Journal of Signal Processing Systems
https://doi.org/10.1007/s11265-023-01881-9

Application Level Resource Scheduling for Deep Learning Acceleration on MPSoC

Cong Gao[1] · Sangeet Saha[1] · Xuqi Zhu[1] · Hongyuan Jing[2] · Klaus D. McDonald-Maier[1] · Xiaojun Zhai[1]

Received: 1 December 2022 / Revised: 21 March 2023 / Accepted: 10 July 2023

Abstract
Deep Neutral Networks (DNNs) have been widely used in many applications, such as self-driving cars, natural language processing (NLP), image classification, visual object recognition, and so on. Field-programmable gate array (FPGA) based Multiprocessor System on a Chip (MPSoC) is recently considered one of the popular choices for deploying DNN models. However, the limited resource capacity of MPSoC imposes a challenge for such practical implementation. Recent studies revealed the trade-off between the “resources consumed" vs. the “performance achieved". Taking a cue from these findings, we address the problem of efficient implementation of deep learning into the resource-constrained MPSoC in this paper, where each deep learning network is run with different service levels based on resource usage (where a higher service level implies higher performance with increased resource consumption). To this end, we propose a heuristic-based strategy, Application Wise Level Selector (AWLS), for selecting service levels to maximize the overall performance subject to a given resource bound. AWLS can achieve higher performance within a constrained resource budget under various simulation scenarios. Further, we verify the proposed strategy using an AMD-Xilinx Zynq UltraScale+ XCZU9EG SoC. Using a framework designed to deploy multi-DNN on multi-DPUs (Deep Learning Units), it is proved that an optimal solution is achieved from the algorithm, which obtains the highest performance (Frames Per Second) using the same resource budget.

Keywords FPGA · Embedded systems · MPSoC · Deep Neutral networks · Hardware accelerator · Resource schedule strategy

1 Introduction

Deep Neutral Networks (DNN) have been widely used in image classification and Natural Language Processing (NLP) applications in the last decade. Due to the complexity of the increased layer interconnections and weights, the accuracy of new DNN models has been greatly improved. However, although these models can provide more sophisticated and state-of-the-art accuracy, the run-time cost of models is also increased significantly.

In a number of fields, including computer vision, bioinformatics, NLP, and robotics, to name a few, deep learning has recently become the de facto methodology [1]. Its success can be attributed to its capacity to draw knowledge from vast amounts of data. The Internet of Things is another area well known for producing enormous amounts of data (IoT). Due to recent developments in the reduction of low-power embedded devices' size and advancements in the optimization of machine learning (ML) algorithms, tiny machine learning (TinyML) is also emerging as a new Internet of Things (IoT) prospect that calls for putting the ML algorithm within the IoT device [2].

Traditionally, DNN models are normally deployed on GPUs and CPUs. However, due to resource constraints in many IoT devices, one of the widespread approaches is to implement DNNs on an ASIC (Application-specific integrated circuit) or an FPGA (field-programmable gate array). As for ASIC, this usually needs a long development cycle and cost for production, and it is unsuitable for applications that need flexibility. Therefore, to maximize the flexibility and performance of the application at run-time, FPGAs are usually a better choice due to their reconfiguration ability.

✉ Xiaojun Zhai
xzhai@essex.ac.uk

1 School of Computer Science and Electronic Engineering, University of Essex, Colchester CO4 3SQ, Essex, UK

2 Beijing Key Laboratory of Information Service Engineering, Beijing Union University, Beijing 100101, China

When deploying DNN models on FPGAs, the balance between performance and run-time cost (such as power consumption) should be considered. Although FPGAs provide a swift hardware resource allocation mechanism, the total hardware resources are limited. Usually, models with a similar network structure will perform better and cost more energy if they use more computing and memory units. However, sometimes, the performance of an embedded application is a higher priority. In comparison, the power consumption can be lowered at the cost of some acceptable accuracy loss.

Many researchers have focused on modifying the network to achieve high performance with limited resources. Mainly their objective was to reduce its size by pruning and quantizing. Recently there have been some other approaches to modifying the network and making models fit a specific hardware platform: in [3], researchers raise a framework to train the network with a flexible structure parameter (i.e. kernel size, depth, width, and channel numbers) and gain a super-network with 2×10^6 sub-networks contained and by using the network searching strategy, they can select the best network under a specific hardware platform; in [4] researchers modify the searching algorithm of OFA (once for all) and raising a dynamic network searching strategy to find a set of networks based on the accuracy and latency of the OFA super-network.

Another research direction focuses on hardware/software co-design and adjusting the hardware/software resources in a customized way with FPGAs in the design stage. For example, in [5–8], researchers develop an efficient design methodology to consider both hardware, software, and DNN structures in the network design or training stage.

Though FPGAs are becoming a popular choice for DNN tasks, resource constraints are a common bottleneck. In [9, 10], the authors have assumed that the computing server has sufficient FPGA resources to extract intermediary features using deep learning layers. However, these assumptions will be violated in many real-life cases. For example, in the case of resource-constrained IoT environment [11], successful completion of the application is more critical than achieving the higher performance [11, 12]. Hence, to successfully execute deep learning in a resource-constrained FPGA-based system, we consider each deep learning network to be equipped with multiple distinct implementations represented by *"service levels"*. Each implementation can produce the same result of prediction or classification but with different performance levels (e.g. Frames Per Second, FPS). A higher service level normally will return a higher performance but at a cost of increased resource utilization.

The research findings in [13] support the concept of distinct service levels for deploying deep learning networks. In this work, the authors have found that the memory requirement of the weight parameters contributes most to the memory footprint. Furthermore, the research further proves that a reduced precision in representing 20% weight parameters results in 1% performance loss. Taking a cue from these findings, we assumed that depending upon the availability of the resource budget, each deep learning network on an FPGA platform executed at a particular service level can be optimized in order to achieve higher performance.

In this paper, we propose a strategy for efficiently implementing deep learning into FPGA-based systems, where multiple DPUs are used for executing multiple neural networks on the application level. Further, each DNN can be executed in different service levels to achieve optimal performance.

