



UNICA

UNIVERSITÀ
DEGLI STUDI
DI CAGLIARI



Università di Cagliari

UNICA IRIS Institutional Research Information System

This is the Author's *accepted* manuscript version of the following contribution:

F. Manca, F. Ratto, C. Rubattu, L. Raffo and F. Palumbo, "Adaptive CNN Acceleration on FPGAs: Closing the Gap With State-of-the-Art Solutions," in *IEEE Embedded Systems Letters*, vol. 18, no. 4, pp. 257-261, Aug. 2026.

© 2025 IEEE. Personal use of this material is permitted. Permission from IEEE must be obtained for all other uses, in any current or future media, including reprinting/republishing this material for advertising or promotional purposes, creating new collective works, for resale or redistribution to servers or lists, or reuse of any copyrighted component of this work in other works.

The publisher's version is available at:

<http://dx.doi.org/10.1109/LES.2025.3599237>

When citing, please refer to the published version.

Adaptive CNN Acceleration on FPGAs: Closing the Gap with State-of-the-Art Solutions

Federico Manca, Francesco Ratto, Claudio Rubattu, Luigi Raffo, and Francesca Palumbo, *Senior Member, IEEE*

Abstract—This paper presents a comparative study of a design flow for generating Convolutional Neural Network (CNN) accelerators on Field Programmable Gate Arrays (FPGAs), based on an extension of the Multi-Dataflow Composer (MDC) tool, against established frameworks: HLS4ML, FINN and Vitis AI. The proposed design flow explores a previously untapped area of the design space: runtime reconfigurable accelerators. By enabling runtime reconfigurability, it provides adaptivity support, filling a gap in current FPGA-based accelerator design options. The analysis focuses on the trade-offs and benefits of each approach, particularly regarding performance and adaptivity.

Index Terms—Adaptivity, QONNX, Convolutional Neural Networks, FPGAs, Cyber-Physical Systems

I. INTRODUCTION

CNNs are increasingly applied at the edge [1] thanks to the performance benefits that can be guaranteed, particularly when using customizable devices like FPGA-based Multi-Processor System on Chip, that provide general-purpose cores tightly coupled with an FPGA. Hardware (hw)-software (sw) codesign approaches tackle the challenges posed by limited computational resources and memory, but require complex, time-consuming design processes. Demanding, in turn, automated methodologies to streamline CNN accelerator codesign on edge FPGAs.

State-of-the-art (SoA) solutions for accelerating CNNs on FPGAs can be classified into two categories: single processing engines, typically implemented as matrices of processing elements, and streaming architectures, made by a pipeline of customizable hw blocks [2]. For single processing engines, various solutions have been proposed [3]–[5], with Vitis AI¹ standing out as the most mature framework for AMD devices. Similarly, several streaming architecture-based solutions are there [6]–[8], with HLS4ML [9] and FINN [10] emerging as the most established ones. They enable designing custom CNN accelerators for FPGAs with efficient memory and hw utilization by tailoring the design to the target CNN. They lack runtime reconfigurability, which enables the execution of different profiles [11]. This feature is crucial for adapting to diverse computational tasks² and adjusting to varying execution constraints [12], particularly in vision and

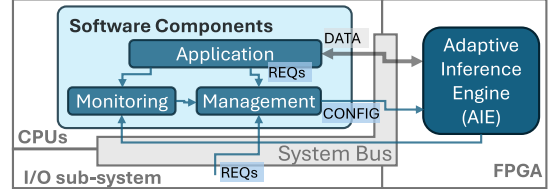


Fig. 1: Adaptive edge node. The Management component reconfigures the AIE considering the data gathered by the Monitor and the Requirements imposed by the application, e.g. desired accuracy, and the external sources, e.g. a user.

image recognition tasks such as Unmanned Aerial Vehicle applications [13]. Adaptation can be achieved by monitoring the system execution metrics and triggering reconfiguration, for example, to minimize energy consumption while meeting application or external requirements [14]. Such a system, as depicted in Fig. 1, requires a monitoring infrastructure as [15], a management component³ and an Adaptive Inference Engine (AIE), the focus of this work.

