MACHINE
LEARNING
Science and Technology

CrossMark

OPEN ACCESS

RECEIVED
17 July 2023

REVISED
2 November 2023

ACCEPTED FOR PUBLICATION
15 November 2023

PUBLISHED
24 November 2023

PAPER

Exploring machine learning to hardware implementations for large data rate x-ray instrumentation

Mohammad Mehdi Rahimifar[*], **Quentin Wingering**, **Berthié Gouin-Ferland**, **Hamza Ezzaoui Rahali**, **Charles-Étienne Granger** and **Audrey C Therrien**

Interdisciplinary Institute for Technological Innovation-3IT, Sherbrooke (Québec), Canada
[*] Author to whom any correspondence should be addressed.

E-mail: mohammad.mehdi.rahimifar@usherbrooke.ca

Keywords: machine learning, neural networks, FPGA, instrumentation, data acquisition

Abstract

Over the past decade, innovations in radiation and photonic detectors considerably improved their resolution, pixel density, sensitivity, and sampling rate, which all contribute to increased data generation rates. This huge data increases the amount of storage required, as well as the cabling between the source and the storage units. To overcome this problem, edge machine learning (EdgeML) proposes to move computation units near the detectors, utilizing machine learning (ML) models to emulate non-linear mathematical relationships between detector's output data. ML algorithms can be implemented in digital circuits, such as application-specific integrated circuits and field-programmable gate arrays, which support both parallelization and pipelining. EdgeML has both the benefits of edge computing and ML models to compress data near the detectors. This paper explores the currently available tool-flows designed to translate software ML algorithms to digital circuits near the edge. The main focus is on tool-flows that provide a diverse range of supported models, optimization techniques, and compression methods. We compare their accessibility, performance, and ease of use, and compare them for two high data-rate instrumentation applications: (1) CookieBox, and (2) billion-pixel camera.

1. Introduction

New instrumentation detectors have better sensitivity, sampling rate, and pixel density. These improvements significantly increase the total data velocity, exceeding terabytes per second (TB s^{-1}) in particle physics and medical imaging experiments and surpassing the capacity of current acquisition systems [1, 2]. For example, the data generation of Large Hadron Collider (LHC) experiments at CERN reach 1200 GB s^{-1} [3]. The detectors at the LHC use multi-level trigger systems and still need massive data centers to analyze, compress and save the final data. The current methods save only a small fraction of the total generated data, recording only 1 in 10 to 1 in 100 bunch crossings happening at 40 MHz [4, 5]. Another example is the LINAC Coherent Light Source (LCLS) at the Stanford Linear Accelerator Center (SLAC) National Accelerator Laboratory, which has a repetition rate of 1 MHz leading to colossal data velocity, exceeding TB s^{-1} [6].

The current paradigm of collecting all raw data in a centralized node requires expensive hardware, significant power, and has a large environmental impact. A potential solution is to move the computational units closer to the edge of the system, either in adjacent hardware or directly embedded within the detector control and acquisition circuits, a method called edge computing [7]. By placing computing resources and data storage at the system's edge, edge computing reduces system latency while enabling real-time analytic and reducing operational costs [8, 9].

Edge computing allows only limited computing resources due to power and physical constraints, which may not be enough when it comes to analyses requiring complex classical algorithms. Training machine learning (ML) algorithms to model these complex algorithms can achieve the same behavior with less computing complexity. Combining edge computing and ML is called edge ML (EdgeML). EdgeML offers

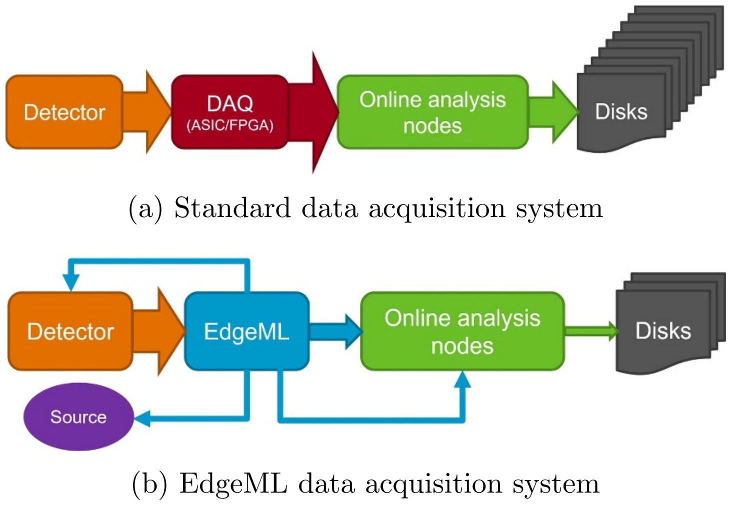


Figure 1. Comparison of classic data acquisition system and EdgeML data acquisition system.

reduced latency, and bandwidth requirements. Figure 1 presents where the EdgeML fits in an edge computing paradigm and how it differs from a standard data acquisition (DAQ) [10].

As can be seen, ML on the edge can provide intelligent, low-latency feedback to the detector and the radiation source, enabling parameter adjustments. The amount of stored data in the EdgeML DAQ paradigm is also significantly lower compared to standard DAQ. ML algorithms can be implemented in hardware to increase pipelining and parallelization. While several processing units and digital circuits are available for general tasks and ML purposes, only a few are suitable for use near the edge.

Choosing the correct processing units for EdgeML applications is a crucial subject due to the necessity of very low latency for real-time DAQ. Moreover, low power consumption, low inference latency, low unit cost, and high integration level are the next crucial factors for a good EdgeML system. Algorithms can be implemented in hardware to increase pipelining and parallelization [11]. Application-specific integrated circuits (ASICs) and field-programmable gate arrays (FPGAs) are two integrated circuits that can run with a lower latency compared to other micro-controllers, and common processors such as central processing unit, and graphics processing unit [12]. ASICs and FPGAs also achieve inherent parallelism through their optimized architecture, allowing for the execution of multiple tasks or operations simultaneously. Moreover, low power usage, and large number of input/output (I/O) ports for high-throughput communication also make ASICs, and FPGAs two excellent choice for EdgeML applications [13]. The architecture of ML algorithms and these digital circuits are a great match since ML algorithms generally use arithmetic that is simple for digital circuits to execute, such as additions and multiplications [14]. However, ML algorithms are progressing rapidly, and ASIC require long development cycles, making FPGA a better choice for prototyping and low cost EdgeML applications.

Generally, there is ongoing research to integrate FPGA-based EdgeML models in high data-rate instrumentation, particularly for online event selection, and DAQ paradigms at the edge of the system [15]. In [16], an FPGA-based ML event classification for custom electronics-based trigger systems in high energy physics is introduced, where the lowest latency for real-time event classification is required. Additionally, the authors in [17] have presented an FPGA-embedded system for ML-based tracking and triggering in the electron–ion collider experiment. Moreover, a fast muon tracking with ML implemented in FPGA for first-level trigger at LHC experiment is presented in [18].

Although FPGAs are great choice for high data-rate instrumentation, implementing ML models on FPGAs requires a high level of expertise and knowledge in hardware design. In this paper, we first explore the available tool-flows for mapping ML algorithms onto FPGAs, near the edge of the sensors. The main focus is on tool-flows that provide a wide range of supported models, optimization techniques, and lower latency to find the most suitable for instrumentation. After finding the most suitable one, we use it to implement different ML models on FPGAs focused on two high data-rate instrumentation application: (1) CookieBox, and (2) billion-pixel camera. We design our ML models, translate them to hardware code, and implement the models for both application on the Zynq UltraScale+ MPSoC ZCU104 evaluation kit as the target board.