We specifically respond to the following query: *How can we guarantee that the multiple DNNs will be effectively executed at a specific service level while maximizing the overall performance (FPS), given the resource constraints of the DPUs in FPGA?*. To this end, we proposed a heuristic-based strategy, Application Wise Level Selector (AWLS). This scheduling strategy is incorporated and further verified using a physical FPGA-based hardware/software co-design framework. This framework is based on Zynq UltraScale+ XCZU9EG multiprocessor system on a chip (MPSoC) is used to configure the "service level" of different DNN applications, and we can also calculate the overall "performance" and obtain the DPU "resource" by analyzing the data recorded with this framework. By providing the "resource" and "performance" of each DNN model at a different "service level" to the proposed strategy, it will find out an optimal solution for a multi-DNN application. It has been observed that the results obtained from the real frameworks follow a similar trend as observed in software simulation.

The contributions of this work are summarized as follows:

- Formulating the problem and development of heuristic-based, namely AWLS, for selecting service levels for deep learning applications.
- Evaluating the proposed heuristic strategy with simulation experiments and comparing it with the optimal ILP-based technique. As a result, we found that the performance of the proposed heuristic is comparable to that of ILP.

- Proposing a framework for deploying deep learning in FPGA-based MPSoC systems with multiple service levels.
- Demonstrating the proof-of-concept of the proposed strategy by implementing a multi-DNN application on an MPSoC.

2 System Model and Problem Definition

2.1 System Model

We assumed an FPGA-based system, where each FPGA may contain multiple DPUs. In the given edge computing environment, let us assume that A denotes the set of N applications (DNNs) executing on the FPGA: $A = \{A_1, A_2, ..., A_N\}$.

It has been assumed that based on the degree of resources allocated, each application will be equipped to execute in different service levels based on the available resources. Each DNN can only be executed in any one service level among the possible q service levels i.e., $l_i = \{l_i^1, l_i^2, \dots, l_i^q\}$. Hence, j^{th} service level of A_i can be denoted as l_i^j. The service of a level is proportional to its level ID. Thus, 1 is the lowest, and q denotes the highest execution level.

It can be concluded that the higher the service levels, the higher its resource consumption will be. This resource consumption could be in terms of the hardware resource, e.g. utilization. On the other hand, executing the network at a high service level will enhance the performance level more. This work assumes that higher be the service level of A_i^j, the higher its resource consumption Res_i^j ($l_i^j > l_I^{j'} \implies Res_i^j > Res_i^{j'}$). Res_i^j denotes the resource consumed by A_i while it executes in j^{th} service level [14]. Similarly, we have also assumed that performance per_i^j will be assigned to A_i^j if the i^{th} the FPGA successfully executes the deep learning network in j^{th} service level by fulfilling the resource demand. The overall resource budget $\vec{R}_{total}$ is fixed for hardware. The detailed calculation is provided in Section 2.2. Having the given $\vec{R}_{total}$, each application has to finish the execution of the deep learning network by selecting a service level.

Table 1 represents all the acronyms and their explanations used in this paper.

Table 1 Acronyms and their explanations.

Acronym	Explanation
DPU	Deep Learning Unit
AWLS	Application Wise Level Selector
VART	Vitis AI Run-time
DF	Decision Factor
PRP	Performance Per Unit Resource
IGF	Immediate Gain Factor
OGF	Overall Gain Factor
DNN	Deep Neural Network

2.2 Mathematical Representation of the Problem

In this section, we will attempt to formalize the proposed problem based on the system mddel described in the previous section For this purpose, we define a binary decision variable: i. $\mathcal{Z} = \{Z_i^j : i = 1, 2, ..., N; j = 1, 2, ..., q$. Here, indices i and j respectively denote applications and corresponding selected service level ID. $Z_i^j = 1$, if application A_i executes in j^{th} service level and obtains Per_i^j performance value. $Z_i^j = 0$, otherwise.

We now present the required constraints on the decision variable to model this problem before presenting its overall objective function.

1. Overall resource budget constraint: The complete amount of resources ($\vec{R}_{total}$) in hardware must be used to execute the deep learning network. This basically indicates that the total amount of resources used by the available accelerators shouldn't exceed the total hardware allocated budget. The following equation imposes this restriction.

$$\sum_{i=1}^{N} \sum_{j=1}^{q} Res_i^j \times Z_i^j \leq \vec{R}_{total} \tag{1}$$

   where the Res_i^j refers to the resource utilization when application A_i executes in j^{th} service level.
2. Unique service level execution constraint: Each application will only be allowed to execute the deep learning network at one certain service level. That is,

$$\sum_{j=1}^{q} Z_i^j \leq 1, \forall i \in [1, N], Z_i^j \in \{0, 1\} \tag{2}$$

3. **Objective:** The objective of the formulation is to choose a feasible solution that maximizes the overall performance of the prediction/testing process through the appropriate choice of service levels. Hence, the objective can be written as follows:

$$Maximize \quad \sum_{i=1}^{N} \sum_{j=1}^{q} Per_i^j \times Z_i^j \tag{3}$$

3 AWLS: <u>A</u>pplication <u>W</u>ise <u>L</u>evel <u>S</u>elector Heuristic

In this section, we propose a heuristic strategy named the Application Wise Level Selector (AWLS). It is a fast yet efficient heuristic strategy that allows resource balance to be restored quickly through a greedy but elegant approach proceeding level by level so that higher overall performance can be achieved. It is observed that, in order to achieve a good result, AWLS must be aware of the remaining resources at any stage of the algorithm. Thus, AWLS must be aware of individual performances during service level enhancements. Along with this, it also needs to consider the account for the number of incremental resources required during such level enhancement processes.

In order to achieve this objective, we have transformed the parameter "performance" to "*PPR*" (**P**erfromance **P**er Unit **R**esource) and defined two new factors. The first factor is termed as *IGF* (**I**mmediate **G**ain **F**actor). IGF defines the difference in *PPR* between a current level (l) and the immediate next higher ($l + 1$) level. Thus, the Immediate Gain Factor (IGF_i) for an application A_i can be calculated as:

$$IGF_i = \frac{Per_i^{l+1} - Per_i^l}{Res_i^{l+1} - Res_i^l} \tag{4}$$

Similarly, we have defined another factor called *OGF* (**O**verall **G**ain **F**actor). OGF defines the difference in *PPR* between a current level (l) and the maximum possible service level (q). Thus, the Overall Gain Factor (OGF_i) for A_i can be calculated as:

$$OGF_i = \frac{Per_i^q - Per_i^l}{Res_i^q - Res_i^l} \tag{5}$$

Based on these derived factors, AWLS generates a key term called as "Decision Factor (DF)". While selecting a level, the applications are maintained in a max-heap. An application will be selected for the level up-gradation within the heap, based on "*DF*" which is defined for each A_i as follows:

$$DF_i = max(IGF_i, OGF_i) \tag{6}$$

It can be observed that DF_i is able to provide an appropriate balance between IGF and OGF.

