Advancing Compilation of DNNs for FPGAs using Operation Set Architectures

Burkhard Ringlein, Francois Abel, Dionysios Diamantopoulos, Beat Weiss, Christoph Hagleitner, and Dietmar Fey

Abstract—The slow-down of technology scaling combined with the exponential growth of modern machine learning and artificial intelligence models has created a demand for specialized accelerators, such as GPUs, ASICs, and field-programmable gate arrays (FPGAs). FPGAs can be reconfigured and have the potential to outperform other accelerators, while also being more energy-efficient, but are cumbersome to use with today's fractured landscape of tool flows. We propose the concept of an operation set architecture to overcome the current incompatibilities and hurdles in using DNN-to-FPGA compilers by combining existing specialized frameworks into one organic compiler that also allows the efficient and automatic re-use of existing community tools. Furthermore, we demonstrate that mixing different existing frameworks can increase the efficiency by more than an order of magnitude.

Index Terms—Reconfigurable hardware, Domain-specific architectures, Compilers, Artificial Intelligence

1 Introduction

Machine learning (ML) and artificial intelligence (AI) have evolved dramatically over the past decade. Currently, the complexity of the best models doubles multiple times per year [1]. This evolution has been enabled and fueled by five decades of technology scaling, which is winding down today due to the end of Dennard scaling and the slowing down of transistor development compared to Moore's law. Hence, accelerators, such as GPUs or application-specific integrated circuits (ASICs), have become indispensable for ML. However, both accelerator types have disadvantages when it comes to custom data types, novel deep neural network (DNN) topologies, or simple adaptability to new models over time. Furthermore, the fixed architectures of GPUs and ASICs often limit research on new AI models [1]. Due to these drawbacks, reconfigurable accelerators, such as field-programmable gate arrays (FPGAs), have become increasingly popular. Using FPGAs, the architecture[1] can be adapted precisely to the requirements of each application and its data flow and data types. Consequently, FPGAs can outperform GPUs in terms of latency, throughput, and energy-efficiency [2], [3].

This superior flexibility and efficiency of FPGAs in accelerating AI models should have led to a wide adoption over the past years. However, this did not happen yet due to many problems that hinder the deployment of DNN-to-FPGA tools in the wider community. In this paper, we analyze the limitations of the state of the art and propose two architectural concepts — *operation set architecture* and *organic compilation*— to overcome the identified shortcomings.

- *B. Ringlein, F. Abel, D. Diamantopoulos, B. Weiss, and C. Hagleitner are with IBM Research Europe, Säumerstrasse 4, 8803 Rüschlikon, Switzerland. E-mail: {ngl, fab, did, wei, hle}@zurich.ibm.com*
- *B. Ringlein and D.Fey are with the Department of Computer Science, Computer Architecture of the Friedrich-Alexander University Erlangen-Nürnberg, Germany. E-mail: {burkhard.ringlein, dietmar.fey}@fau.de*

- *Acknowledgment: This work is partially funded by the EU Horizon 2020 Programme under grant agreement No 957269 (EVEREST).*

1. **Terminology:** In this work, the term **topology** refers to the structure of a neural network, i.e. its internal structure of operations or layers. **Architecture** refers to the hardware implementation, and its properties, of a neural network topology. **Accelerator** refers to the subset of an architecture (or to the complete architecture). **Framework** refers to a set of scripts used to turn a topology into an architecture (e.g. hls4ml [4]). **Tool flow** refers to a set of tools used to map (i.e. compile, synthesize, place and route) an accelerator to an FPGA device.