To achieve runtime adaptivity, the proposed design flow integrates the QKeras-to-hw toolchain, built upon MDC, an open-source framework for designing coarse-grained reconfigurable hw co-processors [16]. QKeras⁴, a Keras extension for building and training quantized Neural Networks, is now supported by the re-engineered QONNX2MDC⁵ library, enabling direct generation of hw accelerators from QKeras models with minimal manual intervention. Moreover, QONNX [17], an extension of ONNX for quantized models, is established as the intermediate representation to ensure broader compatibility with other frameworks. Building on our previous work [18], focussed on managing multiple models on the same accelerator and minimizing reconfiguration time and power, in this work we target the challenge of integrating runtime-adaptive models into a complete design flow, ensuring that models are efficiently trained, ported, and deployed, by bringing the following contributions:

- An extended toolchain, compatible with the Machine Learning framework QKeras, and an iterative design flow for generating adaptive CNN accelerators on FPGAs.
- A comprehensive comparison, based on literature data and experimental results, with the SoA frameworks HLS4ML, FINN, and Vitis AI, to demonstrate competitive performance while uniquely supporting adaptivity.

F. Manca (e-mail: federico.manca@unica.it), F. Ratto, L. Raffo and F. Palumbo - University of Cagliari. C. Rubattu - University of Sassari.

“MYRTUS is funded by the European Union, by grant No. 101135183. Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union. Neither the European Union nor the granting authority can be held responsible for them.”

¹<https://xilinx.github.io/Vitis-AI>

²<https://www.xilinx.com/publications/ebooks/introducing-adaptive-system-on-modules.pdf>

³Under development in the MYRTUS EU project, as part of the cognitive engine: <https://myrtus-project.eu/myrtus-cognitive-engine/>.

⁴<https://github.com/google/qkeras.git>

⁵<https://github.com/mdc-suite/qonnx2mdc>

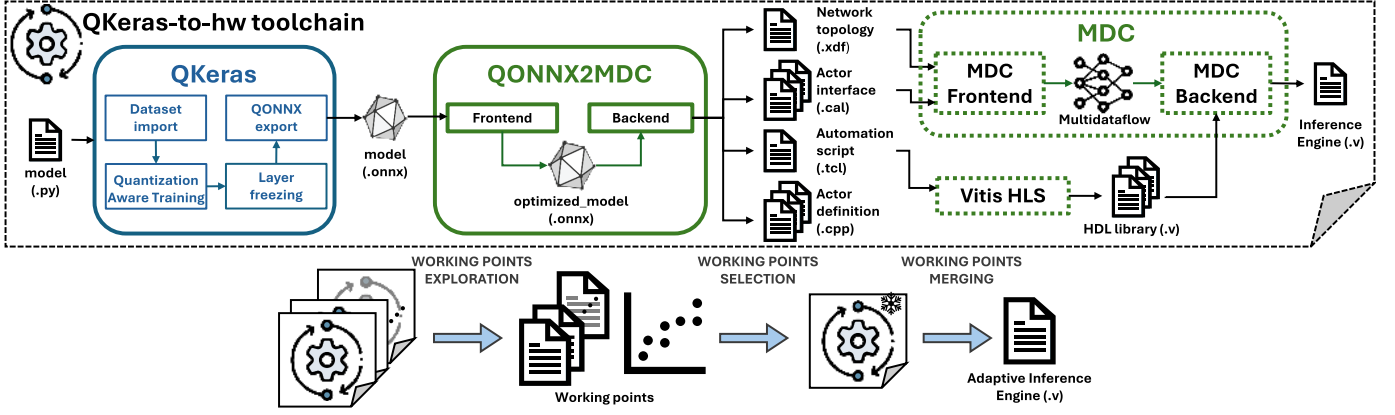


Fig. 2: Top: proposed QKeras-to-hw toolchain [Solid: Novel. Dotted: unchanged]. Bottom: Use for an AIE generation.

II. PROPOSED DESIGN FLOW

Fig. 2 illustrates the proposed automated toolchain and its use. The *QKeras-to-hw toolchain* uses QKeras to define the CNN model with the desired quantization parameters and import the target dataset (*Dataset Import*). Quantization-Aware Training (QAT) is applied to quantized layers to preserve accuracy during training (*Quantization Aware Training*). *Layer Freezing* allows adjustments to unfrozen layers for the generation of distinct working points. After training, *QONNX Export* converts the model to QONNX. The exported model serves as input to the model-to-hw stage of the toolchain. The *QONNX2MDC* library processes the QONNX file, with the *Frontend* applying various optimizations, like folding MatMul and Add nodes into a Gemm node. This results in an optimized QONNX model (*optimized_model*) tailored for efficient High Level Synthesis (HLS) implementation. From the optimized model, the *Backend* generates the following:

- the *Network topology* and *Actor interfaces* files, utilized by MDC Frontend to derive the accelerator dataflow model (*Multidataflow*). In this model, the information travels through arcs (First In First Out channels) which connect nodes, representing different processing units (Actors);
- the *C++ files* and the *Tool Command Language (TCL) script*, used to implement the CNN layers and for the synthesis automation, provided to *Vitis HLS* to generate the *Hardware Description Language (HDL) library*. The C++ layer descriptions use a template architecture, with each actor customizable to implement the input QONNX model. Compared to [18], the templates were updated to shift quantization after activation, aligning with QKeras semantics, with a minimal impact on the hw utilization and no impact on performance.

As the final step of the toolchain, the MDC Backend uses the *Multidataflow* model and the *HDL library* to generate the HDL description of the *Inference Engine*. When processing multiple models with different quantizations, the toolchain produces a reconfigurable model, resulting in a runtime-adaptive accelerator, the AIE.

The toolchain is exploited by the design flow outlined in Alg. 1, and summarized in Fig. 2. It begins with defining the QKeras CNN architecture and target dataset. After setting quantization levels and applying QAT, the QKeras model

Algorithm 1: Process of designing an adaptive accelerator.

```

Input : QKeras CNN architecture, target dataset
Output: Adaptive Inference Engine (AIE)
Data: Working Points = {}

1 repeat
2   do Quantization, QAT and Export;
3   do Non-adaptive HW generation;
4   do Working Point evaluation;
5   if Constraints are met then
6     store Model in the Working Points set
7   else
8     discard Model
9 until Working point exploration is finished;
10 select baseline from Working Points set;
11 repeat
12   select Merging candidate from Working Points set ;
13   do Initialize merging candidate with baseline's weights;
14   do Freeze layers with common precision and retrain;
15   if Accuracy is maintained then
16     store Working Point is selected for merging;
17   else
18     discard Model;
19 until Working points selection is finished;
20 do Generate AIE by merging the selected Working Points

```

is exported to QONNX format and converted into an hw accelerator utilizing the QKeras-to-hw toolchain. If the accuracy is satisfactory and resource utilization remains within acceptable limits, the model is stored, and its working point is added to the Working Points set; otherwise, it is discarded, and new quantization levels are assigned. The process is iterative to generate multiple working points. At the end of the exploration, a baseline working point is selected. A merging candidate is then selected using the stored working points and its QKeras model is initialized with the baseline's weights. The goal is creating models with common layers sharing the same precision for reuse in a reconfigurable coprocessor. Common-precision layers are frozen to keep their parameters constant, while the others are fine-tuned during retraining. The model is stored only if it meets the desired accuracy. The process can be repeated multiple times, obtaining a list of models to be merged in an AIE using the toolchain. The AIE supports runtime reconfiguration between working points with negligible reconfiguration overhead exploiting switching boxes, reducing resource usage and power/energy dissipation by reusing common layers.

III. EXPERIMENTAL RESULTS

A. Evaluation of the design flow

An initial exploration of the quantization levels of a tiny CNN for MNIST generates a working point set. This CNN comprises two blocks with Convolutional (32 filters, 3×3 kernel), ReLU, MaxPool, Flatten, and a Dense layer. The accelerator is deployed on the KRIA Vision AI Starter Kit with synthesis and simulation conducted in Vivado 2022.2. The CNN models use mixed precision, as different layers require varying bit widths [19]. Model names follow the A_xW_yz format, where x and y are the bit widths for activations and weights, and z the model, e.g. M for MNIST. First, accelerators with uniform precision across all layers (A16-W8_M, A8-W4_M, and A4-W2_M) are developed for baseline comparisons. Quantization levels were iteratively selected over a representative range of fixed-point precisions. The models were trained for 10 epochs with categorical cross-entropy loss and the Adam optimizer. Early stopping and learning rate reduction were applied. A4-W2_M shows a significant accuracy drop, see Fig. 3, and is excluded from further refinement. Results indicate that A8-W4_M achieves comparable accuracy to A16-W8_M while using fewer Look-Up Tables (LUTs) and Block RAMs (BRAMs). In contrast, A4-W2_M uses the least resources and power but delivers the lowest accuracy.

