assessments based on four metrics, namely throughput, memory usage, delta of power and temperature.

Figures 5.1 and 5.2 illustrate the frames per second (FPS) of all the platforms during inference. While the ResNet-50 outperformed YOLO when running on the Jetson Nano with a maximum of 13 FPS, we consider the platform only suitable for running real-time inference for fixed robots. Subsequently, ResNet-50 also outperformed YOLO when running on the KV260 with a maximum of 33 FPS, we consider the platform suitable for running real-time inference in fast-moving robots and slow or fixed. Last, the YOLO outperformed ResNet-50 when running on the RTX 3060 with a maximum of 240 FPS, we consider the platform suitable for running real-time inference in aerial robots, fast-moving robots, and slow or fixed. The notable performance fluctuations that we can observe among the accelerators with these DNN models may be attributed to variations in several factors, including the number of convolutional layers, the complexity of the activation functions employed, memory usage, and parameter magnitude. Also, we believe that another factor of the overall lacklustre performance of all platforms is the big size of the images, as the video feed was 1920x1080.

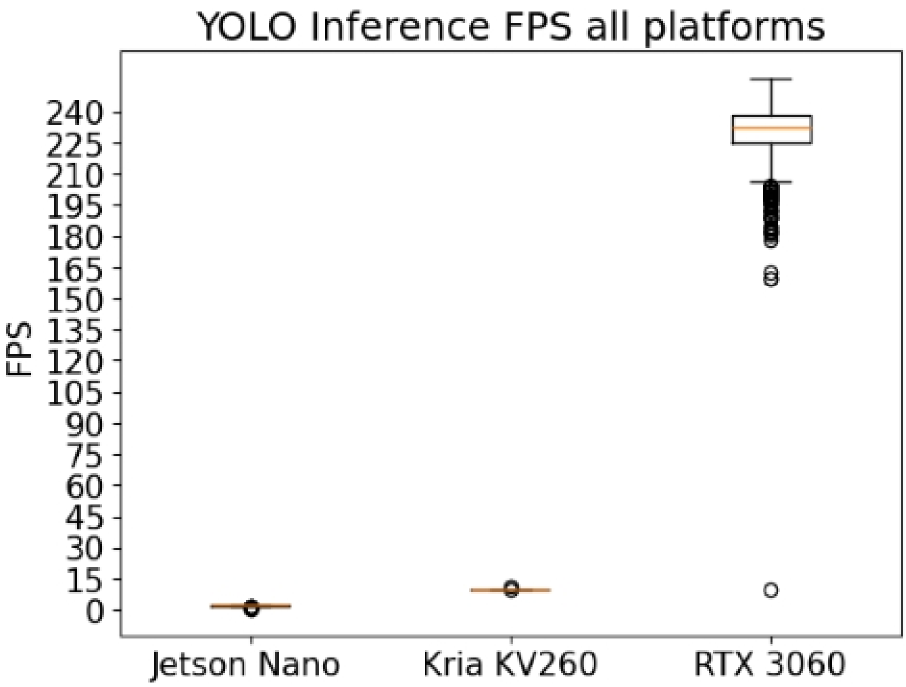


Figure 5.1: YOLO inference all platforms

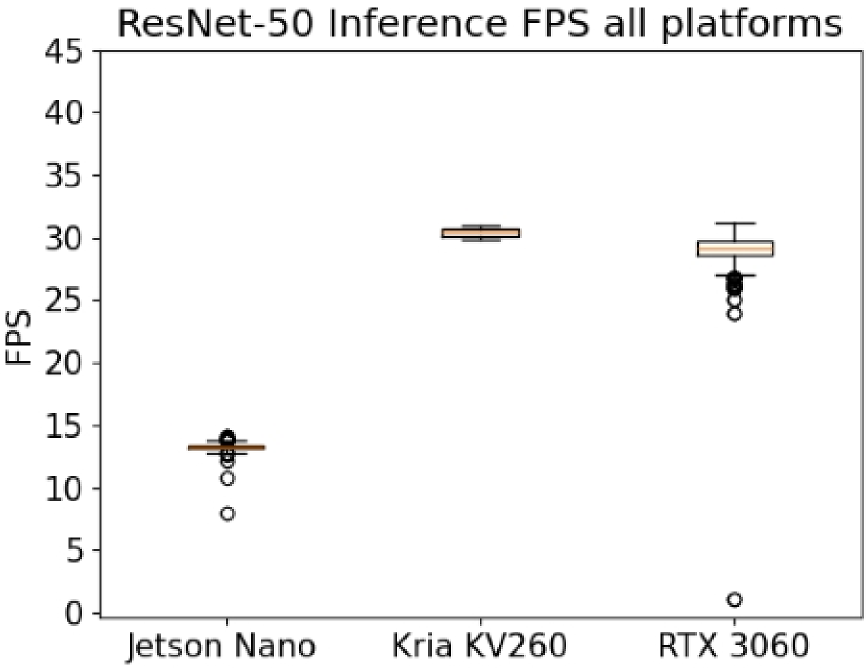


Figure 5.2: Resnet-50 inference all platforms

The memory usage in MB of all the platforms while running the DNN models is shown in the figure 5.3 and in the next table 5.1, we can see the memory usage in comparison with the total memory available in each platform.