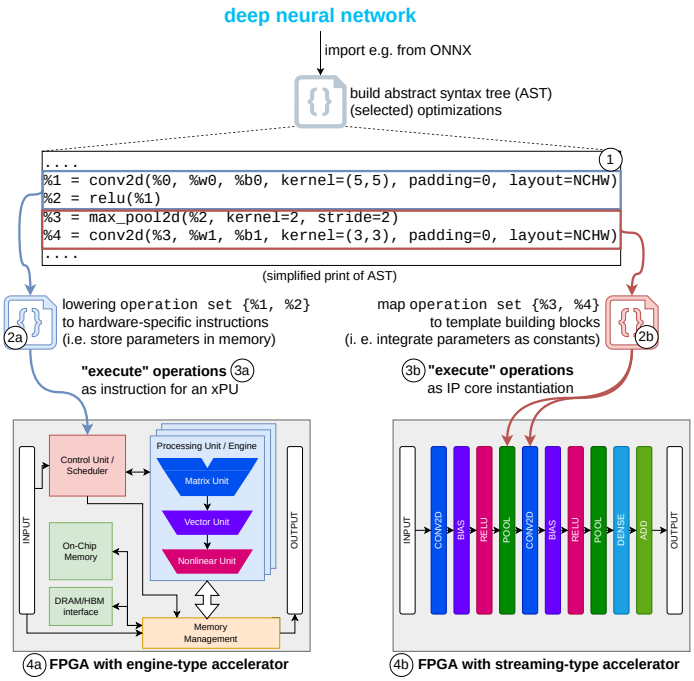


Fig. 1. Basic principle of the Operation Set Architecture (OSA). An operation can be "executed" either by lowering it to an instruction (for engine-type accelerators, left hand side) *or* by implementing it as parameterized hardware IP core (for streaming-type accelerators, right hand side).

2 Motivation and Related Work

2.1 Patterns of existing DNN-to-FPGAs flows

The FPGA community has been researching the implementation of neural networks on FPGAs for nearly 30 years, resulting in a "Cambrian explosion"[2] of DNN-to-FPGA tools[1] that scale from Edge to Cloud and target a wide variety of applications. Despite this variety, the architectures generated by all these existing frameworks can be sorted into two categories: **Engine-type** and **streaming-type** architectures, as depicted in the lower half of Figure 1.

The engine-type (see 4a in Figure 1) consists of one or multiple custom designed processing units (i.e. engines) that can execute domain specific instructions and are often referred to as "NPU" or "xPU". These processing engines frequently contain dedicated units for matrix multiplication, vector processing,

This is the accepted version of the article published by IEEE: DOI 10.1109/LCA.2022.3227643

and non-linear functions, since these are the mathematical foundations of today's DNNs. Consequently, a DNN is broken-down by a compiler into instructions that can be handled by those processing engines. These instructions are issued by a control unit at run-time and scheduled based on memory dependencies and processing unit availability. Although this pattern is simple, the design-space is huge: For example, the processing elements can contain a variety of different specialized units, with different data sizes or types. Examples of this type of architecture are TVM's VTA [5], Xilinx's Vitis AI, and Microsoft's Brainwave [2].

The streaming-type architecture (see (4b) in Figure 1) integrates all the application-specific operations in the FPGA logic, so that the data "just" streams through the fabric at run-time. This type of architecture can achieve a higher throughput with lower latencies, at the cost of higher resource usage, compared to the engine-type. The design-space of this template is also huge: Starting with data types of different precision per operation to a variety of unrolling of loop parallelisms and pipelining options. Example frameworks that generate this type of accelerators are hls4ml [4], Haddoc2 [6], and FINN [7].

Both architecture "templates" are well justified for different reasons. The streaming template is best used for DNNs that require high throughput and/or low latency. The engine-type accelerators are better for large DNNs in latency-relaxed environments or if there are not enough FPGA resources to implement the streaming-type.