TABLE I: Proposed design flow: resource usage and latency

	Name	Latency [us]	LUT [#]	BRAM [#]	DSP [#]	Power [W]
MNIST	A16-W8 _M	329	30.1K	22.5	0	0.227
	A8-W4 _M	329	27.8K	22.5	0	0.189
	A4-W2 _M	329	14.0K	8.5	0	0.102
	Mix-1 _M *	329	23.5K	9.5	0	0.122
	Mix-2 _M *	329	15.6K	9.5	0	0.105
	AIE _M	329	40.3K	23.5	0	0.189/0.105
	M-Sum**	—	43.6K	32	0	—
TEXT	A16-W8 _T	87	14.2K	11.5	86	0.305
	A8-W4 _T	87	10.1K	8	86	0.270
	A4-W2 _T	87	7.8K	7	70	0.198
	Mixed _T	87	7.4K	7	86	0.232
	AIE _T	87	8.2K	8	86	0.232/0.198

* Mix-1_M: 8 bits for the first block and 4 bits for the others. Mix-2_M: 8 bits for activations and Dense layers, and 4 bits for the others.

** Hypothetical configuration; power and latency unavailable.

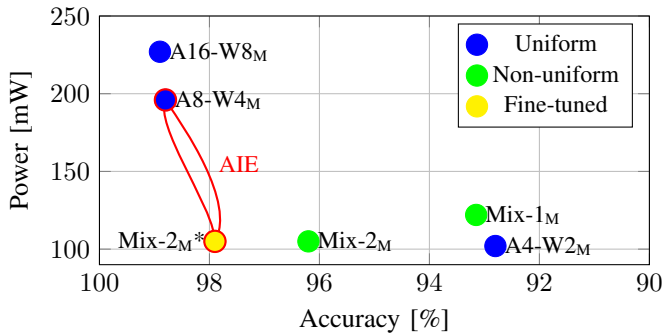


Fig. 3: Power vs. accuracy of MNIST CNN models implemented with the proposed design flow

To demonstrate the design flow’s merging capabilities, two additional non-uniform precision models (Mix-1_M and Mix-2_M) are developed, combining A8-W4_M and A4-W2_M precisions, balancing accuracy and resource usage while keeping shared layers intact for efficient merging. Mix-2_M achieves

higher accuracy (96.2%) than Mix-1_M while maintaining low LUT usage and BRAM consumption. Latency remains identical across all implementations, as Vitis HLS schedules operations regardless of data width in a given code. The A8-W4_M and Mix-2_M models were chosen as baseline and merging candidates. Mix-2_M, initialized with A8-W4_M weights, was fine-tuned for 5 epochs, boosting accuracy from 96.2% to 97.9%. The profiles were merged into an AIE_M, enabling runtime switching between profiles for variable accuracy and power dissipation, as summarized in Tab. I. Enabling resource sharing optimizes resource usage. A hypothetical non-reconfigurable AIE (M-Sum) implementing Mix-2_M and A8-W4_M would require approximately 3000 additional LUTs and 8.5 more BRAMs than the MDC-based AIE.

To validate our design flow on a more complex dataset, we evaluated a CNN for texture classification capable of processing multichannel sensor data (denoted with _T), trained on a 48k-sample, 8-class dataset [20]. Quantized versions achieved sub-100 μ s latency with minimal accuracy loss. A mixed-precision variant (Mixed_T) was merged with the A4-W2_T base to form the AIE_T, saving up to 48% LUTs and 53% BRAMs compared to the fully parallelized design.

B. Comparison with SoA

Tab. II presents the compared frameworks. Our design flow aligns closely with HLS4ML and shares many similarities with FINN, as QONNX input format, streaming architecture, and support for custom quantized precision. The comparison was experimental for HLS4ML and literature-based for Vitis AI and FINN. HLS4ML was tested directly since it shares both frontend and backend with our approach, enabling identical models. FINN’s Brevitas frontend prevents exact replication, while Vitis AI’s distinct hw architecture limits fair comparison, though some tests were conducted given its commercial maturity. The ZYBO Z7-20 board was chosen for HLS4ML models⁶, as it meets toolchain requirements. HLS4ML enables layer-level parallelization tuning by adjusting the Reuse Factor (RF). Different RF configurations were tested for the second convolutional and dense layers, as both exceed the 4096-parameter threshold imposed by Vivado HLS, preventing unrolled compilation. The A8-W4_M model, trained with QKeras following the same methodology as the MDC implementation, was chosen for RF exploration. Tab. III summarizes the results. With RFs of 576/3136, the HLS4ML accelerator H-5 achieves latency comparable to MDC models while maintaining acceptable resource usage. The Mix-2_M model was implemented with the same values, obtaining the H-Mix accelerator. Smaller RFs reduce latency but may exceed board capacity due to higher resource use, as observed with the H-2 model. Even with larger RF, replicating the MDC AIE_M in HLS4ML would demand significantly more resources, as estimated by the H-Sum accelerator, obtained with H-5 and H-Mix instantiated in parallel. In comparison, the AIE_M utilizes 89% fewer BRAMs, 52% fewer LUTs, and avoids using Digital Signal Processings (DSPs), all while achieving a 30% speedup for the inference