The rest of the paper is organized as follows: section 2 compares the current methods for implementing ML on FPGA and explains the available ML to FPGA tool-flows. Section 3 describes the methodology, and the experiment setup that we use to compare the tool-flow performances on FPGA. The simulation and hardware implementation results are presented in section 4. Finally, we discuss the performance of available tool-flows for instrumentation and conclude the paper in sections 5 and 6, respectively.

2. Background

The usual FPGA programming languages are hardware description language (HDL) such as Verilog and, very high-speed integrated circuit hardware description language (VHDL). ML model hardware implementations in real-time systems are complicated, multi-step endeavors. Translating a complex ML algorithm to HDL requires sufficient hardware knowledge and is time-consuming.

High-level synthesis (HLS) is a relatively new alternative for developing FPGA applications. HLS allows software engineers to design applications for FPGA and ASIC platforms using more common programming languages, namely C and C++ [19]. HLS tools can be used to implement ML algorithms with much less development time and effort, but they may bring some compromise on performance compared to direct HDL implementation [20, 21]. Several groups want to take automation a step further and create a tool-flow that does the entire implementation chain from inference model to hardware implementation. These tool-flows use an ML model from common libraries such as *TensorFlow*, *Keras*, and *PyTorch*, enabling FPGA implementation with no hardware knowledge. Table 1 provides a thorough compilation of the presently accessible tool-flows for ML to FPGA applications. It includes information about the supported models, ML environment, release date, and the supported ML model compression methods. The following paragraphs highlight a few of these tool-flows.

HLS4ML [22, 46], is an open-source Python package for ML inference in FPGAs. It first converts a *Keras* or *PyTorch* model to an HLS model and maps it to the corresponding HDL code. HLS4ML was initially designed for microsecond latency applications like the CERN LHC [47]. The HLS code generated by hls4ml can also be used for ASIC design [48, 49]. HLS4ML offers many configuration settings such as I/O type, reuse factor, precision, and different implementation strategies. The reuse factor is a parameter of HLS4ML that determines how many times each FPGA multiplier will be used, directly impacting the model latency.

FINN [23] is a framework for building fast and flexible FPGA accelerators using a heterogeneous streaming architecture. The FINN framework targets binarized neural networks (BNNs) and highly quantized neural networks (NNs) for small boards. FINN converts each layer to an HLS design, and subsequently stitches these sub-components together to make the whole network. Custom models can also be imported from an Open Neural Network Exchange (ONNX) model by calling FINN from a Python script. Compared to HLS4ML, FINN offers less customization and we can only change the target clock, target throughput, and quantization.

It is worth mentioning that the HLS4ML and FINN teams are working together on a more unified test flow so that both *Keras* and *PyTorch* models can be translated into quantized ONNX, as shown in figure 2 [50]. This will make the interaction of HLS4ML, and FINN much easier in the future.

The Vitis AI [24] platform is a comprehensive artificial intelligence (AI) inference development solution for Xilinx devices and Alveo Data Center acceleration cards. Vitis AI is a proprietary configurable intellectual property (IP) core with internal parallelism [48]. Some commonly used models supported by Vitis AI are provided in the Xilinx Model Zoo [51] such as ImageNet networks and some object detection networks. Vitis AI also supports custom models, and users can give it their customized NN model.

Versatile tensor accelerator (VTA) [25] is an open, generic, and customizable deep learning accelerator with a complete Apache tensor virtual machine-based compiler stack. Generally, A VTA instance consists of a vector-matrix and an arithmetic logic unit core, supporting operations on matrix operands. VTA targets architectures similar to ResNet and MobileNet-based NN architectures.

MATLAB deep learning processor (DLP) [26] is a subset of the commercial MATLAB suite and a tool-flow supporting a full ML model compilation, including quantization. It can target any platform compatible with the Matlab HDL Coder [52], such as Xilinx's Zynq and Zynq UltraScale+ platforms. MATLAB DLP has its own front end, but it also can import NN models from currently available libraries such as *PyTorch*. It also supports ONNX, making it inter-operable with other NN libraries and other tool-flows.

OpenVINO provides a set of tools and libraries for optimizing and deploying deep learning models on various Intel hardware platforms, including FPGAs [27]. Although OpenVINO is not designed for FPGAs, it has the same functionality as MATLAB DLP and Vitis AI for FPGA accelerators. It provides boosted deep learning performance for vision, audio, and more models from popular frameworks like *TensorFlow* and

Table 1. An overview of the available ML to FPGA tool-flows.

Tool-flow	ML models	Compression methods	Active	ML environment	Development	Release date
HLS4ML [22]	FCNN[a], CNN[b], RNN[c]	Pruning, quantization	Yes	Keras	Open-source	2020
FINN [23]	FCNN, CNN	Pruning, quantization	Yes	PyTorch	Open-source	2017
Vitis AI [24]	CNN, RNN	Pruning, quantization	Yes	TensorFlow, PyTorch	Commercial	2019
VTA [25]	CNN	Pruning, quantization	Yes	PyTorch	Commercial	2018
Matlab DLP [26]	CNN	Pruning, quantization	Yes	Keras, Caffe	Commercial	2015
OpenVino[d] [27]	CNN	Pruning, quantization	Yes	TensorFlow, PyTorch	Commercial	2019
OpenHLS [28]	CNN	Pruning, quantization	Yes	PyTorch	Open-source	2023
NNGEN [29]	FCNN	Quantization	No	Proprietary	Open-source	2019
ScaleHLS [30]	FCNN	—	No	PyTorch	Open-source	2022
CFU Playground [31]	CNN	—	Yes	TensorFlow	Open-source	2021
VeriGOOD-ML [32]	CNN	—	Yes	ONNX	Open-source	2022
DNNWeaver [33]	CNN	—	No	Proprietary	Open-source	2016
DL2HDL [34]	FCNN	—	No	PyTorch	Open-source	2019
FPGAConvnet [35]	CNN	—	No	Proprietary	Open-source	2022
FINN-L [36]	LSTM	—	No	PyTorch	Open-source	2017
LeFlow [37]	FCNN,CNN	—	No	Proprietary	Open-source	2018
CaFGPA [38]	CNN	—	No	Caffe	Open-source	2018
DNN Builder [39]	CNN	—	No	Caffe	Open-source	2017
FP-DNN [40]	CNN	—	No	TensorFlow	Open-source	2017
Snowflake [41]	CNN	—	No	Proprietary	Open-source	2017
FFTCodeGen [42]	CNN	—	No	Proprietary	Open-source	2016
Haddoc2 [43]	CNN	—	No	Caffe	Open-source	2017
Angel-Eye [44]	CNN	—	No	Caffe	Open-source	2017
Caffein [45]	CNN	—	No	Caffe	Open-source	2018

[a] Fully connected neural network
[b] Convolutional neural network
[c] Recurrent neural network
[d] OpenVino is the only tool-flow that targets Intel FPGAs. Other tool-flows target AMD Xilinx FPGAs.

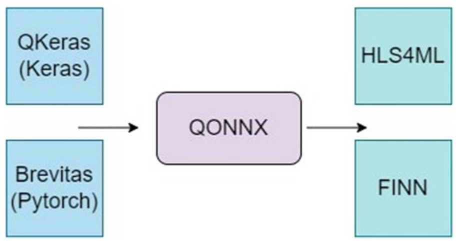


Figure 2. HLS4ML and FINN common interface.

PyTorch. It also supports different quantization and optimization techniques but only supports a limited number of Intel FPGA boards.