The Jetson Nano is an outlier, as it utilizes too much memory, and requires the swap memory to be able to continue working during the inference. Furthermore, we believe this to be another reason for the low inference throughput of the platform, as accessing the swap memory incurs more latency due to the swap space being on a disk which is a lot slower than RAM. Moreover, the table shows us how the quantification process followed in the implementation reduced the memory usage of the KV260, which was only 14,91%. The number of convolutional layers and magnitude of the parameters have contributed to the usage of more memory in the Jetson Nano since the Jetson Nano uses floating-point parameters.

| Platform | Total Memory/Used Memory (MB) | % usage | Need for swap memory |
|---|---|---|---|
| Jetson Nano | 3956/3200 | 80,88 | YES |
| Kria KV260 | 4023/600 | 14,91 | NO |
| RTX 3060 | 6144/480 | 7,81 | NO |

Table 5.1: Total memory usage

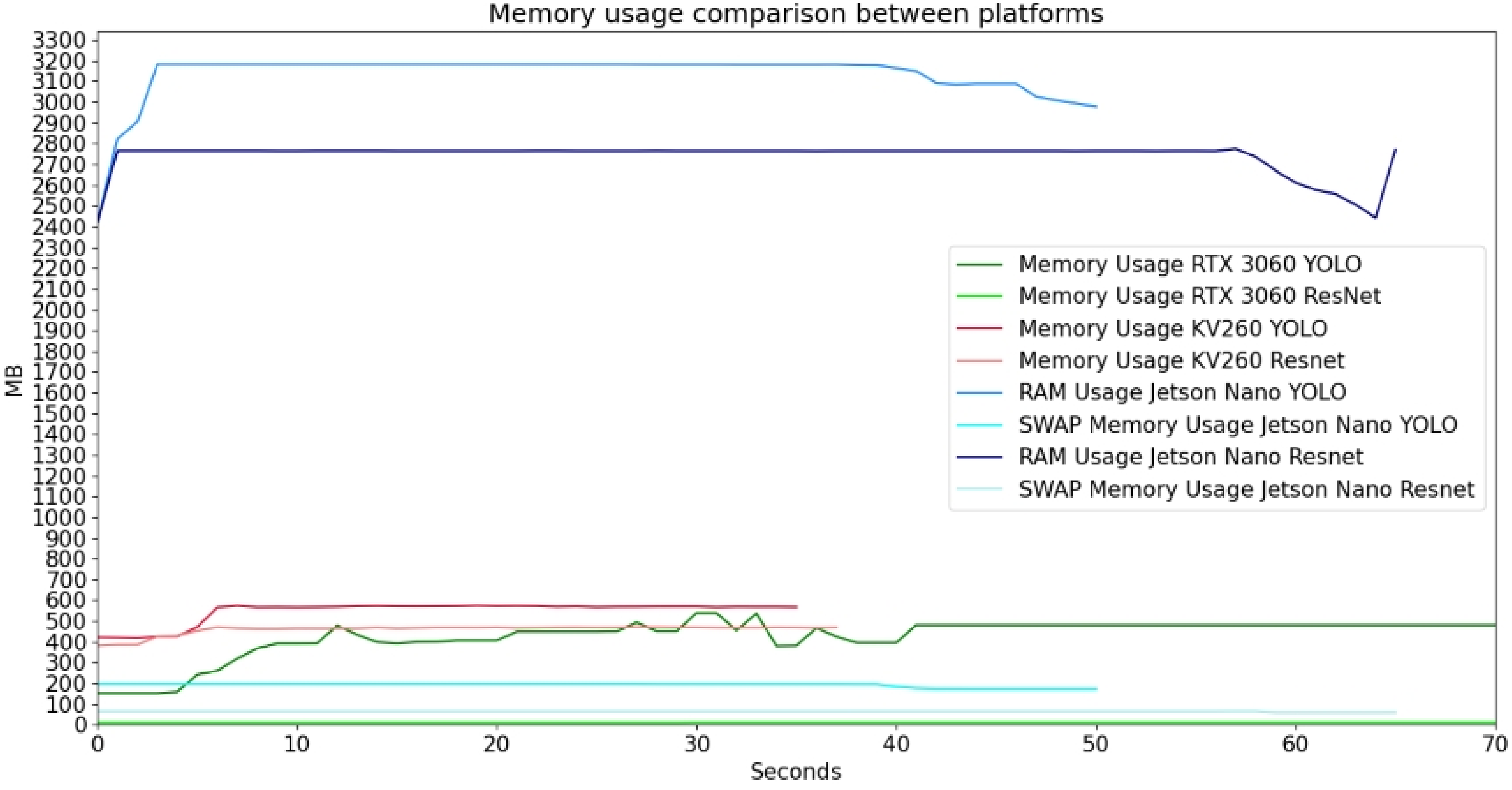


Figure 5.3: Memory Usage Comparison

Evidently, Kria KV260 has outperformed the Jetson Nano in previous metrics. However, before concluding which platform is most suitable for real-time edge computing applications, three additional metrics need evaluation, namely, power consumption, temperature of operation and performance per cost.

| Platform | Idle (W) | Running inference (W) |
|---|---|---|
| Jetson Nano | 1,90 | 9 |
| Kria KV260 | 5,50 | 10,70 |
| RTX 3060 | 10 | 45 |

Table 5.2: Power consumption of the platforms

While the power consumption of the Kria KV260 is higher than the Jetson Nano, it attained a better efficiency and stability based on the $\Delta W$ [35], [36]:

$$\Delta W = \sum_{0}^{n} \frac{W}{idle} \tag{5.1}$$

Specifically, $\Delta W$ is the total power consumption (W) of the calculations with the model parameters divided by the power consumption while the platform is in idle mode. For a platform to achieve efficiency and stability, $\Delta W$ should be between 1 and 2, with 2 being the maximum allowed to consider the platform efficient, more than 2 will indicate an issue with efficiency. Moreover, the system efficiency increases the closer that $\Delta W$ is to 1 since the power consumption of the system running the model is relatively equal to when it is idling. Furthermore, the Kria KV260 obtained, overall, a better performance per watt than the Jetson Nano, as illustrated in figure 5.5. This means that together with $\Delta W$ we can conclude that the Kria KV260 is the most efficient platform in the experiment, as the performance per watt measurement supports the efficiency comparison.