Implication of Decision Factor (DF) Let us consider that the two deep learning networks are currently executing in A_i and A_j, respectively. A_i is executing in priority level l and A_j is executing in level l'. Let us assume based on the selected level, the internal gain for A_i i.e. IGF_i is lower than IGF_j, i.e. the internal gain for A_j. However on the other hand, $OGF_i >> OGF_j$. In this case, if OGF values are not considered as a part of the *Decision Factor* (*DF*), A_j will be selected for level upgradation by one over A_i, despite the fact that the OGF_i is much higher than OGF_j. In the worst case, A_i will hardly get the opportunity of level upgradation, in spite of having high OGF_i. Hence, in such typical scenarios, *DF* will play an important role.

Working Principle of AWLS The working strategy of the AWLS is described as follows. As we can observer, in line 1, AWLS calculates *Decision Factor* (DF_i) for all applications. Based on the calculated *DF* values, AWLS constructs a max-heap (ref. line 3). Initially, the service level for all applications is set to one. As we can observe, from line 7 to 11, AWLS iteratively increments (updates) the service level for all applications till the *max-heap* ($\mathcal{H}$) is empty. Then AWLS extracts A_i from the root of the $\mathcal{H}$ (Ref. line 8). If the Increment in Resource (IR) is greater than the available resource budget (RES_BGT) then A_i is removed from the max-heap. Otherwise, the AWLS increments its service level by 1, updates the *DF* value, and re-cqlculates it in line 12. If any A_i reaches its highest possible priority level (ref. line 11), then the application is removed from the *max-heap*.

```
Algorithm 1: AWLS
   Input:
   i. $Res_i^j$: Resource required for executing application $A_i$ at the $j^{th}$ level
   ii. $per_i^j$: Performance obtained by executing $A_i$ at the $j^{th}$ level
   iii. $RES\_BGT$: The overall resource budget for the FPGA
   Output:
   Selected service level for each DPU
1  begin
2      Compute $DF_i$ using equation 6, $\forall A_i$;
        Store all the calculated $DF$ values in a constructed max-heap $\mathcal{H}$;
        /* Let $Curr_{lev}^i$ represents the current level for each $A_i$; */;
        Initialize $Curr_{lev}^i = 1$; /* At the beginning service level for all
        applications set to one */;
        /* Let $IR$ increment in resource, if there is upgradation of level; */;
        while $\mathcal{H} \neq NULL$ do
3          Extract $A_i$ from the root of the max-heap $\mathcal{H}$;
            if $Curr_{lev}^i$=1 then
4              $IR = Res_i^{Curr_{lev}^i+1}$ ;
5          else
6              $IR = Res_i^{Curr_{lev}^i+1} - Res_i^{Curr_{lev}^i}$;
7          if $RES\_BGT < IR$ then
8              Continue Execution;
                $A_i$ gets removed from $\mathcal{H}$;
                /* There is no available resource that can upgrade level of $A_i$
                level*/
9          else
10             $RES\_BGT = RES\_BGT - IR$; /* Update the overall available
                resource */
                $Curr_{lev}^i$=$Curr_{lev}^i$+1; /*Increase the current level*/
                if $Curr_{lev}^i$=q then
11                 Continue Execution;
                    $A_i$ gets removed from $\mathcal{H}$;
                    /* $A_i$ reached its highest level*/
12             Recalculate $DF_i$ using equation 6;
                Re-heapify $A_i$ in $\mathcal{H}$;
```

3.1 AWLS at Work

In this section, we have illustrated the working mechanism of AWLS through an example for ease of understanding. Let us assume, there exist three applications,i.e., A_1, A_2, and A_3 inside the FPGA. Resource demand (Res_i^j) and corresponding performance per_i^j value for each service level is provided in Table 2. We have also assumed that the available overall resource budget (*RES_BGT*) is 35.

Table 2 Resource and performance values for each DPU.

A_1			A_2			A_3		
Service level	Required resource	Obtained performance	Service level	Required resource	Obtained performance	Service level	Required resource	Obtained performance
1	2	12	1	6	2	1	7	4
2	5	13	2	14	9	2	10	6
3	7	16	3	18	16	3	13	8

Table 3 Outcome: AWLS.

Applications	Selected level	Obtained accuracy
A_1	3	16
A_2	3	16
A_3	2	6
Total obtained accuracy		38

AWLS will begin its operation by calculating the DF value by Eq. 6. The initial *DF* values can be calculated as follows: $DF_1 = max(\frac{13-12}{5-2} \frac{16-12}{7-2}) = 0.8, DF_2 = max(\frac{9-2}{14-6} \frac{16-2}{18-6}) = 1.16$, $DF_3 = max(\frac{6-4}{10-7} \frac{8-4}{13-7}) = 0.67$. Max-heap $\mathcal{H}$ is constructed using these *DF* values. A_2 has the highest *DF* value and is extracted from the heap. *RES_BGT* is updated as (35 – 14) = 21 and $Curr^2_{lev}$ becomes 2. Now, the DF_2 will be re-calculated as $DF_2 = max(\frac{16-9}{18-14} \frac{16-9}{18-14}) = 1.75$, and again A_2 has the highest *DF* value. Hence, the *IR* becomes (18 – 14) = 4 and the condition satisfies the remaining resource becomes $RES\_BGT = 17$ and $Curr^2_{lev}$ becomes 3. A_2 is discarded from further consideration of service level upgradation as reached it its highest level 3. Now, A_1 with the highest *DF* value is extracted from the heap and similarly by completing the iterations, the $Curr^1_{lev}$ becomes 3, and the remaining resource is updated as $RES\_BGT = 17 - 7 = 10$. It can be observed that A_1 also reached its highest possible level and hence, discarded from further consideration. The next iterations follow for A_3 and the level for the A_3 is upgraded accordingly. AWLS terminates when the heap becomes empty. The total obtained result is shown in Table 3.

4 Performance Evaluation of AWLS

The performance of the proposed AWLS has been evaluated using simulation-based experiments. We have also compared the performance of the proposed heuristic with the optimal ILP-based strategy. In this current experimental scenario, We have considered that the FPGA can execute deep learning networks in 5 distinct service levels, and FPGA consists of 2 DPUs, as shown in [15]. The area consumption and corresponding performance values have been taken from [16].