OpenHLS is a lightweight, compiler framework that uses a combination of compiler and HLS techniques to compile the entire deep NN into fully scheduled register-transfer level design [28]. Its architecture is

Table 2. Neural network characteristics for testing tool-flows.

Architecture	FCNN	CNN
Dataset	UNSW-NB15	SVHN
Size (Params.)	3171	4460
Activation	Categorical cross-entropy	Categorical cross-entropy
Optimizer	Adam	Adam
Learning rate	0.001	0.001
Validation split	0.2	0.2
Batch size	256	64

similar to HLS4ML and FINN, but focused on Convolutional NNs (CNNs) in particular, while using low level virtual machine as its core compiler [53].

The tool flows summarized above are the most actively developed ones, based on the level of activity on their GitHub repositories, and offer better configuration and optimization support than other low activity tools. Other available tool-flows have relatively less community support, with fewer features compared to the first seven tool-flows of table 1. Among the tool-flows explained, only HLS4ML and FINN fully support both fully connected NN (FCNN) and CNN layers with no board support limitation. Therefore, we see potential in these tool-flows to implement fully customized models on FPGAs. Additionally, researchers can migrate from FPGA to ASIC for fixed applications if the tool-flow provides HLS/HDL codes, which HLS4ML and FINN do. Therefore, HLS4ML and FINN are chosen as potential candidates for EdgeML high data-rate instrumentation.

In the rest of this paper we investigate HLS4ML, and FINN, compare them, and explore their different configuration settings to find the optimal configuration. Our objective is to find the optimal tool-flow considering the latency and use the best one for two high data-rate instrumentation applications: (1) CookieBox and (2) billion-pixel camera. We design and translate ML models to hardware code, and implement the models for both application on the Zynq UltraScale+ MPSoC ZCU104 evaluation kit as the target board.

3. Methodology

3.1. Tool-flows comparison

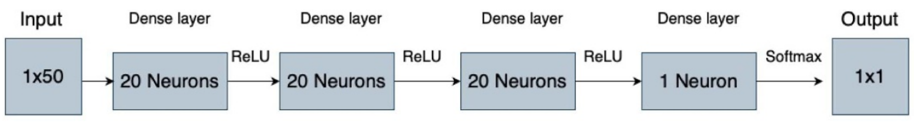
We first selected two different training data sets, and designed NN models for these applications to ensure the fit on our board. For each application we designed an FCNN model and an CNN model.

The first model (FCNN) has three hidden layers and a total size of 3171 parameters. The training data set for the FCNN model is the *UNSW-NB15*, a big data set created to provide a comprehensive network-based data set that can reflect modern network traffic scenarios [54].

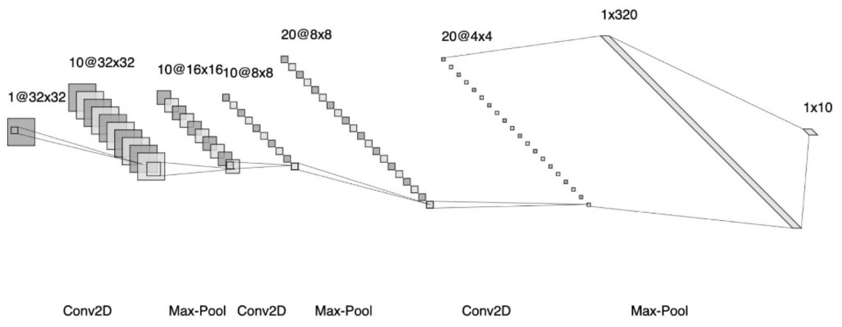
The second model (CNN) has three convolutional layers, two dense layers, and a total size of 4460 parameters. The training data set for the CNN model is the Street View House Number (*SVHN*) data set, which can be seen as similar flavor to MNIST with over 600 000 labeled data [55].

Table 2 provides an overview of the key characteristics of both NN models for testing tool-flows. Both model's architecture are presented in figure 3. Since FINN is only focused on highly quantized models, we also used the *Qkeras* in the HLS4ML front-end to quantize the model for a better side-by-side comparison with FINN. Unfortunately, due to a software limitation in Vivado, we were unable to compare the FCNN and CNN models using the same datasets in the first experiment. Vivado restricts the number of parameters per layer to 4096, which would be exceeded if we were to implement an FCNN model for the SVHN dataset, due to the large input shape.

Once we have selected the NN models and use HLS4ML and FINN to generate their corresponding HLS code, we extract the corresponding IP block of the model, bring it into the Vivado design suite, and finalize the final block design to test and implement on the board. The overall design flow is presented in figure 4. In the final block design, it is crucial to appropriately connect the NN block to various components. These components include a memory controller, a processor, and a counter. The counter counts the number of clock cycles that it takes to complete the inference of the ML block, which indicates the latency of the ML block. The latency has no variability and depends only on architecture of the trained model. We use the advanced extensible interface (AXI) developed by ARM for communication bus protocol in the block design. We chose the Zynq UltraScale+ MPSoC ZCU104 evaluation kit as target boards, since it has an ARM processor, and sufficient resources for our ML models. The processor is not necessary but facilitates the testing process. We created our own ZCU104 block design for HLS4ML, since is not fully supported by



(a) FCNN



(b) CNN

Figure 3. FCNN and CNN models architecture for testing tool-flows.

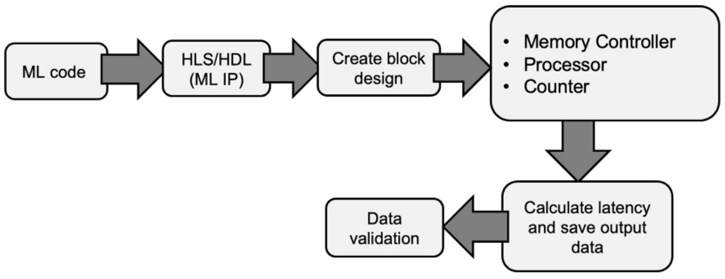


Figure 4. The test flow for evaluating the Tool-Flows.

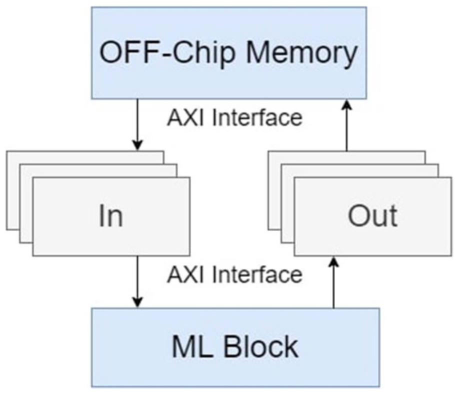


Figure 5. Data transmission flow.

HLS4ML's end-to-end examples [56]. To send the data in and out of the NN block on the board, we use an off-chip memory to send the data with the AXI. This flow is presented in figure 5.

The NN models implemented by HLS4ML were trained on a PC using *Keras* library, since *Keras* is fully supported as HLS4ML front-end. The same NN models are implemented by FINN using *PyTorch* since FINN does not support *Keras* at this time.

Table 3. Neural network characteristic for testing instrumentation applications.

Application	CookieBox		Billion-pixel camera	
Architecture	FCNN	CNN	FCNN	CNN
Size (Params.)	3433	3665	2593	2185
Loss function	SCC[a]	SCC	MSE[b]	MSE
Optimizer	Adam	Adam	Adam	Adam
Learning rate	0.001	0.001	0.0001	0.0001
Validation	0.2	0.2	0.15	0.15
Batch size	256	2042	100	100

[a] Sparse categorical cross-entropy (SCC)

[b] Mean square error + regularization term

3.2. Instrumentation applications

After comparing HLS4ML and FINN using the mentioned models, and selecting the best one with the lowest latency, we now apply a similar process for two real high data-rate instrumentation applications: the CookieBox and the billion-pixel camera [6, 57]. The objective of this experiment is to determine whether the performance of tool-flows is sufficient for these applications. We again compare both FCNN and CNN models to find the optimal ML configuration.