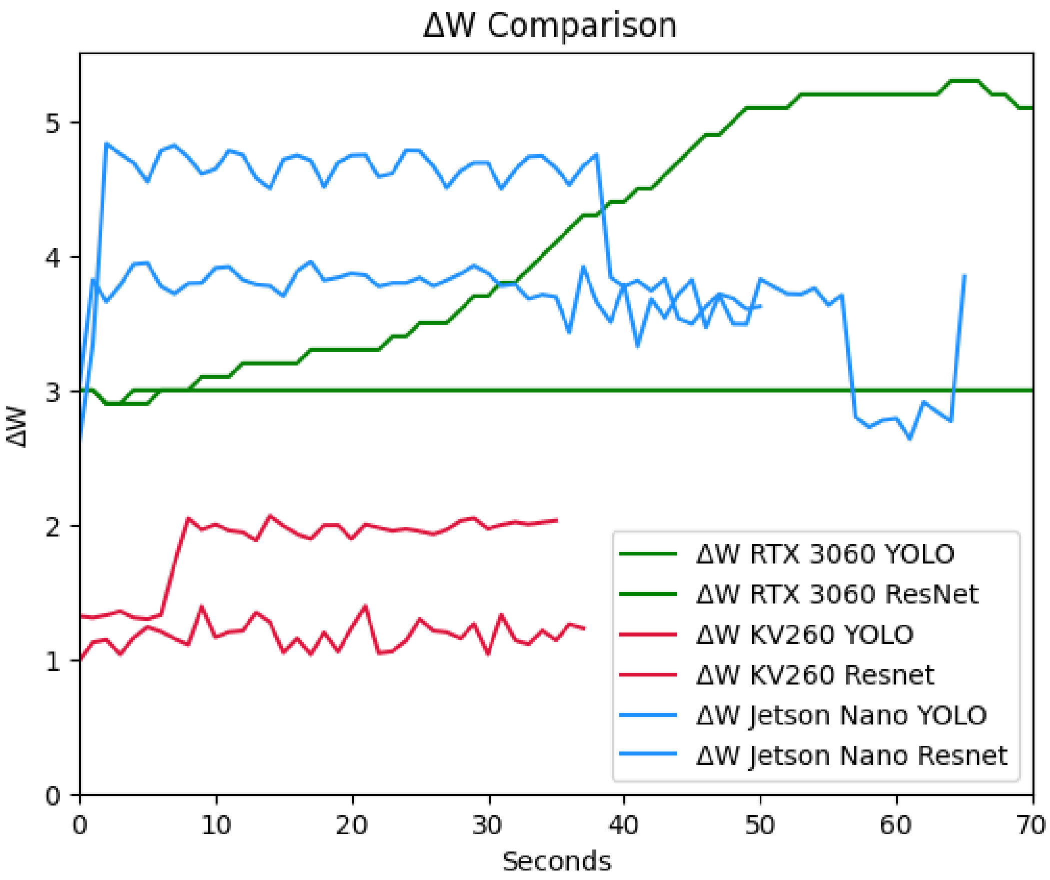


Figure 5.4: Delta W comparison

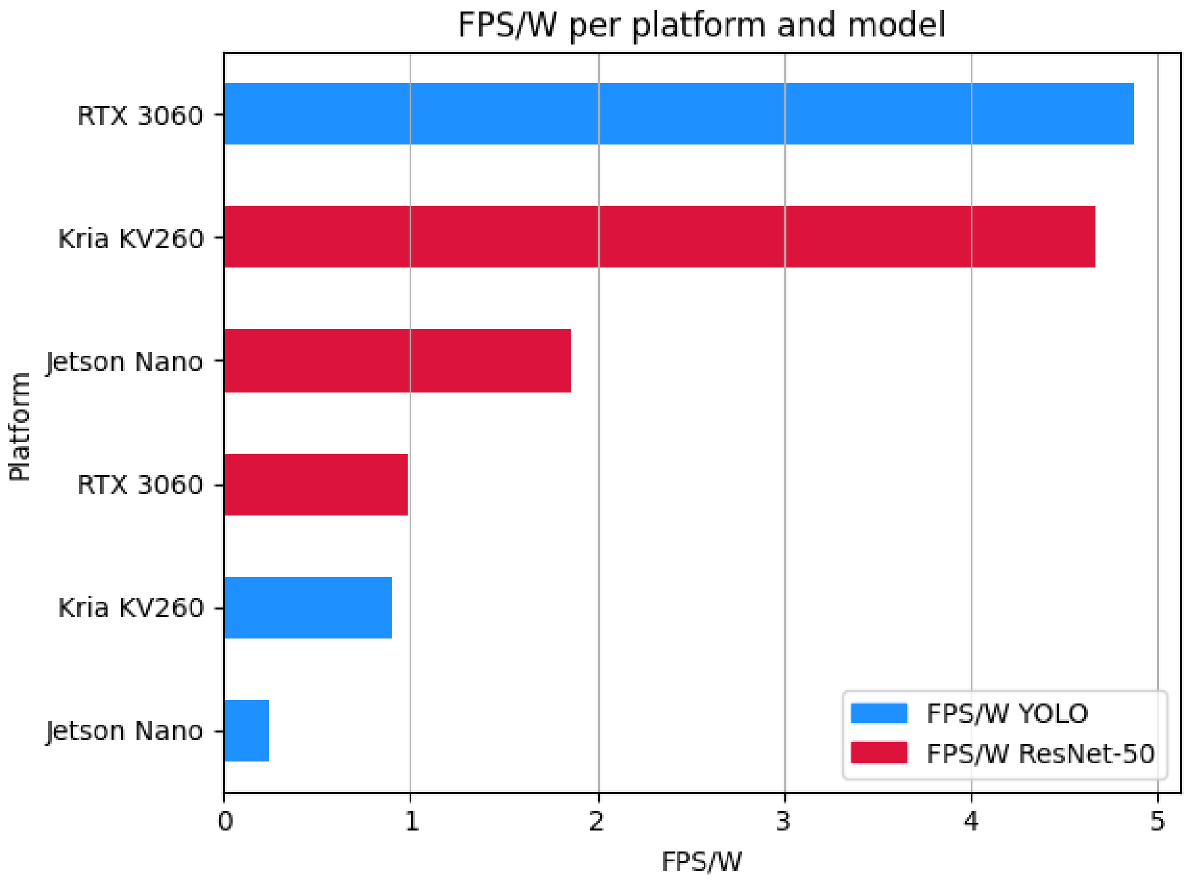


Figure 5.5: FPS per W

Temperature readings serve as a probing point into the platform's operation. The KV260 outperforms the rest, being the coolest as shown in the figure 5.6. We also found that the Jetson Nano shows a continuous increase in temperature during the run, which is an anomaly; this anomaly made us consider running a longer experiment to evaluate the temperature in detail as we expected the platform to reach a thermal throttling state, but the results of the long runs with the Jetson Nano showed the package temperature rose to a dangerous 109ºC, and the GPU to 93ºC, both of these temperatures are dangerous for the silicon and indicates a missing thermal throttling feature. After the readings, we can assure that the Jetson nano requires better optimization or the usage of smaller computational models to run at its operation temperature range.