2.2 Combining Streaming-Type and Engine-Type

In this section, we advocate for the deployment of mixed architectures built upon a combination of specialized streaming- and engine-type accelerators. To support our proposal, we refer to the roofline analysis of a subset of the well-known *LeNet* topology. This roofline is depicted in Figure 2. The vertical dotted lines show the operational intensity (OI) of three internal layers (convolution, pooling, dense layer) implemented with streaming-type (red) and engine-type (orange) of accelerators. The attainable performance of these layers is capped by the FPGA I/O (green), DRAM (red), BRAM (blue), and LUTRAM (orange) bandwidths, and the DSP GFLOPS/s (magenta) lines. For each accelerator type, the relevant caps differ, since the engine-type needs to read weights from the DRAM and inference requests from I/O, while the streaming-type only needs to read inference requests from I/O. As can be seen in Figure 2, the left-hand dense operation implementation has an OI of 1.5, while the right-hand 2D convolution operation has an OI of roughly 50. In this case, the dense operation in an engine-type accelerator would be heavily limited by the network or DRAM bandwidth, while the streaming-type accelerator would achieve near-optimal performance. In contrast, both architecture-types of the two 2D convolution layers are compute-bound and could be implemented by either stream or engine architectures, depending on the intended resource footprint.

Following this path, it would make sense to create an accelerator where the first layers are executed on an engine-type and the last dense layers on a streaming-type architecture. This would achieve the same performance as an all-streaming approach, but would save resources. On the contrary, the required bandwidth for data moving between layers within a DNN tends to decrease throughout the network topology. Consequently, one could argue for using a streaming-based architecture to sustain the high-bandwidth requirements of the first layers, and to finish with an engine-based architecture at the end. Both considerations to combine engine and streaming architectures have the potential to increase the efficiency of the resulting accelerator. Such a mixture of architectures can only be implemented in a flexible way within an FPGA, and it would be a pity for reconfigurable accelerators not to leverage this advantage over GPUs or TPUs/DPUs.

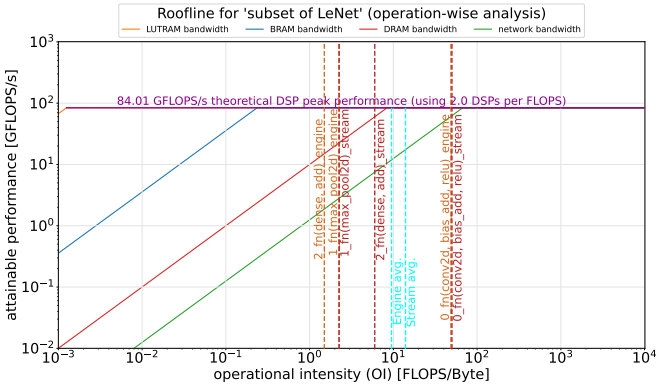


Fig. 2. Per-operation roofline analysis of a subset of LeNet.

2.3 Limitations of Existing DNN-to-FPGA Flows

With this work we explore if and how we could build a framework that can search for the best combination of accelerator types for a given DNN topology with specific performance and resource requirements. Instead of building such a framework from scratch, we decided to leverage the plentiful existing frameworks and tool flows [8], [9] to avoid re-inventing the wheel. Thus, we started to analyze existing DNN-to-FPGA tool flows and frameworks, and we noticed major weaknesses: First, we couldn't find a single holistic DNN example that was supported by a range of existing frameworks. Currently, despite the vast range of options for DNN-to-FPGA flows, all frameworks support only a limited range of DNN operations, or a very narrow set of target devices, or both [8]. While this is a hurdle when evaluating existing frameworks, it emphasizes the advantage of combining them. Second, there exists no guide to help the user select the right accelerator architecture template or framework for a given DNN, let alone a single solution or tool flow to execute arbitrary DNNs on a range of FPGAs. Most published frameworks are tailored to a specific network topology, problem domain, and target hardware [8], [9]. Very recently, the authors of [10] released a framework that can optimize different backend frameworks, but does not combine them or help the user in select the right framework for a given problem. Consequently, to choose a framework for DNN acceleration on FPGAs for a particular application, a user needs to navigate this maze of possibilities and the final decision currently depends mostly on the familiarity, or just pure awareness, of the user with the different solutions. Due to these hurdles, the generally difficult usability, and the poor tool support, FPGAs are not as popular as GPUs or domain-specific ASICs, despite their advantages (cf. [1]).