⁶The newer KRIA board is incompatible with HLS4ML, which supports only up to Vivado 2020.1.

phase when comparing an MDC inference engine. Tab. IV compares FINN (see [21]), Vitis AI, and HLS4ML. For Vitis AI performance, the CNN was executed on the KRIA board, measuring the System on Module total power as direct access to the IP implementation was unavailable. For our design flow and HLS4ML, power consumption was assessed via post-synthesis simulation and power estimation in Vivado. The resource usage and performance of two CNNs models implemented with Vitis AI with the B4096 architecture, and a CIFAR-10 model with the B1152 architecture, show that Vitis AI's DPU supports flexible architectures with varying parallelism, but it lacks precision tuning and QAT, which makes larger models necessary to achieve high accuracy.

TABLE II: Comparison of Machine Learning Tools for FPGAs

Tool	Vitis AI	FINN	HLS4ML	MDC Flow
Source	ONNX/ Tensor-flow	QONNX/ Brevitas	QONNX/ QKeras	QONNX/ QKeras
Output	IP Config. + Compiled model	HDL + Pynq APIs + Scripts	HDL + Pynq APIs + Scripts	HDL + MDC APIs + Scripts
Precision	8-bit	Custom	Custom	Custom
RT Reconf.	No	No	No	Yes

TABLE III: Resources and latencies of CNNs model implemented with HLS4ML. Reuse tab: x/y indicates the RF for convolutional and dense layers. ZYBO board usage and an AIE vs HLS4ML coprocessors comparison are provided.

Name	Reuse	Latency [us]	LUT [# / %]	BRAM [# / %]	DSP [# / %]
H-1	288/3136	244	43k (81%)	103 (73%)	35 (16%)
H-2	36/196	38	79k (149%)	108.5 (77%)	220 (100%)
H-3	96/784	88	86k (162%)	103 (73%)	114 (51%)
H-4	576/784	462	65k (122%)	100.5 (71%)	34 (15%)
H-5	576/3136	470	42k (79%)	103 (73%)	19 (8%)
H-Mix	576/3136	470	40k (75%)	103 (73%)	19 (8%)
H-Sum	576/3136	—	82k (154%)	206 (147%)	38 (17%)
AIE vs H-5		-30%	-5%	-78%	-100%
AIE vs H-Mix		-30%	-0%	-78%	-100%
AIE vs H-Sum		—	-52%	-89%	-100%

TABLE IV: Positioning with respect to SoA

Tools	HLS4ML*	MDC*	FINN [21]	Vitis AI [22]	Vitis AI	Vitis AI
Board	Z7-20	KV260	Pynq-Z2	Z7-20	KV260	KV260
Dataset	MNSIT	MNIST	CIFAR-10	CIFAR-10	MNIST	CIFAR-10
Power [W]	0.93	0.18	1.69	2.36	—	0.17
Energy [μJ]	436	62	2535	1070	—	204
Latency [μs]	470	329	1500	2266	1199	1200
Accuracy [%]	98.8	98.8	84.5	84.9	86.8	97.9
LUT [#]	42K	27K	24K	32K	52K	52K
BRAM [#]	103	22.5	100	123	255	255
DSP [#]	19	0	0	212	710	710
Parameters [#]	25k	25k	1.4M	~1M	69K	552K

* The models implemented are H-5 and A8-W4_M for HLS4ML and MDC, respectively. Power and energy were obtained via simulation.