4.1 Results

Experiments have been conducted to evaluate the performance of the proposed strategies i.e., ILP-based technique and AWLS using different performance metrics under varying scenarios. The performance metrics that have been considered for the evaluation are:

1. Average service level allocated to each server
2. **N**ormalized **O**btained **P**erformance (*NOP*), *NOP* is defined as the ratio between the ultimately achieved performance value for all the applications and the maximum possible achievable performance by executing each application at its highest service level. Mathematically, *NOP* can be formulated as:

$$NOP = \frac{\sum_{i=1}^{N}(\frac{Per_i^j}{Per_i^q})}{N} \times 100\% \tag{7}$$

Figure 1 Average allocated level Vs RES_BGT.

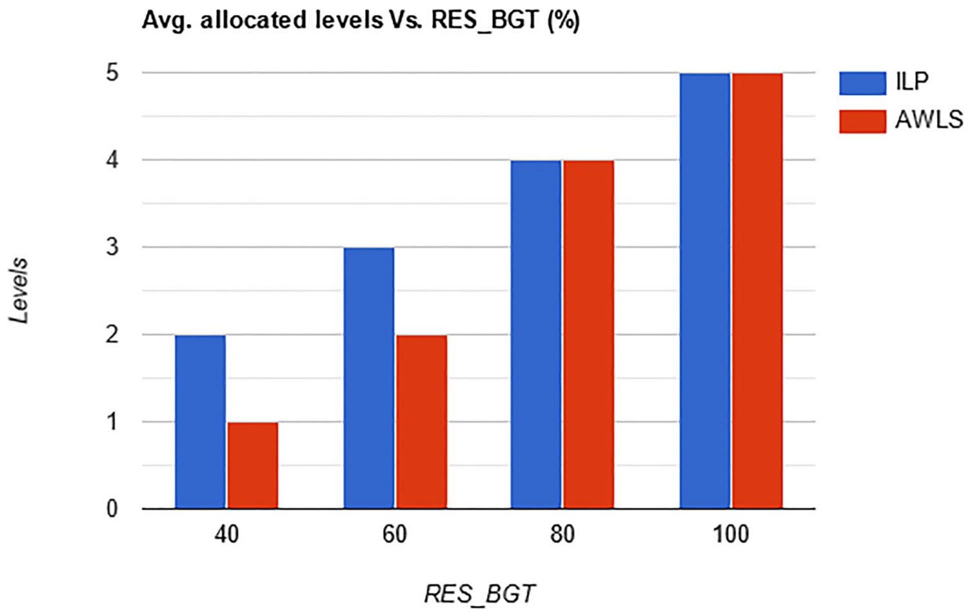

Figure 2 NOP Vs RES_BGT.

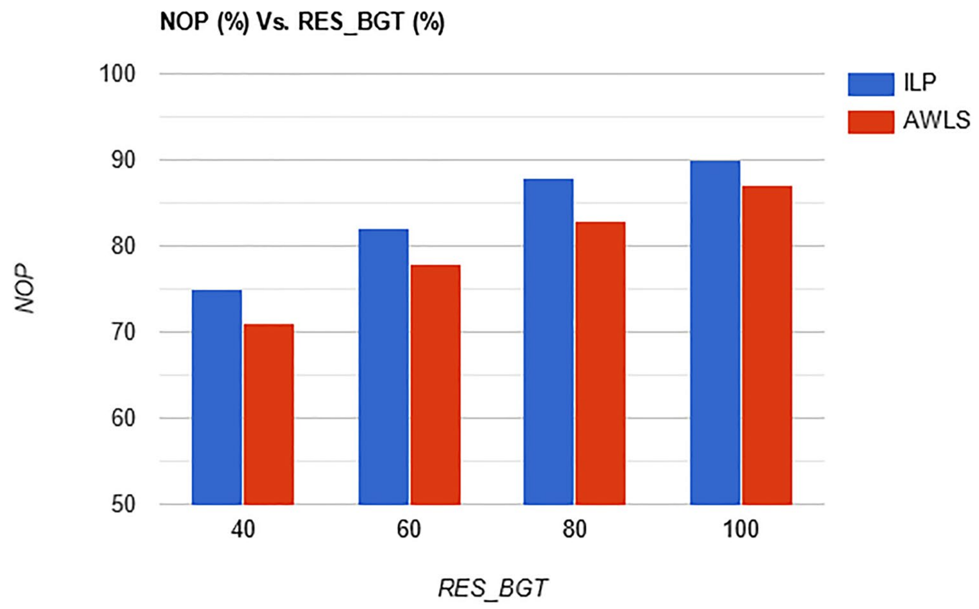

Figure 1 shows the plots for the average levels allocated to each application by both the strategies i.e. ILP-based strategies and AWLS. As the overall resource budget (*RES_BGT*) varies from 40% to 100% of the total available resource budget. It may be observed from the figure that the average level allocated to each application increases with the increasing available overall resource budget. This is because the average resource that may be utilized by an application increases as the total available overall increases.

Although the trends for both the allocation strategies in Fig. 1 are mostly similar, AWLS is seen to allocate slightly lower average levels than ILP-based techniques in all the scenarios. However, this difference in performance decreases with the increase in resources. Hence, the performance of both strategies becomes comparable when there exists an adequate amount of resources. This could be attributed to the fact that as the individual resource increases, the difference between the values of IGF and OGF also increases. Hence, DF plays a significant role in level selection. Thus, AWLS takes judicious level selection decisions that are close to the optimal.

Figure 2 depicts the plot for *NOP* achieved by both the strategies, as the overall resource budget (*RES_BGT*) varies from 40% to 100% of the total available resource budget. It may be observed from the figure that the aggregate *NOP* obtained by both the strategies increase with increasing available *RES_BGT*. This is because the *NOP* obtained by the strategy is directly proportional to the achieved service levels of each application and therefore, obtained levels increase with the available resource budget (as shown in Fig. 1). Additionally, it may be observed from the figure that as the difference between available resources decreases, the performance difference between both strategies is negligible.