The first application is the CookieBox, which is an angular streaking detector for online x-ray beam diagnostic tool in the LCLS-II project by SLAC [6, 58]. LCLS-II operates at a repetition rate of 1 MHz, resulting in a massive amount of data exceeding terabytes per second. To handle this data overload, a strategy is employed to veto some certain pulses. The CookieBox detector is actually used to make these veto decisions. The designed ML model's purpose for CookieBox is to classify different x-ray beam shots, and veto the unnecessary ones [59].

The second application is the billion-pixel camera, an x-ray camera for synchrotron and x-ray free-electron laser experiments. The Billion-pixel camera will generate 1000 to 10 000 images in one second, and each image is around 1–2 GB in size. Accordingly, the billion-pixel camera will generate between 1 TB s^{-1} and 10+ TB s^{-1} of data [60]. The designed ML model's purpose for the billion-pixel camera is to compress input images by reducing sparse representations of the camera's images, followed by quantization and entropy coding for data compression [61]. It is worth mentioning that we needed to add a custom layer to the HLS4ML back-end for the billion-pixel camera experiment. We had to use a parametric soft shrink activation for the billion-pixel camera ML model, which is not supported by HLS4ML default models. The softshrink activation is defined as:

$$\text{Softshrink}_{\lambda}(x) = \begin{cases} x - \lambda, & \text{if } x > \lambda \\ x + \lambda, & \text{if } x < -\lambda \\ 0, & \text{otherwise.} \end{cases} \quad (1)$$

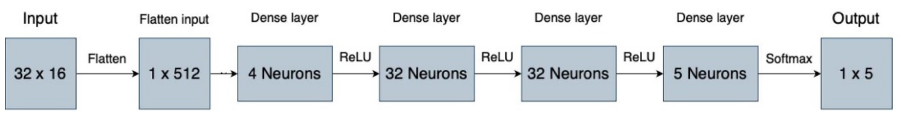
The softshrink activation helps increase the code sparsity, which we measure as the number of zero-value elements in the encoding divided by the total element count. Since the function is not a supported HLS4ML layer, we modified the back-end accordingly in order to convert the model. The NN model training setup for both applications is presented in table 3. The architecture of both models are also presented in figures 6 and 7, respectively.

4. Results

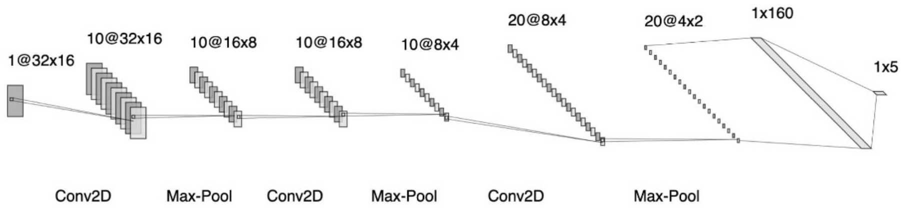
4.1. Tool-flow comparison

In the first experiment, our objective was to determine the tool-flow with the best latency for high data-rate instrumentation. We first use the high throughput configuration of the FINN, which focuses on the lowest latency (highest throughput). The configuration of HLS4ML for the first experiment is also set to a reuse factor of 1, a latency strategy, and a parallel data structure.

For the second experiment, we use the base configuration of the FINN focused on minimizing resource usage. The configuration of HLS4ML for the second experiment is set to a reuse factor of 64, a resource strategy, and a stream data structure. Finally, the target clock frequency for all experiments is set to 100 MHz.

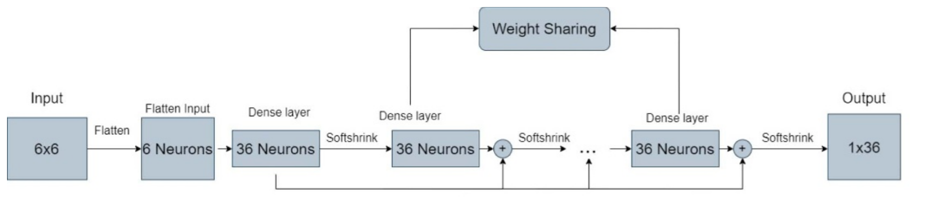


(a) FCNN

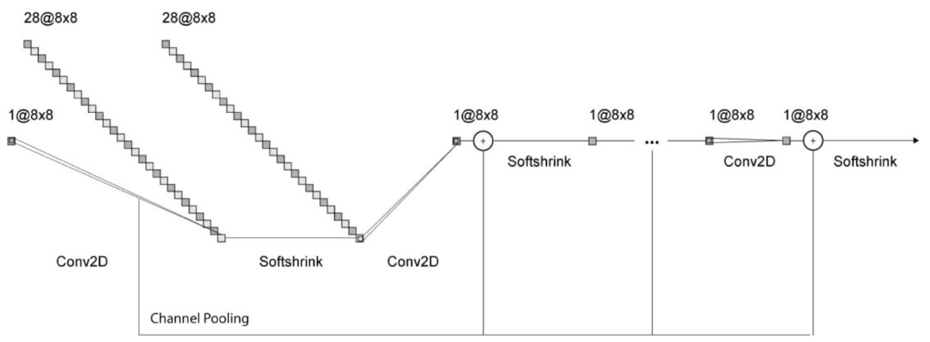


(b) CNN

Figure 6. CookieBox ML models architecture.



(a) FCNN model. The second dense layer and the softshrink function are repeatedly applied to the input. We represent this by three dots in the figure. The total number of repetitions is fixed to 5.



(b) CNN model. Similiar to the FCNN, the convolution and the softshrink are applied repeatedly, 5 times in total.

Figure 7. Billion-pixel camera ML models architecture.

The results for both experiments are shown in tables 4 and 5, respectively. We also compare the implementations of the mentioned models using HLS4ML and FINN with the available related works in both tables. We have chosen related works that concentrate on both latency and resource implementation to draw meaningful comparisons with our implementation. In table 4, we select models from other related works with the lowest latency for comparison. For a fair comparison in table 5, we opt for more compressed models from other related works, as we are emphasizing lower resource utilization in the second experiment.

As can be seen, the latency of HLS4ML for both experiments is lower compared to FINN, since the HLS4ML utilizes several HLS pragmas, such as loop unrolling in the final HLS code, resulting in low latency. By switching the implementation strategy from latency optimization to resource minimization in the HLS4ML configurations, resource usage decreases at the cost of increased latency. FINN performs better when it comes to resource utilization, especially digital signal processor (DSP) usage. FINN shows efficient power usage, as expected, due to its primary design targets, which are smaller boards and models. It is

Table 4. Implementation results for both fully connected (FCNN) and convolutional (CNN) models targeting a lower latency.