Therefore, to evaluate our proposal of combining different architectural templates, we characterized the performance and resource consumption of a selection of existing frameworks (cf. Table 1) and we built roofline models for each of them. To mix engine and streaming architectures, we realized that two more steps were necessary: First, we needed to identify a proper level of abstraction that enabled us to compare different solutions. Second, we had to create a way for different frameworks to interoperate. This analysis led to the concept of organic compilation and the definition of an operation set architecture, both of which will be presented in the next section.

3 Organic Compilation for FPGAs

We envision a holistic framework that is able to find the optimal mix of architectural templates in a particular DNN graph while considering a given set of resource, performance, or latency constraints. For example, when targeting a high throughput application that also requires high accuracy, the first convolutional layers of a DNN, like in Figure 2, could be implemented with a high-throughput streaming architecture from the open source

framework haddoc2 [6]. Subsequently, the last dense layers in this example can be mapped to the engine-type accelerator VTA [5]. Such a combination of different architectural templates for each layer increases the efficiency of the whole design, since the individual OIs of the layers are considered. In another situation, if the user has to solve a problem that requires extreme throughput, but could maintain its accuracy with binary weights, LogicNets would be the optimal solution [3]. After we studied the abundance of existing DNN-to-FPGA tools, each providing optimized implementations for specific areas within the overall design-space, we conceive this proposed holistic framework to be an "*organic compiler*": Such an organic compiler analyzes the DNN and performs the design-space exploration with consideration of both architectural templates, i.e. streaming and engine. On top of this, it also *recommends the best mix of existing frameworks* to implement all parts of the given DNN on FPGAs in the most efficient way.

3.1 Enabler: The Operation Set Architecture

In order to compare different mixtures of architecture-templates and to combine them into optimal solutions, we propose the operation set architecture (OSA) principle. The levels of abstraction of these operation sets are selected such that it allows the compiler to make meaningful comparisons of performance and resource metrics between totally different implementations. With respect to the intermediate representation (IR) snippet of an abstract syntax tree (AST) in Figure 1 ①, the `conv2d`, `relu`, and `max_pool2d` constitute components of such operation sets. In essence, the operation sets are groups of those components. We selected this level of abstraction after we revisited the long list of specialized frameworks, existing domain-specific languages (DSLs), IRs, and optimization techniques of existing tool flows for DNNs. We observed that compilers of frameworks that target engine-type accelerators apply the heaviest optimizations. This results from the fact that DNNs are compiled into engine-type instructions, for which well known optimization methods from classical CPU and GPU compilers can be used. Such optimizations include constant folding, dead code elimination, operator fusion, elimination of common sub-expressions, simplify paddings, and simplify the data flow graph for inference. Surprisingly, frameworks targeting streaming-type accelerators perform either no or only a few hardware-specific optimizations and even leave constant folding to the synthesis tools in most cases.

Therefore, we concluded that the level of abstraction for comparing different implementations should be chosen such that the decision between engine-type and streaming-type accelerators happens after these »basic« optimizations but before the program gets lowered further, since many of these optimizations would help the streaming-type accelerators, too. Following this path, the compiler can optimize a DNN graph above or at this level of abstraction (cf. ① in Figure 1), detached from the lower level details of the execution of one operation. This level is similar to the abstraction levels used by popular DSLs like RelayIR [5] or ONNX [11]. Only after that the compiler decides between engine and streaming implementations of each operation. Consequently, operations that are chosen to run on an engine are lowered into engine-specific instructions ②a and operations that are selected to be implemented on a streaming architecture are synthesized as parameterized IP blocks ②b. From an AST ① point of view, all operations are guaranteed to be "executed" by the available specialist frameworks, illustrated with the arrows ③a and ③b. How the operations will be "executed" in detail is up to the different frameworks (cf. ②a and ②b). Thus, each of these frameworks supports different a *set of operations* through its chosen target architectures (cf. ④a and ④b). However, the AST can be optimized and transformed without considering these details. In the example of Figure 1, the first two higher-level operations, `conv2d` and `relu`, are executed on an engine-type accelerator, while the following two operations, `max_pool2d` and `conv2d`, are then "executed" by implementing IP cores in a streaming way. The method ① — ④ is what we refer to as *»Operation Set Architecture«*, because it provides a meaningful unified abstraction level and utilities to compare and optimize totally different implementations of similar sets of operations.