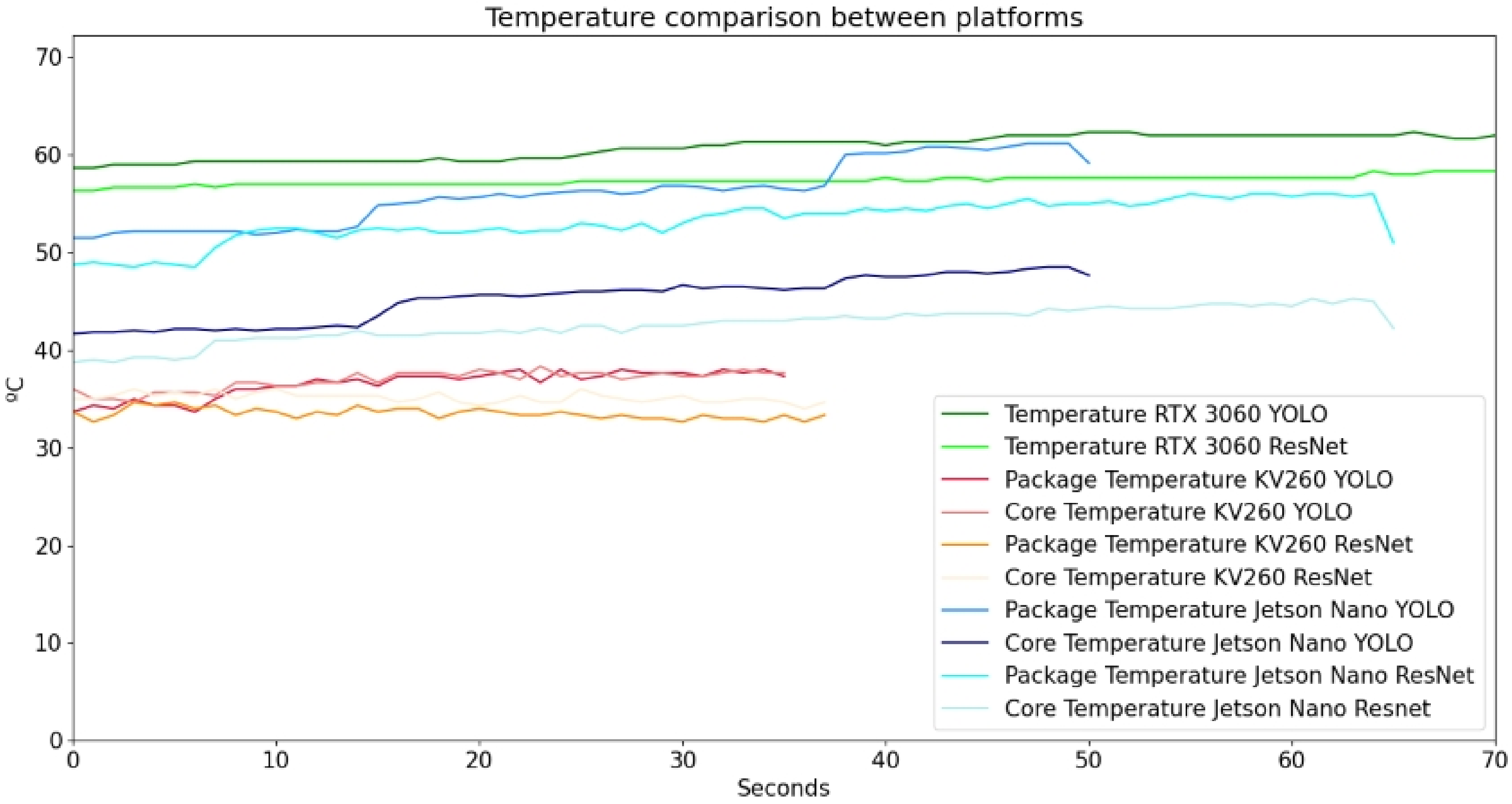


Figure 5.6: Temperature comparison

The FPS per euro formula is a heavily used metric in evaluating the choice of the accelerated hardware, especially for enthusiasts. As formulated in (5.2), this formula can also measure the value per euro. In specific, this measurement provides us with an overview of the performance in relation to the total cost. Moreover, this is another aspect that research teams have to evaluate when choosing which platform to acquire for development. Again, the Kria KV260 has the overall best performance per cost, with an outlier in the RTX 3060 running the YOLO model, as shown in figure 5.7. This also shows that the FPGA-based platform is the best value for money spent. Moreover, if we have taken into account the other metrics mentioned previously, the Kria KV260 platform is considered the most appropriate for real-time edge computing applications.