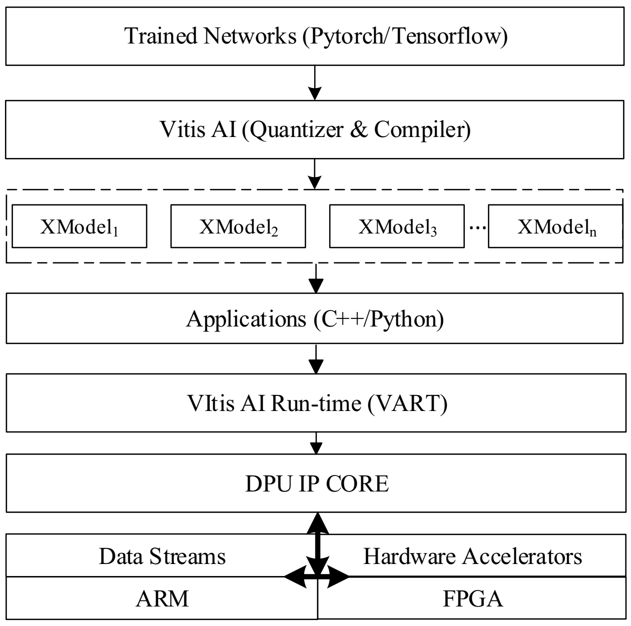

Figure 3 Overview of the proposed framework.

5 Implementation of the Proposed Framework

To further verify the proposed strategy, we implemented a framework using a ZCU104 development board equipped with a Zynq UltraScale+ XCZU9EG MPSoC. In our previous work, a video analysis system is designed using the proposed framework in [17], and in this paper, we further explore the resource scheduling algorithms to achieve optimal performance.

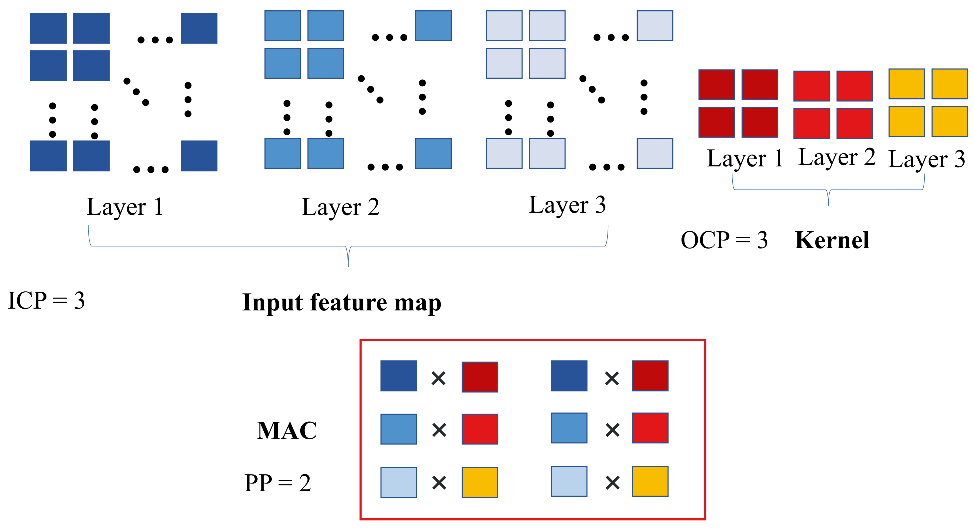

Figure 4 An example of DPU internal arithmetic operation flow.

AMD-Xilinx DPU IP module and Vitis-AI library are used [18, 19] in this framework. Figure 3 shows the overview of the framework diagram. We will now discuss the different components of the proposed framework.

5.1 Vitis AI Run-time (VART)

Vitis AI Runtime (VART) is a part of Vitis-AI software that enables the applications to interact with the hardware by calling the unified high-level API. VART offers asynchronous submission and collection of jobs to the accelerator and supports multi-threading, and multi-process execution [19].

5.2 DPU (Deep Learning Unit)

DPU is an AMD-Xilinx hardware IP core, which can support DNN instructions that are compiled from conventional DNN development frameworks (e.g. PyTorch, TensorFlow, etc.) using the Vitis-AI toolchain.

There are eight different DPU architectures in the Vitis-AI library, where each architecture is configured according to the three dimensions of parallelism: pixel parallelism (PP), input channel parallelism (ICP), and output channel parallelism (OCP).

Figure 4 shows an example of each of the three dimensions. For instance, pixel parallelism (PP) is 2, and input channel parallelism and output channel parallelism are equal to 3. Due to the nature of the calculation, the input channel parallelism is always similar to the output channel parallelism. In general, the larger DPU architectures can achieve better throughput than the smaller DPUs at the cost of more hardware resources. Table 4 lists all DPU architecture and their parallelism parameter configurations.

Table 4 DPU Configurations.

DPU architecture	PP	ICP	OCP	Peak operations
B512	4	8	8	512
B800	4	10	10	800
B1024	8	8	8	1024
B1152	4	12	12	1152
B1600	8	10	10	1600
B2304	8	12	12	2304
B3136	8	14	14	3136
B4096	8	16	16	4096

5.3 Proposed Framework

To implement the framework, there are two parts to handle, the hardware designed on the PL (Programmable Logic)/FPGA part and the Linux system designed on the PS (Processing System)/CPU part. A DFX (Dynamic Function eXchange) hardware platform is designed with the DPU IP, and the DPU is set into a special reconfigure area to enable real-time partial reconfiguration. The hardware platform will be exported into AMD-Xilinx official Linux-system design tools, Petalinux, and with the help of that, an embedded Linux system with a configurable hardware setting is created. On this system, we can access the DPUs and send tasks to them via VART, so various DNN model-based applications can be designed and tested on the board. DNN model-based applications run both on ARM-based CPUs and the DPUs accelerators on FPGAs. As can be seen in Fig. 5, The CPU and FPGAs are physically connected through high-speed AXI (Advanced eXtensible Interface) protocol, which enables high bandwidth data movements between hardware and software. For example, the instruction fetch unit in the DPU will fetch the instructions of the DNN models through VART and XRT (i.e. Xilinx runtime),