3.2 Organic Compilation Framework

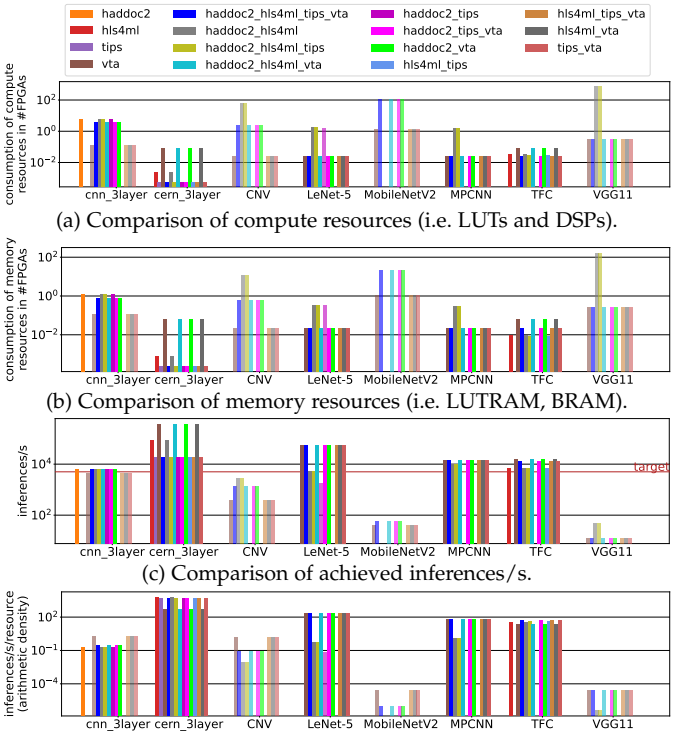
Starting from an application description using the OSA, we want to reuse the large body of existing frameworks and their highly optimized implementations as plugins for one holistic compiler. This way, we can reuse and combine specific parts of the various existing frameworks that are already generating highly optimized, thought-through, and high-performance implementations for sub-sets of DNN operations. This ecosystem should eventually converge to an "organic compiler", capable of generating an optimized implementation of the full set of operations required for DNNs in a fully automated manner. To be practical and autonomous, an organic compiler must be able to compare all the different architectural templates and concrete implementations offered by the available specialist frameworks. Therefore, we need a *unified criterion*, such as FLOPS or iterations-per-second, to perform this comparison. Next, we require one interface to blend — or to mix the cocktail of — various solutions towards one optimal implementation. This includes a *mechanism to interface* with all kinds of different specialist frameworks in order to reuse them and not be forced to "re-invent the wheel". These interfaces must allow to predict the performance of a particular operation, to generate the implementation of it, and to derive the necessary "glue-logic" to connect all the different operation implementations automatically. Therefore, the prediction of the performance and resource consumption of a particular operation implementation is delivered by the design-space exploration (DSE) of each framework individually using these interfaces. Consequently, the OSA allows to combine the different DSE mechanisms of the individual frameworks into one holistic model. In addition, the analysis provided by the OSA would allow a precise calculation of the communication costs among the different operation set implementations.

4 Evaluation

The idea of the "organic compiler" builds on two fundamental assumptions: First, the expectation that one way to seamlessly combine different research will benefit the whole AI-on-FPGA ecosystem compared to constantly reinventing the wheel. Second, we expect that each specialist framework is better than others in one aspect, hence combining them should lead to more efficient results. While the first assumption is optimistic but difficult to measure, the second one can be evaluated with some micro benchmarks.