Tool-flow	HLS4ML (this work)		FINN (this work)		[22]	[48]	[49]	[62]
Dataset	UNSW-NB15	SVHN	UNSW-NB15	SVHN	SVHN	VoltageNet	Cityscapes	MNIST
Model	FCNN	CNN	FCNN	CNN	CNN	CNN	CNN	FCNN
Size (Params.)	3171	4460	3171	4460	170 000	81 000	14 000	118 000
Accuracy (%)	93	86	92	83	90	NA	77	93
Board	ZCU104	ZCU104	ZCU104	ZCU104	VU9P	Zedboard	ZCU102	VU19P
Clock (MHz)	100	100	100	100	200	100	140	200
Latency (μs)	4	45	9	72	6	110	4800	10
Power (w)	3.5	5.3	2.9	4.5	NA	NA	NA	NA
BRAM[a] (%)	0	23	1	28	3	93	56	70
DSP[b] (%)	1	16	0	2	95	100	60	38
LUT[c] (%)	3	8	7	31	15	48	46	20
FF[d] (%)	0	4	3	17	3	25	25	7

[a] Block RAM
[b] Digital signal processor
[c] Look-up table
[d] Flip-flops

Table 5. Implementation results for both fully connected (FCNN) and convolutional (CNN) models targeting a lower resource utilization.

Tool-flow	HLS4ML (this work)		FINN (this work)		[22]	[48]	[49]	[62]
Dataset	UNSW-NB15	SVHN	UNSW-NB15	SVHN	SVHN	VoltageNet	Cityscapes	MNIST
Model	FCNN	CNN	FCNN	CNN	CNN	CNN	CNN	FCNN
Size (Params.)	3171	4460	3171	4460	170 000	81 000	5300	118 000
Accuracy (%)	85	83	85	81	88	NA	81	93
Board	ZCU104	ZCU104	ZCU104	ZCU104	VU9P	Pynq-Z1	ZCU102	VU19P
Clock (MHz)	100	100	100	100	200	100	140	200
Latency (μs)	5.5	60	10	80	30	3300	4900	100
Power (W)	3.1	5.2	2.1	4.2	NA	NA	NA	NA
BRAM (%)	1	51	7	70	2	1.4	25	20
DSP (%)	0	12	0	2	20	0	18	1
LUT (%)	1	5	5	26	18	21	30	5
FF (%)	0	3	1	13	4	11	16	6

important to note that the power reported in the tables those reported by Xilinx Vivado post layout implementation. Both tool-flows demonstrate a good performance compared to the related works. Although [22] demonstrates a better latency, it uses higher clock frequency and a bigger board compared to the others, and also uses much more resources. The resource utilization of HLS4ML and FINN is also relatively better, considering our target is a smaller board (except than [48]). Additionally, it is worth noting that power usage data for the other works is not available for a direct comparison, and obtaining power values for the related works was not feasible.

In summary, HLS4ML outperforms FINN in terms of latency, which is the most crucial factor for high data-rate instrumentation applications, as previously mentioned. Consequently, we move forward with HLS4ML and test real instrumentation models with various configurations.

4.2. Instrumentation applications

First, we implemented an FCNN model for both the CookieBox and the billion-pixel camera, focusing on both latency and resource utilization in separate tests. The results are shown in table 6. In this table, the resource utilization percentage demonstrates the usage of each FPGA resource.

As mentioned earlier, the NN model used for the billion-pixel camera is larger than the one used for the CookieBox. This results in higher resource utilization and power usage. Although the FCNN model for the billion-pixel camera is larger, it has a smaller input shape. This leads to lower latency compared to the model designed for the CookieBox. By scrutinizing the waveform analysis in Vivado simulations, we noticed that most of the inference time is spent on the HLS4ML blocks trying to fetch the input data. Consequently, a

Table 6. Instrumentation experiment (CookieBox and billion-pixel camera) results with an FCNN model.

Strategy	Best latency		Resource optimal	
Model	CookieBox	Billion-pixel camera	CookieBox	Billion-pixel camera
Latency (μs)	5.70	0.89	10	1.01
Power (w)	4	5.9	3.7	3.8
BRAM (%)	1	1	5	2
DSP (%)	1	15	1	8
LUT (%)	39	38	20	19
FF (%)	2	7	6	4

Table 7. Instrumentation experiment (CookieBox and billion-pixel camera) results with a CNN model.

Strategy	Best latency		Resource optimal	
Model	CookieBox	Billion-pixel camera	CookieBox	Billion-pixel camera
Latency (μs)	21.14	94.12	150	120
Power (w)	4.8	6.2	4.5	4.2
BRAM (%)	30	2	35	3
DSP (%)	20	17	4	3
LUT (%)	45	40	24	31
FF (%)	8	19	9	18

smaller input data shape results in less latency. Moreover, the latency results with a resource implementation strategy are slightly higher, but we can achieve lower resource utilization, which is ideal for smaller boards.

We repeat both tests with a CNN model as presented in table 7. Like the FCNN results, the latency of the billion-pixel camera is lower compared to the CookieBox for the resource optimal case. However, the billion-pixel camera latency is higher for the best latency case. The main reason behind this is that due to the usage of a custom layer for the billion-pixel camera application, the latency strategy did not give usable results. Instead, we used the resource strategy with reuse factor 1. The bigger model of the billion-pixel camera causes a higher resource utilization. CNN models are usually big and difficult to fit on smaller boards. However, with the HLS4ML resource strategy, it is doable to fit CNN models on a board with much lower resource utilization.

Although the latency results in both tables 6 and 7 are the lowest that achieved with HLS4ML, these results are with uncompressed ML models. To compress the model size, we used different quantization settings, with the most optimum HLS4ML configuration focused on the latency, such as reuse factor 1, and latency strategy, to find the best configuration for the CookieBox and billion-pixel camera. We additionally decrease the bit depth of various models, which corresponds to the input image size and the complexity of the model input. The implementation results for different quantization bit depths for the CookieBox application are presented in table 8. We were able to achieve 1.9 μs with similar accuracy. Moreover, reduced bit depth also deflates the model sizes and lowers resource utilization. Thus, by using a lower bit depth in HLS4ML, the final model can be implemented on a small board with excellent latency. According to table 8, choosing higher bit depth and CNN models causes higher latency and resource utilization.

The results for the FCNN with 7 bit depth are not available because of a software limitation in Vivado, which limits the number of parameters per layer to 4096. The 7 bit FCNN exceeds that limit when using a 16×128 flattened image input size.

The same experience is repeated for the billion-pixel camera application in table 9. Here, the NN aims to find sparse representations of large gray-scale images. Because this is an image encoding and decoding process, there is no accuracy metric to rely on. Instead, we use sparsity rates and peak signal to noise ratio (PSNR) metrics to judge the quality of the encoding and decoding, respectively, where higher values indicate better NN performance. Similar to the CookieBox, we test multiple quantization bit depths. The 8 bit depth is a reference value due to its good latency, sparsity, PSNR, and low resource utilization. We test both a lower bit depth for lower latency and a higher bit depth to confirm the effect of quantization on resource utilization. The FPGA implementation resource utilization and quantization relation for both CookieBox, and billion-pixel camera applications are also illustrated in figures 8 and 9, respectively. The provided information includes latency calculations in microseconds, along with the usage ratio for DSP, look-up table,

Table 8. CookieBox model quantization results.

Model	FCNN			CNN		
Bit depth	3	5	7	3	5	7
Accuracy (%)	81	84	NA	82	86	84
Size (Params.)	1897	3433	9577	3065	3665	6065
Latency (us)	1.9	5.7	NA	11.4	21.1	91
BRAM	4	4	NA	46	96	190
DSP (%)	0	1	NA	13	20	26
LUT (%)	21	39	NA	29	45	63
FF (%)	1	2	NA	6	8	10

Table 9. Billion-pixel camera model quantization results.