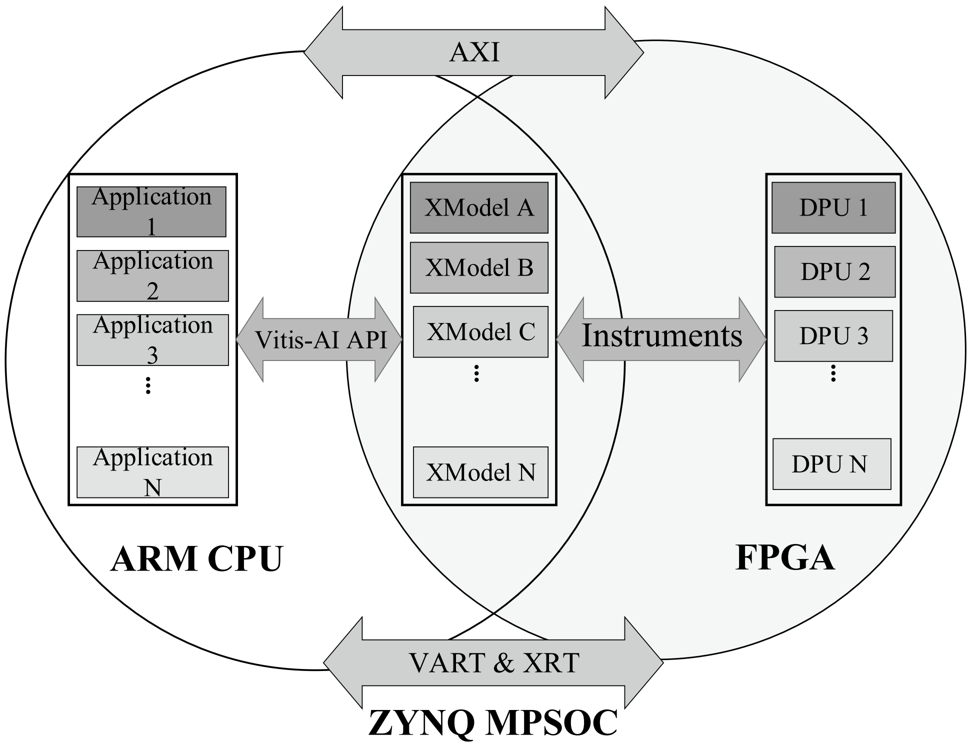

Figure 5 Proposed Zynq-based MPSOC framework.

and then the sequence of instructions will be loaded from DDR memory to the computing unit of DPUs. A DPU “Instruction" is a basic operator for the DPU arithmetic calculation, such as a “convolution operation" which is a sequence of instructions to perform a convolution operation. In the proposed framework, 2 DPU IP cores are configured to run different DNN models for image classification applications. The three DNN models are compiled into different arithmetic operations, the two DPUs will then process them in the order.

6 Exhibition of the Proof-of-Concept

Before presenting our proof-of-concept study, we revisit the problem description from a physical implementation perspective. We have several different DNN-based models running on various service levels. A single application on a higher service level will cost more DPU utilization, and the individual cost is varied on the scales of the deployed DNN models. Our goal is to fully use the DPU resources to achieve optimal running performance.

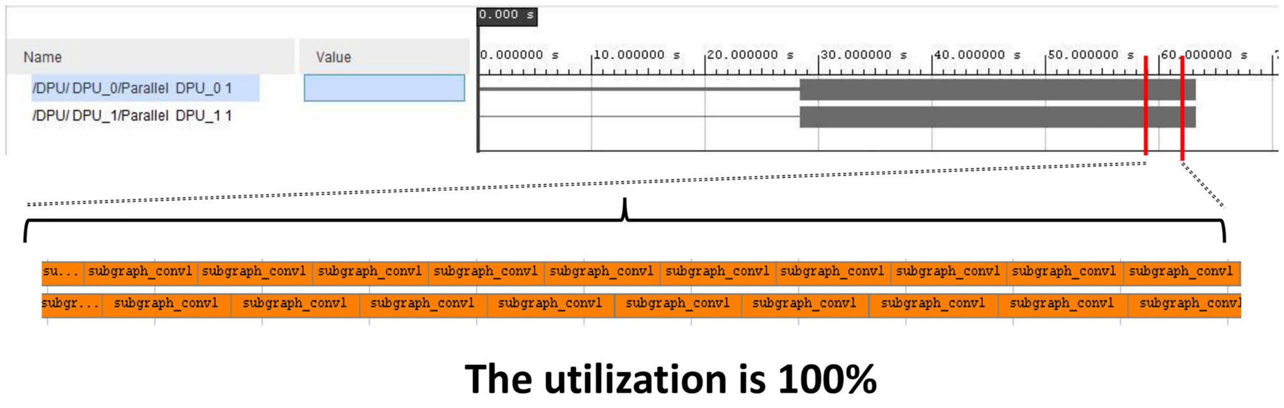

Figure 6 DPU utilization in the benchmark test.

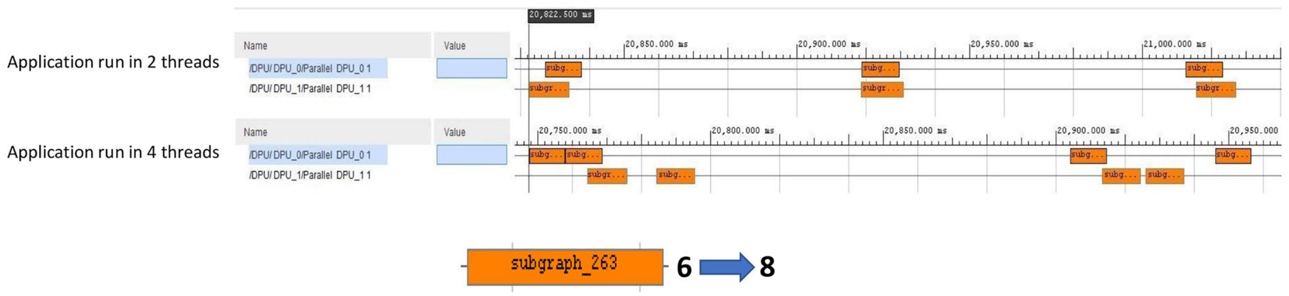
Application run in 2 threads
Application run in 4 threads
Name
Value
/DPU/ DPU_0/Parallel DPU_0 1
/DPU/ DPU_1/Parallel DPU_1 1
subgraph_263
6
8

Figure 7 DPU utilization in the application tests with different threads.

A number of onboard tests are designed to verify the proposed strategy, and the proposed DNN-based multi-application framework is implemented using an AMD-Xilinx ZCU104 development board. In the test, we test three different DNN models for image classification applications (e.g. Resnet50, Resnet18, and Mobilenet), and then the system metrics in real-time are recorded accordingly, which includes the FPS of each application, peak GOP (Giga [billion] Operations Per Second) of DPU, and total time consumption. The following sections introduce the details of the experiment and the definitions of "service level", "resource," and "performance" in the experiments carried out on the physical FPGA.