$$\frac{FPS}{€} = \frac{FPS}{TotalCost} \tag{5.2}$$

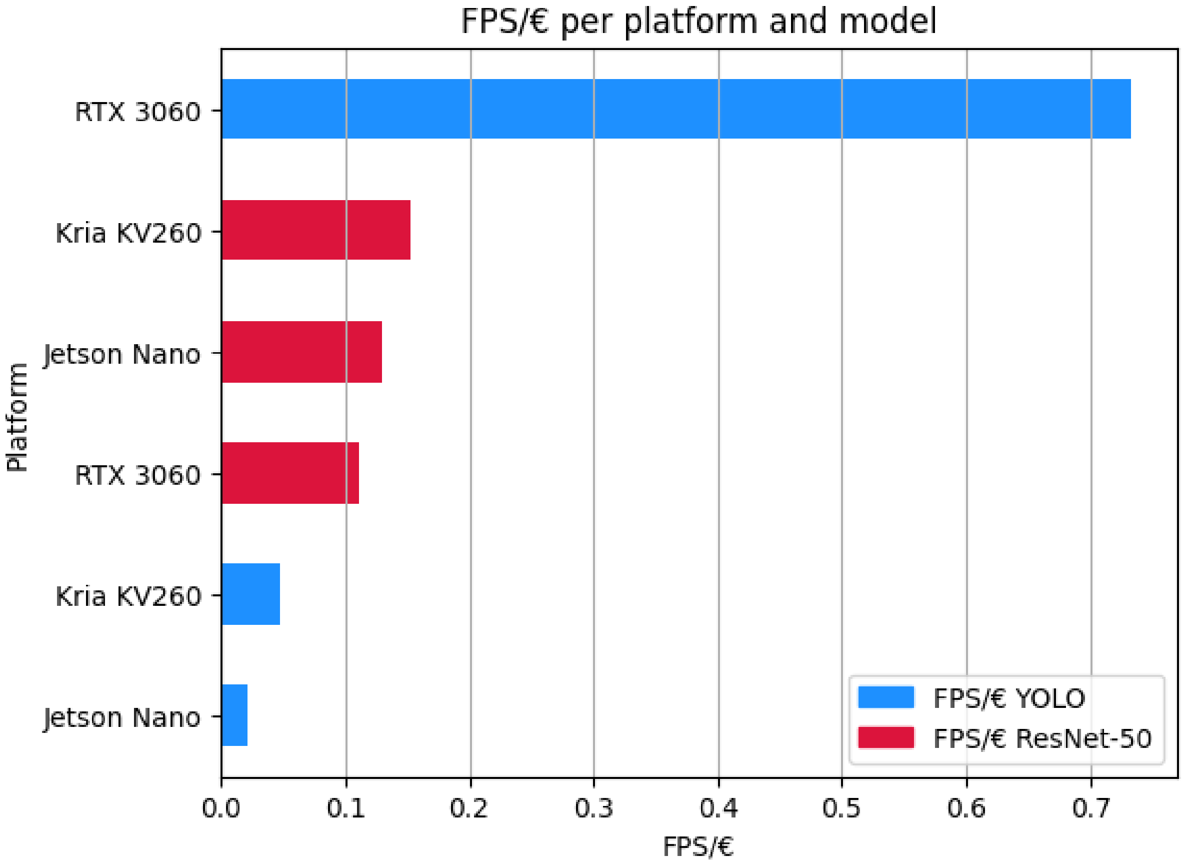


Figure 5.7: FPS per euro

# 6 Conclusion

## 6.1 Discussion

Now that we have collected all the graphs and experiment results, we can answer the questions we had at the beginning of the development of this thesis.

1. What type of hardware accelerator to choose in a new development?
2. Can FPGAs perform better than GPUs in running inference of YOLO and ResNet-50 for object detection, based on our criteria of evaluation?
3. How efficient are the three platforms(Jetson Nano, Kria KV260, RTX 3060) in running inference YOLO and ResNet-50 for object detection, can the difference in efficiency modify the election between them?

The type of hardware accelerator to choose for development depends on five factors, namely, performance (measured by FPS), power efficiency, time of development, unit cost and the size of the device. We are going to explain our decision for each factor. Firstly, if the development requires the best performance and there is no efficiency and cost limit, the best choice would have been RTX 3060, as it performs the best in each task. Secondly, if energy efficiency is the most essential target, then the best choice would be the Kria KV260 as it performs the best in terms of FPS/W. Thirdly, if the development cycle is limited, based on figure 6.1, the RTX-3060 is the most suitable, since it took one day to deploy the system, whereas, with Kria KV260, it took 25 days to have the system ready. However, if the aim of the design is to minimize the unit cost, then RTX-3060 would not be a suitable option. Lastly, considering that the RTX 3060 platform is a lot bigger than the other two platforms, if the available size is smaller than a laptop, then then decision is between the other two platforms, in that case, the lowest development time was using the Jetson Nano.