Therefore, we selected well-known open source frameworks from the community and created combined designs to see if the combination of specialist frameworks does lead to superior results. The details of the used frameworks and their architectural style are shown in Table 1. The model zoo used for the evaluation is given in Table 2. We evaluated the employed frameworks and built a roofline-like model for each framework to estimate the resource usage and the performance for each layer of the model zoo. Afterwards, we used these models to calculate the resource consumption, throughput, and efficiency of different combinations. The results are shown in Figure 3.

Note that all the created designs were guided to produce a throughput of 5,000 inferences-per-second, but some of them were unable to achieve that target, as shown in Figure 3c. This number was chosen to serve low-latency high-throughput applications [3]. Also, not all frameworks were able to achieve this target with realistic resource consumption, e.g. Figures 3a



(a) Comparison of compute resources (i.e. LUTs and DSPs).

(b) Comparison of memory resources (i.e. LUTRAM, BRAM).

(c) Comparison of achieved inferences/s.

(d) Comparison of arithmetic density (i.e. efficiency).

Fig. 3. Comparison of different framework combinations. Each bar represents a combination of frameworks (see Table 1). Please note that not all frameworks can implement all test cases. FPGA resources are based on a Xilinx `VU37P`.

TABLE 1
DNN-to-FPGA frameworks used for evaluation

Name	Architecture type	Source
haddoc2	**Streaming**, impl. in VHDL	[6]
hls4ml	**Streaming**, impl. in Vivado HLS	[4]
VTA	**Engine**, impl. in Vivado HLS, fixed architecture for all used test cases	[5]
TIPS	**Engine**, impl. in Vivado HLS, size of matrix-vector unit adapted for each test case	implemented by the authors

and 3b show that a streaming-architecture for `VGG11` with 8bit weights would require 100+ Xilinx `VU37P` FPGAs. This case underlines the sometimes extreme resource consumption of streaming-type architectures. However, Figure 3d also shows that the combination of different specialist frameworks and architectural styles is indeed more efficient, i.e. it can achieve a higher throughput-per-resource. In general, we consider this efficiency as the main metric to evaluate the concept of organic compilation, since i) all generated combinations are optimized towards the user-provided throughput goal (e.g., 5,000 iterations-per-second) and ii) the advantage provided by the combination of different frameworks using the OSA is a higher efficiency and not necessarily a higher performance. For example, the mix of one streaming architecture (haddoc2) with one engine unit (VTA or TIPS) leads to the highest arithmetic densities in the `cnn_3layer` test case. Also, this combination delivers the best performance for the `MobileNetV2` (see Figure3c). Similarly, the combination of hls4ml and TIPS results in the highest arithmetic densities for the `cern_3layer` and `TFC` examples and can deliver up to 2.5 times more inferences-per-second-per-resource. Interestingly, for the `TFC` case, the combination of two engines (TIPS and VTA) is more efficient than just using VTA, since there must be multiple "cores" of these engines to meet the high performance target. Furthermore, the combination of different streaming architectures is beneficial, as can be seen in the `LeNet-5` case: The haddoc2 framework cannot implement the dense layers, and hls4ml cannot implement large convolutions, but combined they can meet the throughput requirement. However, this example also shows that for the small `LeNet-5` the engine VTA alone can achieve enough throughput and is the most efficient. As a summary, combining different specialist frameworks enables the implementation of DNN topologies that single frameworks cannot implement and has the potential to increase the efficiency of those solutions by more than one order of magnitude, as can be seen by the higher arithmetic density for combined solutions for the examples `cnn_3layer`, `cern_3layer`, `MobileNetV2` and `TFC`.