6.1 Representation of "Service Levels"

In the proposed experiment, the number of threads is chosen to represent the different service levels. The experiment aims to classify images, and multi-thread enables it to process multiple images simultaneously. While applications run with multi-threads, the quad-core (ARM Cortex™-A53) will be able to send more image data to the DPUs simultaneously, thus more DPU utilization will be allocated. To highlight the difference in performances at each service level, the following equation is used to describe the thread and service level:

$$Thread = 2^{ServiceLevel-1} \tag{8}$$

6.2 Representation of "Resource" in MPSoC

The DPU utilization is used to represent the notion of resources for physical MPSoC. First, a DPU performance benchmark is proposed to use the DPU resource fully. The benchmark will generate synthetic data and keep DPUs utilization full all the time, as it is shown in Fig. 6, DPU is processing CONV operation all the time. Second, a DL-based image classification application is used in the test, where different threads (e.g. threads 1-8) are used for the testing. In this case, DPUs have some idle time due to waiting for the new data from the CPU, therefore, there exists a gap between two basic DPU operations (CONV) and the total utilization of the DPU is less than 100%. For instance, Fig. 7 shows a comparison of DPU utilization between two threads and four threads settings. In the same period of time, it processes only 6 "CONV" operations when the application runs in 2 threads and processes 8 when runs in 4 threads.

The GOP is considered to describe a proper "resource" value. We choose the average GOP in the time scale (GOP/s) as a standard unit for utilization of DPUs and consider the average GOP in the benchmark as 100% used DPU resource. In order to obtain the resource usage in different service levels with various DNN models, we compare the average GOP in a DL image classification task with the benchmark results respectively.

Thus we obtain all the "resource" values with three DNN models at different service levels presented in Table 5.

Table 5 Resource and performance values for each Application.

$A_1(Resnet50)$			$A_2(Resnet18)$			$A_3(Mobilenet)$		
Service level	Required resource	Score	Service level	Required resource	Score	Service level	Required resource	Score
1	133	20	1	79	23	1	14	23
2	140	21	2	88	24	2	15	24
3	166	24	3	89	25	3	15	25
4	167	24	4	88	24	4	15	24

Table 6 Applications scheduling solution suggested by AWLS.

Combination	Service level		
	Resnet50	Resnet18	Mobilenet
A	3	1	3
B	1	3	3
C	2	3	2

Table 7 Outcome: Heuristic AWLS & ILP.

Deep learning network	Selected level	Obtained score
D_1	3	24
D_2	1	23
D_3	3	25
Total obtained performance		72

6.3 Representation of the "Performance" in Experiment

To define the actual onboard performance for each service level. In the proposed experiment, FPS is used as the main performance metric. The equation below explains how the FPS is calculated, where I denotes the number of images processed in each thread, T denotes the total number of threads used, and P denotes the total time consumption.

$$FPS = \frac{I \times T}{P} \tag{9}$$

6.4 Result and Analysis

After processing the data in the experiment, we calculate the optimal result using both ILP and AWLS (RES_BGT = 260) algorithms, and the result is the same as the simulation, see Table 7.

An image classification application with three different DNNs models is designed in the proposed framework to verify the result given by the proposed scheduling strategy. In this experiment, we test three different DNN combinations with various service levels at runtime. Then we measure the performance to verify whether the output combination selected by the proposed scheduling strategy can achieve the best result.

As per Table 6, Combination-A of the strategy is set at D1, D2, and D3 on service-level 3, 1, and 3, respectively. We also set the other two combinations according to the resource budget. For example, combination-B is set at D1, D2, and D3 on service-level 1, 3, and 3, respectively. Combination-C is set at D1, D2, and D3 on service levels 2,3, and 2, respectively.

Figure 8, illustrates that combination-A (chosen by the proposed strategy) obtains the highest score compared to combination-B and combination-C. At the same time, the recourse budget (RES_BGT = 260) remains the same as we can observe, combination A that various deep learning model is running at different service level. While another interesting observation can be drawn from this figure, i.e., though combination B obtains less score, its resource

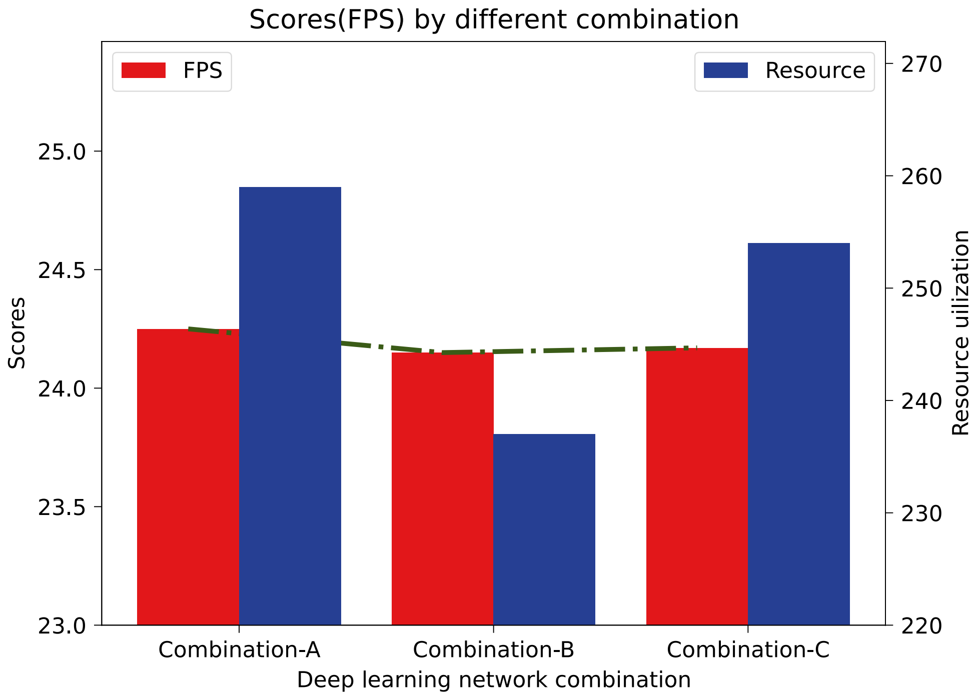

Figure 8 Comparison with different DNN combinations.

consumption is significantly less. Thus, it exhibits the efficacy of our proposed idea of "service levels". In case of stringent resource constraints, our strategy will be able to select a different combination of deep learning models with less difference in score.