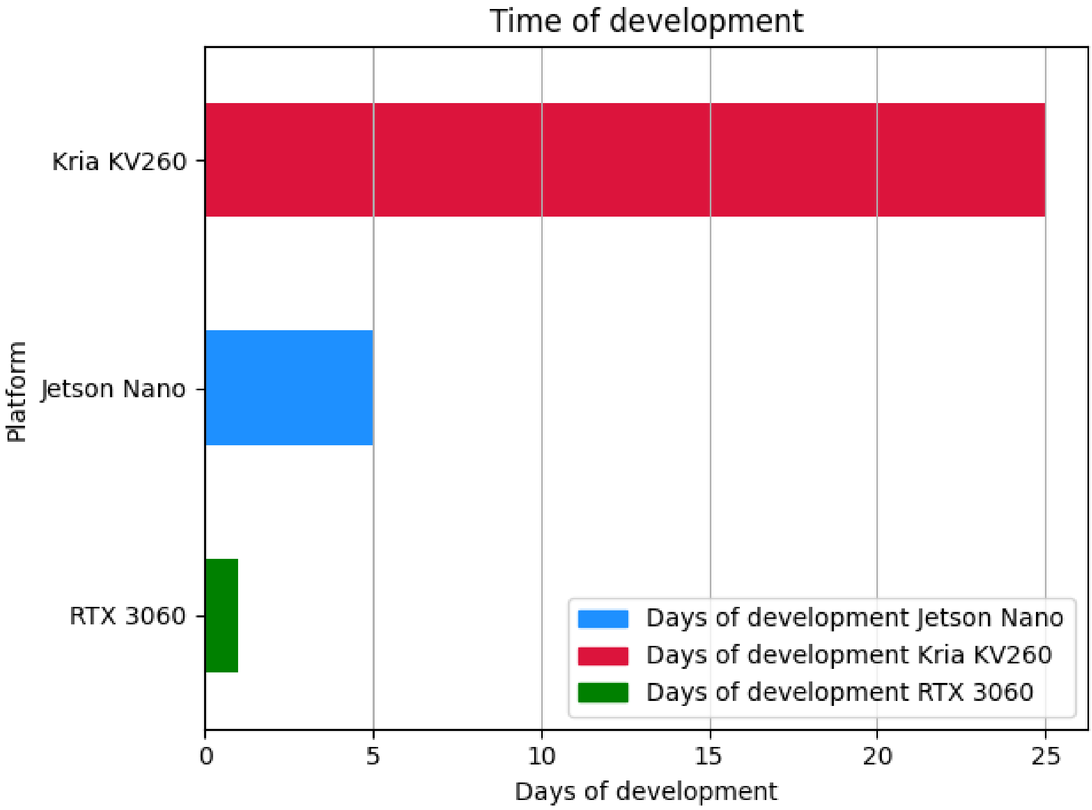


Figure 6.1: Time of development

Figure 6.2 shows a summary of the five factors of each platform mentioned previously, the closer to the centre of the pentagon, the worse that characteristic is, for example, the RTX 3060 is the most expensive platform out of the three, this makes its graph closer to the centre in the cost axis.

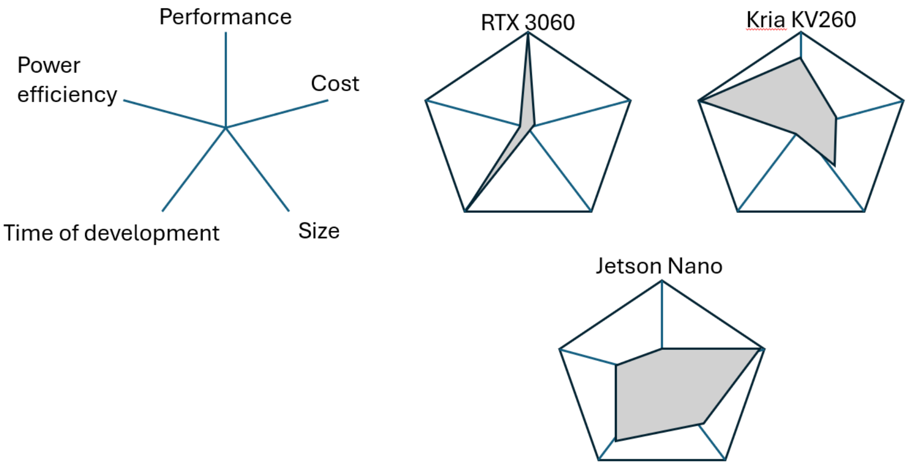


Figure 6.2: Pentagram comparison between the three platforms

Regarding the second research question: Can FPGAs perform better than GPUs in running inference of YOLO and ResNet-50 for object detection? From the frame-rate per wattage comparison in figure 5.5, we can see that the Kria KV260 has performed comparably to the RTX3060 and better than the Jetson Nano. Moreover, in terms of frame-rate per euro comparison in figure 5.7, we can also notice that the Kria KV260 performs better than the Jetson Nano and trades blows against the RTX3060. The last research question brings us to the power efficiency of each platform and how it can modify the election between them, we believe that the requirements for each environment and experiment will modify which platform to choose. If the overall energy budget is low, choosing a more energy-efficient platform will provide more headroom to power other hardware needed or keep the experiment running for longer on a battery pack.

Reflecting from the development of the object-detection model, we would like to address the following topics: Firstly, there is a lack of documentation and lack of maintenance of the documentation of the Kria KV260. Thus, this has been the main challenge that prolonged our development time with the KV260. Secondly, there was a lack of documentation on the object detection topic. We had to build our own specific solution to address the research questions that have not been documented before, as compared to the better-maintained platforms like the Jetson Nano or the RTX 3060. Evidently, if the team that is going to work with the Kria KV260 is not an expert in this platform already, then they will spend a long time to develop the system. Hence, this could be a impossible for research teams that require short development cycle.

Concerning the topic of thermal dissipation and power consumption, the Jetson Nano, as shown in figures 5.2 and 5.6, can reach temperatures higher than 100ºC and, moreover, it can continue to increase the power drawn. This indicates a lack of a thermal throttling feature, and hence, can potentially cause major damage as it does not include active cooling either. Nonetheless, NVidia guarantees that it should be safe enough to run on passive cooling without a fan.

## 6.2 Future works

The thesis has evaluated the performance and energy efficiency of the platforms Jetson Nano, Kria KV260 and RTX 3060 running inference of the models YOLO and ResNet-50. To have an even deeper insight on the different characteristics of the evaluated computing platforms, the following steps should be performed:

First, the design of a new thermal-protective test suit, with its own temperature and power readings based on different sensors and cameras, which will help diagnosing problems like overheating on the Jetson Nano. This test suit could include its own active cooling system, with multiple fans or heat-dissipating components.

Second, the use of cameras would influence the ways of acquiring images for the inference. In the context of this thesis, we implemented the system with a 1920x1080 RGB-camera, which can provide the full details for the feature extraction during the training of the DNN model. We could have alternatively used a camera with a different data transfer method, such as using different type of communication bus, or camera sensors that provide allow adjustable resolution and quality, or cameras with built-in image pre-processor.