TABLE 2
Model zoo for evaluation (all using 8bit weights)

Task	Network topology	# Conv.	# Dense	Parameter (KB)
Jet Tagging	CERN 3 layer	0	4	4.4
Hand Gestures	MPCNN	3	2	70.3
MNIST	TFC	0	4	59.2
	LeNet-5	3	2	32.6
CIFAR-10	3 layer (only conv. part)	3	0	90.6
	CNV	6	3	1,544.8
CIFAR-100	VGG11	8	3	129,176.0

5 Conclusion

Reconfigurable hardware including FPGAs is ideally suited to address the demanding and diverse performance, latency, and energy efficiency requirements of AI workloads. The adoption of FPGAs for AI workloads is, however, limited by the narrow solution space covered by the many existing DNN-to-FPGA compilation frameworks. We propose and evaluate the operation set architecture, which provides an optimal level of abstraction to map the operations found in a DNN to FPGAs, using a combination of architectural templates provided by existing compilation frameworks. The combination of existing frameworks and their architectural templates allows us to cover the union of their solution spaces. The intersections of the solution spaces enable us to choose the best available template for each DNN layer, which improves the efficiency, throughput, and arithmetic density of the generated designs by more than 10x. Based on the encouraging results from our evaluation of the operation set architecture, the next step is to develop an end-to-end organic compiler, where existing and future architectural templates can be easily plugged in. We are convinced that optimizing the best mix of architectural templates for every specific mix of operations can also be leveraged for different applications, like HPC, and for different SoC architectures, such as CGRAs and Xilinx' latest ACAP devices. The presented organic compiler concept could lead to a growing ecosystem that can reuse a rich variety of existing research in the community.

References

[1] S. Hooker, "The hardware lottery," *Commun. ACM*, Nov. 2021. DOI: 10.1145/3467017.

[2] J. Fowers *et al.*, "A configurable cloud-Scale DNN processor for real-Time AI," *Proceedings - International Symposium on Computer Architecture*, 2018. DOI: 10.1109/ISCA.2018.00012.

[3] Y. Umuroglu *et al.*, "LogicNets: Co-Designed Neural Networks and Circuits for Extreme-Throughput Applications," in *Proceedings of the 30th IEEE International Conference on Field-Programmable Logic and Applications (FPL)*, Gothenburg, Sweden: IEEE, 2020. DOI: 10.1109/FPL50879.2020.00055.

[4] J. Duarte *et al.*, "Fast inference of deep neural networks in FPGAs for particle physics," *Journal of Instrumentation*, 2018. DOI: 10.1088/1748-0221/13/07/P07027.

[5] T. Moreau *et al.*, "A hardware–software blueprint for flexible deep learning specialization," *IEEE Micro*, Sep. 2019. DOI: 10.1109/MM.2019.2928962.

[6] K. Abdelouahab *et al.*, "Tactics to directly map cnn graphs on embedded fpgas," *IEEE Embedded Systems Letters*, Dec. 2017. DOI: 10.1109/LES.2017.2743247.

[7] M. Blott *et al.*, "FINN-R: An End-to-End Deep-Learning Framework for Fast Exploration of Quantized Neural Networks," *ACM Trans. Reconfigurable Technol. Syst.*, Dec. 2018. DOI: 10.1145/3242897.

[8] S. I. Venieris *et al.*, "Toolflows for mapping convolutional neural networks on fpgas: A survey and future directions," *ACM Comput. Surv.*, Jun. 2018. DOI: 10.1145/3186332.

[9] K. Guo *et al.*, "[DL] A survey of fpga-based neural network inference accelerators," *ACM Trans. Reconfigurable Technol. Syst.*, 2019. DOI: 10.1145/3289185.

[10] A. Montgomerie-Corcoran *et al.*, "SAMO: Optimised Mapping of Convolutional Neural Networks to Streaming Architectures," in *Proceedings of the 32st IEEE International Conference on Field-Programmable Logic and Applications (FPL)*, Belfast, United Kingdom: IEEE, 2022. DOI: 10.1109/FPL57034.2022.00069.

[11] The ONNX community. (2022). "Open Neural Network Exchange Intermediate Representation (ONNX IR) Specification." https://github.com/onnx/onnx/blob/main/docs/IR.md, visited on 2022-03-18.