Model	FCNN			CNN		
Bit depth	5	8	11	5	8	11
Code Sparsity (%)	94	99	99	60	94	96
PSNR (dB)	40.7	44.6	43.8	32.4	29.5	29.4
Size (Params.)	2593	2593	2593	2185	2185	2185
Latency (us)	0.82	0.89	0.9	69.1	94.1	97.3
BRAM	2	2	2	9	9	9
DSP (%)	17	15	58	0	17	55
LUT (%)	22	38	55	25	40	36
FF (%)	3	5	5	14	19	20

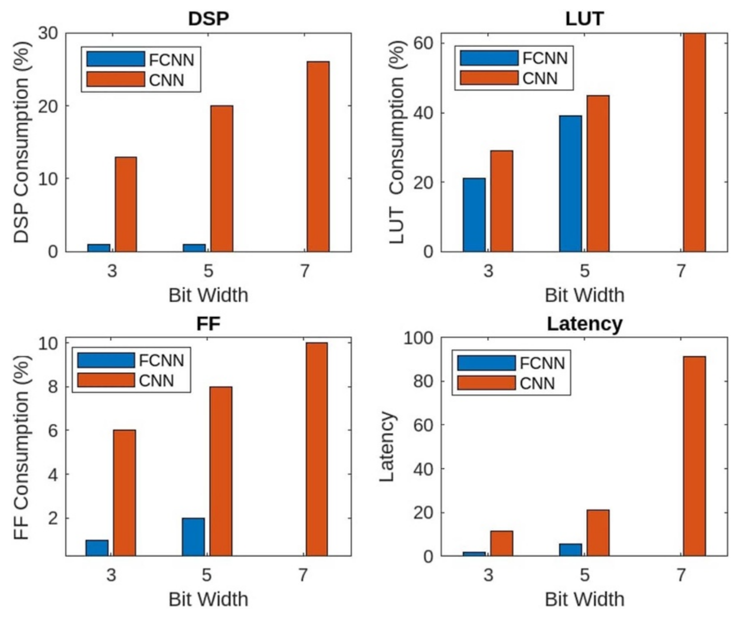


Figure 8. CookieBox model different bit widths relation with implementation results.

and flip-flop, which represent the percentage usage of each resource out of their total count in the FPGA, for both applications.

5. Discussion

It is evident that selecting a higher bit depth leads to improved accuracy in the final FPGA implementation. However, increasing the quantization bits, especially in CNNs, substantially increases resource utilization and latency. However, the relationship between model size and resource utilization is not exactly linear. Choosing a high-precision model for FCNN with our current strategy is limited by the number of parameters per layer

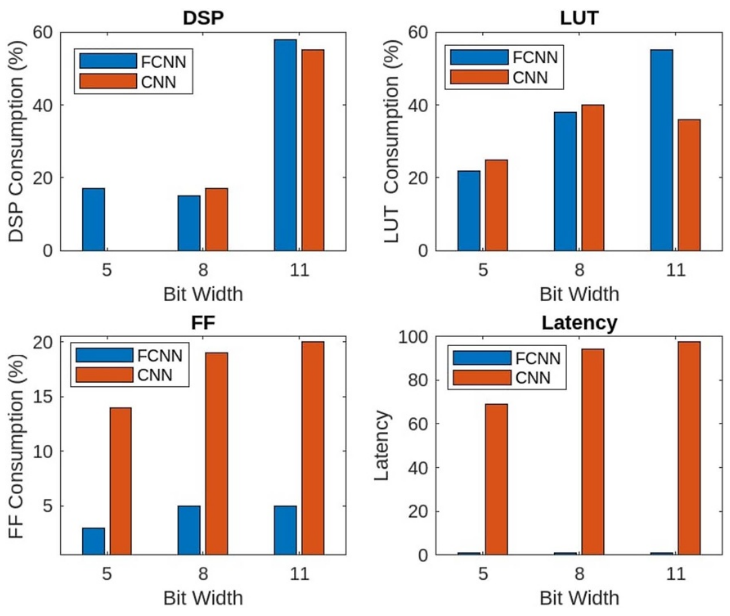


Figure 9. Billion-pixel camera model different bit widths relation with implementation results.

of 4096. To address this issue, one approach is to use a smaller model. Additionally, Vivado's forthcoming new features and updates for larger boards can provide a solution to this challenge in the future.

Most existing ML to FPGA tool-flows are designed for CNNs rather than other neural network architectures. CNNs are widely used in several image applications but are harder to fit on FPGAs due to the large number of operations for the convolutional layers. However, for high data-rate instrumentation applications, FCNN models are often sufficient. These models are fully supported by two available tool-flows only: HLS4ML and FINN. Moreover, after testing an EdgeML application with an FPGA, researchers may migrate to ASICs for the final fixed application. This is not possible with the tool-flows that do not provide the HLS/HDL codes. However, moving to ASIC using HLS4ML and FINN is doable. All in all, we chose HLS4ML and FINN as potential candidates in EdgeML instrumentation applications.

We first showed that FINN is generally a better choice for smaller boards and that HLS4ML performs better considering the latency, which makes it a great candidate for high data-rate instrumentation applications. As shown in the instrumentation applications' results, we were able to achieve excellent results with different quantized models using HLS4ML for the CookieBox, and the billion-pixel Camera. As mentioned earlier, although EdgeML models have been utilized for instrumentation, there has been limited prior research focusing on the CookieBox, and the billion-pixel camera. An HDL-based ML model for CookieBox has been utilized in [58], demonstrating a 20 μs latency, which is higher than this paper's implementation using HLS4ML. In addition, [61] demonstrates a remarkable 100:1 high compression ratio, and a 99% code sparsity for the billion-pixel camera with a minimal latency of 0.89 μs latency on FPGA, by using the HLS4ML tool-flow.

Furthermore, we examined the latency of different models and observed that a significant portion of the latency arises from fetching the input data, rather than processing it. This explains why the billion-pixel camera model runs with lower latency, as it has a smaller input data shape. However, this situation could be improved by increasing the input bus limit, resulting in a significant decrease in latency. The reason HLS4ML and FINN might not allow this could be their reliance on main target boards with processors, which imposes limits on increasing the input bus. Additionally, we also noticed that the ML blocks by HLS4ML are not fully pipelined. This is not a problem for the block latency but limits the throughput. The pipelining technique can enhance the throughput of all models in the ML block.

It is worth mentioning that although HLS4ML supports a wide selection of layers, it is not straightforward to change its back-end codes and add custom layers. We had to add a custom layer for the billion-pixel camera application but the new custom layer behaved strangely in some cases, specifically with the CNN model where the latency strategy did not produce meaningful results. The use of the resource strategy explains the high latency for the billion-pixel camera's best latency strategy in table 7, as well as the low resource usage for the CNN in table 9, especially considering DSPs. The incompatibility between the latency strategy and the custom layer may be fixed in future updates.

6. Conclusion

In this paper, we explained the new developments in high data-rate instrumentation and why they essentially need low-latency solutions such as EdgeML. We presented an exhaustive exploration of currently available tool-flows for EdgeML on FPGA with a focus on their usability for scientific high data-rate instrumentation applications. Our selection has been on those tool-flows that offer an attractive variety of supported networks, optimization, compression, platforms, and accessibility. After comparing, and testing ML to FPGA tool-flows, we noticed that the best choice for a practical instrumentation application with lower latency, especially the high data-rate instrumentation applications, is HLS4ML due to its numerous optimizations, configuration options, and the possibility of being used for ASICs. For lower-resource platforms and smaller FPGAs, FINN is a more suitable tool-flow since it is mainly focused on small and highly quantized NN models. HLS4ML, and FINN differ in their implementation strategies: HLS4ML demonstrates the potential for low-latency ML applications, and FINN minimizes resource and power usage. Here, we see HLS4ML as an excellent candidate for further research in instrumentation as we tested it for two different high data-rate instrumentation applications: (1) CookieBox and (2) billion-pixel camera. In the short term, we plan to implement EdgeML models using these tool-flows near a detector emulator, such as an arbitrary waveform generator. Subsequently, this work will guide future FPGA implementations as a part of an EdgeML-based real-time analysis of high-velocity data in large experiments.