Thirdly, by adding more edge computing platforms to the evaluation, we could have used two edge computing platforms and a normal user platform, but other platforms in the research field are being used. Additionally, we could add more platforms to evaluate using the previous techniques such as Google Coral [37] and Intel Movidius [38] since they could provide a wider view of the current edge computing environment and which platform to choose for each type of development.

# References

[1] M. Satyanarayanan, P. Bahl, R. Cáceres, and N. Davies, “The case for vm-based cloudlets in mobile computing”, *IEEE Pervasive Computing*, vol. 8, pp. 14–23, 4 Oct. 2009, ISSN: 15361268. DOI: 10.1109/MPRV.2009.82.

[2] R. Hussain and S. Zeadally, “Autonomous cars: Research results, issues, and future challenges”, *IEEE Communications Surveys and Tutorials*, vol. 21, pp. 1275–1313, 2 Apr. 2019, ISSN: 1553877X. DOI: 10.1109/COMST.2018.2869360.

[3] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, “Edge computing: Vision and challenges”, *IEEE Internet of Things Journal*, vol. 3, pp. 637–646, 5 Oct. 2016, ISSN: 23274662. DOI: 10.1109/JIOT.2016.2579198.

[4] A. A. Samuel, A. A. Adeleke, E. N. Anosike-Francis, *et al.*, “Advent of artificial intelligence in automotive engineering”, *2023 2nd International Conference on Multidisciplinary Engineering and Applied Science, ICMEAS 2023*, 2023. DOI: 10.1109/ICMEAS58693.2023.10429907.

[5] Y. Al.Amery, “Fpga-based hardware accelerators for deep learning in mobile robotics”, *UTU pub*, 2023, [visited:07.05.2024]. [Online]. Available: https://www.utupub.fi/handle/10024/176115.

[6] K. Karras, E. Pallis, G. Mastorakis, *et al.*, “A hardware acceleration platform for ai-based inference at the edge”, *Circuits Syst Signal Process 39, 1059–1070 (2020)*, [Online]. Available: https://doi.org/10.1007/s00034-019-01226-7.

[7] S. C. Magalhães, F. N. dos Santos, P. Machado, A. P. Moreira, and J. Dias, “Benchmarking edge computing devices for grape bunches and trunks detection using accelerated object detection single shot multibox deep learning models”, *Engineering Applications of Artificial Intelligence*, vol. 117, p. 105 604, 2023, ISSN: 0952-1976. DOI: https://doi.org/10.1016/j.engappai.2022.105604.

[8] P. Mousouliotis, S. Zogas, P. Christakos, *et al.*, “Exploiting vitis framework for accelerating sobel algorithm”, Institute of Electrical and Electronics Engineers Inc., Jun. 2021, ISBN: 9780738133614. DOI: 10.1109/MECO52532.2021.9460221.

[9] T. P. Dinh, C. Pham-Quoc, T. N. Thinh, B. K. D. Nguyen, and P. C. Kha, “A flexible and efficient fpga-based random forest architecture for iot applications”, *Internet of Things*, vol. 22, p. 100 813, 2023, ISSN: 2542-6605. DOI: https://doi.org/10.1016/j.iot.2023.100813.

[10] S. Valladares, M. Toscano, R. Tufiño, P. Morillo, and D. Vallejo-Huanga, “Performance evaluation of the nvidia jetson nano through a real-time machine learning application”, in *Intelligent Human Systems Integration 2021*, D. Russo, T. Ahram, W. Karwowski, G. Di Bucchianico, and R. Taiar, Eds., Cham: Springer International Publishing, 2021, pp. 343–349, ISBN: 978-3-030-68017-6.

[11] B. Yang, T. Chen, A. Chen, S. Duan, and L. Wang, “A lightweight cnn based on memristive stochastic computing for electronic nose”, *International Journal of Bifurcation and Chaos*, vol. 34, no. 03, p. 2 450 027, 2024. DOI: 10.1142/S0218127424500275.

[12] F. Schirrmeister, “Multicore architectures”, *Real World Multicore Embedded Systems*, pp. 33–73, Jan. 2013. DOI: 10.1016/B978-0-12-416018-7.00003-1.

[13] S. D. Brown, R. J. Francis, J. Rose, and Z. G. Vranesic, *Field-programmable gate arrays - stephen d. brown, robert j. francis, jonathan rose, zvonko g. vranesic - google libros 1992*.

[14] J. D. Owens, M. Houston, D. Luebke, S. Green, J. E. Stone, and J. C. Phillips, “Gpu computing”, *Proceedings of the IEEE*, vol. 96, no. 5, pp. 879–899, 2008. DOI: `10.1109/JPROC.2008.917757`.

[15] T. McKercher, J. Cheng, and M. Grossman, “Professional cuda c programming”, *John Wiley & Sons*, 2014.

[16] L. Hasan, M. Kentie, and Z. Al-Ars, “Dopa: Gpu-based protein alignment using database and memory access optimizations”, *BMC research notes*, vol. 4, p. 261, Jul. 2011, This work is licensed under the Creative Commons Attribution 2.0 International License. To view a copy of this license, visit `http://creativecommons.org/licenses/by/2.0/`. DOI: `10.1186/1756-0500-4-261`.

[17] S. K. Tewksbury, “Application-specific integrated circuits (asics)”, 1996 West Virginia University.

[18] E. Hollis, *Design of VLSI Gate Array ICs*. Prentice-Hall, 1987, ISBN: 9780132019309.

[19] “TPU v5e”, Google Cloud. [visited 01.05.2024]. (), [Online]. Available: `https://cloud.google.com/tpu/docs/v5e`.