It is to be noted that the performance on the physical test bed for multi-application does not match the "score" obtained by the algorithm (AWLS). This is mainly due to the fact that we set FPS to describe the performance, and there is not a simple linear relationship with the FPS while deploying multi-DNNs. The algorithm can guide us to arrange the optimal combination for the applications but can not predict the output FPS (see Table 7).

7 Conclusion

This work introduces a new concept of efficient implementation of deep learning with multiple service levels for FPGAs. The problem has been formulated as an optimization problem where each DNN can be executed with different service levels by exhibiting *performance Vs. Resource trade-offs*. A heuristic strategy (AWLS) has been proposed to maximize overall performance without violating resource constraints. Then a proof of the concept for the proposed strategy using a Xilinx ZCU104 development board is presented, and then a set of tests is designed to discuss the DPU resource allocation mechanism and define the onboard concept of "service level", "resource," and "performance" raised in the strategy. Finally, with a framework designed to deploy the multi-DNN application, the proposed solution can achieve the highest performance (FPS) using the same resource budget. Our future work will consider using the different DPU architectures and frequencies. Our final goal is to establish an adaptive system where DNN models and hardware resource utilization can be reconfigured at runtime using a real-time scheduling strategy.

Author Contributions CG and SS collected data and performed most analyses and experiments as well as drafted the manuscript. XQZ performed some analysis and summarised the conclusion. XJZ, HJ, KDM, and SS conducted to review and editing and provided critical contributions to the manuscript. All authors have read and agreed to the published version of the manuscript.

Funding This work is supported by the UK Engineering and Physical Sciences Research Council through grants EP/R02572X/1, EP/V034111/1, EP/V000462/1, EP/X015955/1, and EP/P017487/1.

Data Availability Not Applicable.

Declarations

Ethics Approval Not Applicable.

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

References

1. Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep learning*. MIT press.
2. Dutta, L., & Bharali, S. (2021). TinyML meets IoT: A comprehensive survey. *Internet of Things, 16*, 100461.
3. Cai, H., Gan, C., Wang, T., Zhang, Z., & Han, S. (2019). Once-for-all: Train one network and specialize it for efficient deployment. Preprint retrieved from http://arxiv.org/abs/1908.09791
4. Lou, W., Xun, L., Sabet, A., Bi, J., Hare, J., & Merrett, G. V. (2021). Dynamic-OFA: Runtime DNN architecture switching for performance scaling on heterogeneous embedded platforms. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition* (pp. 3110–3118).
5. Korol, G., Jordan, M. G., Rutzig, M. B., & Beck, A. C. S. (2022). AdaFlow: A framework for adaptive dataflow CNN acceleration on FPGAs. In *2022 Design, Automation & Test in Europe Conference & Exhibition (DATE)* (pp. 244–249). IEEE.
6. Wang, Z., Xu, K., Wu, S., Liu, L., Liu, L., & Wang, D. (2020). Sparse-YOLO: Hardware/software co-design of an FPGA accelerator for YOLOv2. *IEEE Access, 8*, 116569–116585.
7. Hao, C., Zhang, X., Li, Y., Huang, S., Xiong, J., Rupnow, K., Hwu, W-M., & Chen, D. (2019). FPGA/DNN co-design: An efficient design methodology for 1ot intelligence on the edge. In *2019 56th ACM/IEEE Design Automation Conference (DAC)* (pp. 1–6). IEEE.
8. Lu, Y., Zhai, X., Saha, S., Ehsan, S., & McDonald-Maier, K. D. (2022). A self-adaptive SEU mitigation scheme for embedded systems in extreme radiation environments. *IEEE Systems Journal*.
9. Li, H., Ota, K., & Dong, M. (2018). Learning IoT in edge: Deep learning for the internet of things with edge computing. *IEEE Network, 32*(1), 96–101.
10. Liu, C., Cao, Y., Luo, Y., Chen, G., Vokkarane, V., Yunsheng, M., Chen, S., & Hou, P. (2017). A new deep learning-based food recognition system for dietary assessment on an edge computing service infrastructure. *IEEE Transactions on Services Computing, 11*(2), 249–261.
11. Wang, S., Tuor, T., Salonidis, T., Leung, K. K., Makaya, C., He, T., & Chan, K. (2019). Adaptive federated learning in resource constrained edge computing systems. *IEEE Journal on Selected Areas in Communications, 37*(6), 1205–1221.
12. Mohammadi, M., Al-Fuqaha, A., Sorour, S., & Guizani, M. (2018). Deep learning for IoT big data and streaming analytics: A survey. *IEEE Communications Surveys & Tutorials, 20*(4), 2923–2960.
13. Deng, Z., Xu, C., Cai, Q., & Faraboschi, P. (2015). Reduced-precision memory value approximation for deep learning. *Hewlett Packard Labs, HPL-2015-100*.

14. Gao, C., Saha, S., Lu, Y., Saha, R., McDonald-Maier, K. D., & Zhai, X. (2022). Deep learning on FPGAs with multiple service levels for edge computing. In *2022 27th International Conference on Automation and Computing (ICAC)* (pp. 1–6). IEEE.
15. Goel, S., Kedia, R., Balakrishnan, M., & Sen, R. (2020). Infer: Interference-aware estimation of runtime for concurrent CNN execution on DPUS. In *2020 International Conference on Field-Programmable Technology (ICFPT)* (pp. 66–71). IEEE.
16. Lin, G-Z., Nguyen, H. M., Sun, C-C., Kuo, P-Y., & Sheu, M-H. (2021). A novel bird detection and identification based on DPU processor on PYNQ FPGA. In *2021 IEEE International Conference on Consumer Electronics-Taiwan (ICCE-TW)* (pp. 1–2). IEEE.
17. Lu, Y., Gao, C., Saha, R., Saha, S., McDonald-Maier, K. D., & Zhai, X. (2022) FPGA-based dynamic deep learning acceleration for real-time video analytics. In *35th GI/ITG International Conference on Architecture of Computing Systems*. IEEE.
18. AMD-Xilinx. (2022). DPUCZDX8G for zynq ultrascale+ MPSoCs product guide (pg338). Retrieved August 2022, from https://docs.xilinx.com/r/en-US/pg338-dpu/reg_dpu_isr
19. AMD-Xilinx. (2022). Xilinx Vitis-ai 2.5 release. Retrieved August 2022, from https://docs.xilinx.com/r/en-US/ug1414-vitis-ai