Data availability statement

The data that support the findings of this study are available upon reasonable request from the authors.

Funding

This research was undertaken, in part, thanks to funding from the Canada Research Chairs Program. A C Therrien holds the Canada Research Chair in Real-Time Intelligence Embedded for High-Speed Sensors.

Conflict of interest

The authors declare no conflicts of interest.

Ethics statement

This paper reflects the authors' own research and analysis in a truthful and complete manner.

Appendix. List of abbreviations

Table of abbreviations

Abbreviation	Definition
ML	Machine learning
FPGA	Field programmable gate arrays
ASIC	Application-specific integrated circuits
EdgeML	Edge machine learning
TB	Tera bytes
LHC	Large Hadron Collider
LCLS	LINAC coherent light source
GB	Giga bytes
SLAC	Stanford Linear Accelerator Center
DAQ	Data acquisition
CPU	Central processing unit
GPU	Graphics processing unit
EIC	Electron–ion collider
I/O	Input/output
HDL	Hardware description language
HLS	High level synthesis
NN	Neural network
BNN	Binarized neural network
ONNX	Open Neural Network Exchange
QONNX	Quantized Open Neural Network Exchange
AI	Artificial intelligence
VTA	Versatile tensor accelerator
IP	Intellectual property
TVM	Tensor virtual machine
ALU	Arithmetic logic unit
DLP	Deep learning processor
DNN	Deep neural network
RTL	Register transfer level
LLVM	Low level virtual machine
CNN	Convolutional neural network
FCNN	Fully connected neural network
RNN	Recurrent neural network
AXI	Advanced extensible interface
SCC	Sparse categorical cross-entropy
MSE	Mean square error
DSP	Digital signal processors
LUT	Look-up table
FF	Flip-flops
SVHN	Street View House Number data set
XFEL	x-ray free-electron laser
PSNR	Peak signal to noise ratio
AWG	Arbitrary waveform generator

ORCID iDs

Mohammad Mehdi Rahimifar https://orcid.org/0000-0002-6582-8322
Quentin Wingering https://orcid.org/0009-0007-6301-8450

References

[1] Aiello M, Cavaliere C, D'Albore A and Salvatore M 2019 The challenges of diagnostic imaging in the era of big data *J. Clin. Med.* **8** 316
[2] Das P *et al* (CMS Collaboration) 2022 An overview of the trigger system at the CMS experiment *Phys. Scr.* **97** 054008
[3] Jeitler M 2017 Trigger systems of LHC experiments *J. Instrum.* **12** C05012
[4] Smith W H 2002 Triggering at LHC experiments *Nucl. Instrum. Methods Phys. Res.* A **478** 62–67
[5] Valente M 2019 The ATLAS trigger and data acquisition upgrades for the high luminosity LHC (HL-LHC) *Technical Report* ATL-COM-DAQ-2019-156
[6] Hartmann N *et al* 2018 Attosecond time–energy structure of x-ray free-electron laser pulses *Nat. Photon.* **12** 215–20
[7] Lin S, Zhou Z, Zhang Z, Chen X and Zhang J 2020 *Edge Intelligence in the Making: Optimization, Deep Learning and Applications* (*Synthesis Lectures on Learning, Networks and Algorithms* vol 1) pp 1–233
[8] Zada Khan W, Ahmed E, Hakak S, Yaqoob I and Ahmed A 2019 Edge computing: a survey *Future Gener. Comput. Syst.* **97** 219–35