[20] A. Zhang, Z. C. Lipton, M. Li, and A. J. Smola, “Dive into deep learning”, *Journal of the American College of Radiology*, vol. 17, pp. 637–638, 5 Jun. 2021, ISSN: 1558349X. DOI: `10.1016/j.jacr.2020.02.005`.

[21] S. Sharma, S. Sharma, and A. Athaiya, “Activation functions in neural networks”, *Towards Data Sci*, vol. 6, no. 12, pp. 310–316, 2017.

[22] J. Han, D. Wang, Z. Li, N. Dey, and F. Shi, "An improved residual-network model-based conditional generative adversarial network plantar pressure image classification: A comparison of normal, planus, and talipes equinovarus feet", ISSN: 2693-5015 This work is licensed under the Creative Commons Attribution 4.0 International License. To view a copy of this license, visit http://creativecommons.org/licenses/by/4.0/. DOI: 10.21203/rs.3.rs-262837/v1.

[23] "Vitis ai optimizer • vitis ai user guide (ug1414) • reader • amd technical information portal". [visited:22.04.2024]. (), [Online]. Available: https://docs.amd.com/r/en-US/ug1414-vitis-ai/Vitis-AI-Optimizer.

[24] D. M. L. Barbato and O. Kinouchi, "Optimal pruning in neural networks", *Physical Review E*, vol. 62, no. 6, pp. 8387–8394, Dec. 2000, Publisher: American Physical Society. DOI: 10.1103/PhysRevE.62.8387. (visited on 05/01/2024).

[25] *Vitis ai optimizer, github.io documentation*, [visited:02.04.2024]. [Online]. Available: https://xilinx.github.io/Vitis-AI/3.0/html/docs/workflow-model-development.html?cv=1.

[26] R. Gray and D. Neuhoff, "Quantization", *IEEE Transactions on Information Theory*, vol. 44, no. 6, pp. 2325–2383, 1998. DOI: 10.1109/18.720541.

[27] M. Nagel, M. Fournarakis, R. A. Amjad, Y. Bondarenko, M. Van Baalen, and T. Blankevoort, "A white paper on neural network quantization", *arXiv preprint arXiv:2106.08295*, 2021.

[28] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection", in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, ISSN: 1063-6919, Jun. 2016, pp. 779–788. DOI: 10.1109/CVPR.2016.91.

[29] T.-Y. Lin, M. Maire, S. Belongie, *et al.*, "Microsoft coco: Common objects in context", in *Computer Vision–ECCV 2014: 13th European Conference, Zurich,*

*Switzerland, September 6-12, 2014, Proceedings, Part V 13*, Springer, 2014, pp. 740–755.

[30] J. Terven and D. Cordova-Esparza, “A Comprehensive Review of YOLO Architectures in Computer Vision: From YOLOv1 to YOLOv8 and YOLO-NAS”, *Machine Learning and Knowledge Extraction*, vol. 5, no. 4, pp. 1680–1716, Nov. 2023, arXiv:2304.00501 [cs], ISSN: 2504-4990. DOI: 10.3390/make5040083.

[31] B. Koonce, “ResNet 50”, in *Convolutional Neural Networks with Swift for Tensorflow: Image Recognition and Dataset Categorization*, B. Koonce, Ed., Berkeley, CA: Apress, 2021, pp. 63–72, ISBN: 978-1-4842-6168-2. DOI: 10.1007/978-1-4842-6168-2_6.

[32] K. He, X. Zhang, S. Ren, and J. Sun, *Deep Residual Learning for Image Recognition*, This work is licensed under the arxiv nonexclusive distribution. To view a copy of this license, visit http://arxiv.org/licenses/nonexclusive-distrib/1.0/, Dec. 2015. DOI: 10.48550/arXiv.1512.03385.

[33] “Resnet | pytorch”. [visited:25.03.2024] This work is licensed under the Creative Commons Attribution 3.0 International License. To view a copy of this license, visit http://creativecommons.org/licenses/by/3.0/. (), [Online]. Available: https://pytorch.org/hub/pytorch%5C_vision%5C_resnet/.

[34] “Microsoft common objects in context dataset”. [visited:01.04.2024]. (), [Online]. Available: https://cocodataset.org/#explore.

[35] K. P., P. J., A. V., *et al.*, “Definition and classification of power system stability ieee/cigre joint task force on stability terms and definitions”, *IEEE Transactions on Power Systems*, vol. 19, no. 3, pp. 1387–1401, 2004. DOI: 10.1109/TPWRS.2004.825981.

[36] K. E. BIJKER, G. DE GROOT, and A. P. HOLLANDER, "Delta efficiencies of running and cycling", *Medicine & Science in Sports & Exercise*, vol. 33, no. 9, pp. 1546–1551, 2001.

[37] J. Winzig, J. C. A. Almanza, M. G. Mendoza, and T. Schumann, "Edge ai-use case on google coral dev board mini", in *2022 IET International Conference on Engineering Technologies and Applications (IET-ICETA)*, IEEE, 2022, pp. 1–2.

[38] S. Rivas-Gomez, A. J. Pena, D. Moloney, E. Laure, and S. Markidis, "Exploring the vision processing unit as co-processor for inference", in *2018 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, 2018, pp. 589–598. DOI: `10.1109/IPDPSW.2018.00098`.

[39] L. Hasan, M. Kentie, and Z. Al-Ars, "Dopa: Gpu-based protein alignment using database and memory access optimizations", *BMC Research Notes*, vol. 4, pp. 1–11, 1 Jul. 2011, ISSN: 17560500. DOI: `10.1186/1756-0500-4-261/TABLES/2`.

[40] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition", [visited:07.05.2024]. [Online]. Available: `http://image-net.org/challenges/LSVRC/2015/`.

[41] T. Y. Lin, M. Maire, S. Belongie, *et al.*, "Microsoft coco: Common objects in context", *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 8693 LNCS, pp. 740–755, PART 5 2014, ISSN: 16113349. DOI: `10.1007/978-3-319-10602-1\_48/COVER`.