[9] Liu F, Tang G, Li Y, Cai Z, Zhang X and Zhou T 2019 A survey on edge computing systems and tools *Proc. IEEE* **107** 1537–62
[10] Therrien A C, Gouin-Ferland B and Mehdi Rahimifar M 2022 Potential of edge machine learning for instrumentation *Appl. Opt.* **61** 1930–7
[11] Li B, Gu J and Jiang W 2019 Artificial intelligence (AI) chip technology review *2019 Int. Conf. on Machine Learning, Big Data and Business Intelligence* (*MLBDBI*) (IEEE) pp 114–7
[12] Hu Y, Liu Y and Liu Z 2022 A survey on convolutional neural network accelerators: GPU, FPGA and ASIC *2022 14th Int. Conf. on Computer Research and Development* (*ICCRD*) (IEEE) pp 100–7
[13] Parra D and Camargo C 2018 A systematic literature review of hardware neural networks *2018 IEEE 1st Colombian Conf. on Applications in Computational Intelligence* (*ColCACI*) (IEEE) pp 1–6
[14] Suresh A, Reddy B N and Renu Madhavi C H 2020 Hardware accelerators for edge enabled machine learning *2020 IEEE Region 10 Conference* (*TENCON*) (IEEE) pp 409–13
[15] Furletov S, Barbosa F, Belfore L, Dickover C, Fanelli C, Furletova Y, Jokhovets L, Lawrence D and Romanov D 2022 Machine learning on FPGA for event selection *J. Instrum.* **17** C06009
[16] Hong T M, Carlson B T, Eubanks B R, Racz S T, Roche S T, Stelzer J and Stumpp D C 2021 Nanosecond machine learning event classification with boosted decision trees in FPGA for high energy physics *J. Instrum.* **16** 08016
[17] Xuan T, Durao F and Sun Y 2022 High performance FPGA embedded system for machine learning based tracking and trigger in sPhenix and EIC *J. Instrum.* **17** C07003
[18] Sun C, Nakajima T, Mitsumori Y, Horii Y and Tomoto M 2023 Fast muon tracking with machine learning implemented in FPGA *Nucl. Instrum. Methods Phys. Res.* A **1045** 167546
[19] Nane R *et al* 2015 A survey and evaluation of FPGA high-level synthesis tools *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.* **35** 1591–604
[20] Lahti S, Sjövall P, Vanne J and Hämäläinen T D 2018 Are we there yet? A study on the state of high-level synthesis *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.* **38** 898–911
[21] Cong J, Lau J, Liu G, Neuendorffer S, Pan P, Vissers K and Zhang Z 2022 FPGA HLS today: successes, challenges and opportunities *ACM Trans. Reconfigurable Technol. Syst.* **15** 1–42
[22] Duarte J *et al* 2021 Fast convolutional neural networks on FPGAs with hls4ml *Mach. Learn.: Sci. Technol.* **2** 045015
[23] Blott M , Preußer T B, Fraser N J, Gambardella G, O'brien K, Umuroglu Y, Leeser M and Vissers K 2018 FINN-*r*: an end-to-end deep-learning framework for fast exploration of quantized neural networks *ACM Trans. Reconfigurable Technol. Syst.* **11** 1–23
[24] Xilinx Vitis AI 2022 (available at: www.xilinx.com/products/design-tools/vitis/vitis-ai.html) (Accessed 16 May 2022)
[25] TVM VTA 2022 (available at: https://tvm.apache.org/docs/topic/vta/index.html)
[26] MATLAB DLP 2022 Deep Learning HDL Toolbox (available at: www.mathworks.com/products/deep-learning-hdl.html)
[27] OpenVINO 2023.2 2022 (available at: https://docs.openvino.ai/latest/index.html)
[28] Levental M, Khan A, Chard K, Foster I, Chard R and Yoshi K 2023 OpenHLS: high-level synthesis for low-latency deep neural networks for experimental science (arXiv:2302.06751)
[29] NNgen: a fully-customizable hardware synthesis compiler for deep neural network 2022 (available at: https://github.com/NNgen/nngen)
[30] Ye H, Hao C, Cheng J, Jeong H, Huang J, Neuendorffer S and Chen D 2022 ScaleHLS: a new scalable high-level synthesis framework on multi-level intermediate representation *2022 IEEE Int. Symp. on High-Performance Computer Architecture* (*HPCA*) (IEEE) pp 741–55
[31] Prakash S, Callahan T, Bushagour J, Banbury C, Green A V, Warden P, Ansell T and Janapa Reddi V 2022 CFU playground: full-stack open-source framework for tiny machine learning (tinyML) acceleration on FPGAs (arXiv:2201.01863)
[32] Esmaeilzadeh H *et al* 2021 VeriGOOD-ML: an open-source flow for automated ML hardware synthesis *2021 IEEE/ACM Int. Conf. On Computer Aided Design* (*ICCAD*) (IEEE) pp 1–7
[33] Sharma H, Park J, Amaro E, Thwaites B, Kotha P, Gupta A, Kyung Kim J, Mishra A and Esmaeilzadeh H 2016 DnnWeaver: from high-level deep network models to FPGA acceleration *The Workshop on Cognitive Architectures*
[34] Wielgosz M and Karwatowski M 2019 Mapping neural networks to FPGA-based iot devices for ultra-low latency processing *Sensors* **19** 2981
[35] Venieris S I and Bouganis C-S 2016 fpgaConvNet: a framework for mapping convolutional neural networks on FPGAs *2016 IEEE 24th Annu. Int. Symp. on Field-Programmable Custom Computing Machines* (*FCCM*) (IEEE) pp 40–47
[36] Rybalkin V, Pappalardo A, Ghaffar M M, Gambardella G, Wehn N and Blott M 2018 FINN-L: library extensions and design trade-off analysis for variable precision lstm networks on FPGAs *2018 28th Int. Conf. on Field Programmable Logic and Applications* (*FPL*) (IEEE) pp 89–897
[37] Noronha D H, Salehpour B and Wilton S J E 2018 LeFlow: enabling flexible FPGA high-level synthesis of tensorflow deep neural networks *FSP Workshop 2018; 5th Int. Workshop on FPGAs for Software Programmers* (VDE) pp 1–8
[38] Xu J, Liu Z, Jiang J, Dou Y and Li S 2018 CaFPGA: an automatic generation model for CNN accelerator *Microprocess. Microsyst.* **60** 196–206
[39] Xu P, Zhang X, Hao C, Zhao Y, Zhang Y, Wang Y, Li C, Guan Z, Chen D and Lin Y 2020 AutoDNNchip: an automated DNN chip predictor and builder for both FPGAs and ASICs *Proc. 2020 ACM/SIGDA Int. Symp. on Field-Programmable Gate Arrays* pp 40–50
[40] Guan Y, Liang H, Xu N, Wang W, Shi S, Chen Xi, Sun G, Zhang W and Cong J 2017 FP-DNN: an automated framework for mapping deep neural networks onto FPGAs with RTL-HLS hybrid templates *2017 IEEE 25th Annu. Int. Symp. on Field-Programmable Custom Computing Machines* (*FCCM*) (IEEE) pp 152–9
[41] Gokhale V, Zaidy A, Xian Ming Chang A and Culurciello E 2017 Snowflake: an efficient hardware accelerator for convolutional neural networks *2017 IEEE Int. Symp. on Circuits and Systems* (*ISCAS*) (IEEE) pp 1–4
[42] Zeng H, Chen R, Zhang C and Prasanna V 2018 A framework for generating high throughput CNN implementations on FPGAs *Proc. 2018 ACM/SIGDA Int. Symp. on Field-Programmable Gate Arrays* pp 117–26
[43] Abdelouahab K, Bourrasset C, Pelcat M, Berry F, Quinton J-C and Sérot J 2016 A holistic approach for optimizing DSP block utilization of a CNN implementation on FPGA *Proc. 10th Int. Conf. on Distributed Smart Camera* pp 69–75
[44] Guo K, Sui L, Qiu J, Yu J, Wang J, Yao S, Han S, Wang Y and Yang H 2017 Angel-Eye: a complete design flow for mapping CNN onto embedded FPGA *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.* **37** 35–47
[45] Zhang C, Sun G, Fang Z, Zhou P, Pan P and Cong J 2018 Caffeine: toward uniformed representation and acceleration for deep convolutional neural networks *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.* **38** 2072–85
[46] Khoda E E *et al* 2023 Ultra-low latency recurrent neural network inference on FPGAs for physics applications with hls4ml *Mach. Learn.: Sci. Technol.* **4** 025004

[47] Duarte J *et al* 2018 Fast inference of deep neural networks in FPGAs for particle physics *J. Instrum.* **13** 07027
[48] Plagwitz P, Hannig F, Ströbel M, Strohmeyer C and Teich J 2021 A safari through FPGA-based neural network compilation and design automation flows *2021 IEEE 29th Annu. Int. Symp. on Field-Programmable Custom Computing Machines* (*FCCM*) (IEEE) pp 10–19
[49] Ghielmetti N *et al* 2022 Real-time semantic segmentation on FPGAs for autonomous vehicles with hls4ml *Mach. Learn.: Sci. Technol.* **3** 045011
[50] Borras H *et al* 2022 Open-source FPGA-ML codesign for the MLPerf Tiny Benchmark (arXiv:2206.11791)
[51] Xilinx Vitis AI Model Zoo 2023 (available at: https://github.com/Xilinx/Vitis-AI/tree/master/model_zoo) (Accessed 16 February 2023)
[52] HDLCoder 2023 (available at: www.mathworks.com/products/hdl-coder.html) (Accessed 16 February 2023)
[53] MLIR 2023 Multi-level intermediate representation overview (available at: https://mlir.llvm.org/)
[54] UNSW-NB15 data set 2023 (available at: https://research.unsw.edu.au/projects/unsw-nb15-dataset) (Accessed 16 September 2023)
[55] The Street View House Numbers (SVHN) Dataset 2023 (available at: http://ufldl.stanford.edu/housenumbers/)
[56] HLS4ML_Additional_Boards 2023 (available at: https://github.com/MehdiRh17/HLS4ML_Additional_Boards.git/)
[57] Wang Z *et al* Billion-pixel x-ray camera (BiPC-X) *Rev. Sci. Instrum.* **92** 043708 2021
[58] Corbeil Therrien A, Herbst R, Quijano O, Gatton A and Coffee R 2019 Machine learning at the edge for ultra high rate detectors *2019 IEEE Nuclear Science Symp. and Medical Imaging Conf.* (*NSS/MIC*) (IEEE) pp 1–4
[59] Gouin-Ferland B, Mehdi Rahimifar M, Granger C-E, Coffee R and Corbeil Therrien A 2022 Combining optimized quantization and machine learning for real-time data reduction at the edge *2022 IEEE Nuclear Science Symp. and Medical Imaging Conf.* (*NSS/MIC*) (IEEE) pp 1–4 accepted
[60] Hu C *et al* 2019 Ultrafast inorganic scintillator-based front imager for gigahertz hard x-ray imaging *Nucl. Instrum. Methods Phys. Res.* A **940** 223–9
[61] Ezzaoui Rahali H, Mehdi Rahimifar M M, Étienne Granger C, Wang Z and Therrien A C 2024 Efficient compression at the edge for real-time data acquisition in a billion-pixel x-ray camera *Nucl. Instrum. Methods Phys. Res.* A **1058** 168829
[62] Ngadiuba J *et al* 2020 Compressing deep neural networks on FPGAs to binary and ternary precision with hls4ml *Mach. Learn.: Sci. Technol.* **2** 015001