Summarizing: i) The proposed flow supports larger models, but on-chip memory limits FPGA deployment, as with FINN and HLS4ML. Additionally, the iterative quantization process may increase training time. Vitis AI can use the off-chip memory to accommodate large models, but does not support deep quantization and QAT. ii) HLS4ML and FINN allow exploration of models with varying latencies and resource usage by adjusting parallelization, whereas MDC supports a single parallelization level. iii) Compared to HLS4ML, MDC shows lower latency and reduced resource and power consumption (see Tab. III and Tab. IV), though HLS4ML offers broader

support for a wider range of Neural Network models. iv) FINN provides lower latency for larger models (e.g., 1.5 ms for a 1.4M-parameter CIFAR-10 model). v) HLS4ML and FINN cannot deploy reconfigurable accelerators, requiring separate accelerators for each model, leading to resource-intensive solutions. In contrast, MDC's AIE_M combines models like Mix-2_M and A8-W4_M, reducing resource usage.

CONCLUSION

We presented a design flow for reconfigurable CNN accelerators with runtime adaptability. Compared to FINN, HLS4ML, and Vitis AI, competitive performance are achieved, while uniquely enabling reconfigurability. Our design flow, currently focused on CNNs, can extend to RNNs and other QKeras-compatible models. QONNX compatibility ensures long-term extensibility for integrating evolving quantized training methods. Future work will focus on adaptivity management.

REFERENCES

- [1] R. Singh *et al.*, "Edge AI: A survey," *Internet of Things and Cyber-Physical Systems*, vol. 3, 2023.
- [2] S. I. Venieris *et al.*, "Toolflows for mapping convolutional neural networks on fpgas: A survey and future directions," *ACM Comput. Surv.*, vol. 51, June 2018.
- [3] T. Sledevič *et al.*, "mNet2FPGA: A Design Flow for Mapping a Fixed-Point CNN to Zynq SoC FPGA," *Electronics*, vol. 9, Nov. 2020.
- [4] K. Guo *et al.*, "Angel-Eye: A Complete Design Flow for Mapping CNN Onto Embedded FPGA," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 37, Jan. 2018.
- [5] Y. Guan *et al.*, "FP-DNN: An Automated Framework for Mapping Deep Neural Networks onto FPGAs with RTL-HLS Hybrid Templates," in *IEEE FCCM*, IEEE, Apr. 2017.
- [6] S. I. Venieris *et al.*, "fpgaConvNet: A Framework for Mapping Convolutional Neural Networks on FPGAs," in *IEEE (FCCM)*, May 2016.
- [7] A. Ghaffari *et al.*, "Cnn2gate: An implementation of convolutional neural networks inference on fpgas with automated design space exploration," *Electronics*, vol. 9, 2020.
- [8] F. Ratto *et al.*, "An automated design flow for adaptive neural network hardware accelerators," *Journal of Signal Processing Systems*, 2023.
- [9] T. Aarrestad *et al.*, "Fast convolutional neural networks on fpgas with hls4ml," *Mach. Learn. Sci. Technol.*, vol. 2, no. 4, p. 45015, 2021.
- [10] Y. Umuroglu *et al.*, "Finn: A framework for fast, scalable binarized neural network inference," in *ACM/SIGDA Symp. on FPGAs*, 2017.
- [11] S. Jiang *et al.*, "SCYLLA: QoE-aware Continuous Mobile Vision with FPGA-based Dynamic Deep Neural Network Reconfiguration," in *IEEE INFOCOM*, 2020.
- [12] A. Biondi *et al.*, "A Framework for Supporting Real-Time Applications on Dynamic Reconfigurable FPGAs," in *IEEE RTSS*, Nov. 2016.
- [13] S. Minakova *et al.*, "Scenario Based Run-Time Switching for Adaptive CNN-Based Applications at the Edge," *ACM Trans. Embed. Comput. Syst.*, vol. 21, Mar. 2022.
- [14] F. Palumbo *et al.*, "Myrtus: Multi-layer 360 dynamic orchestration and interoperable design environment for compute-continuum systems," in *ACM CF*, 2024.
- [15] T. Fanni *et al.*, "Run-time performance monitoring of heterogenous hw/sw platforms using PAPI," *CoRR*, vol. abs/2103.01195, 2021.
- [16] C. Sau *et al.*, "The Multi-Dataflow Composer tool: An open-source tool suite for optimized coarse-grain reconfigurable hardware accelerators and platform design," *Microprocessors and Microsystems*, 2021.
- [17] A. Pappalardo *et al.*, "QONNX: representing arbitrary-precision quantized neural networks," *CoRR*, vol. abs/2206.07527, 2022.
- [18] F. Manca *et al.*, "ONNX-To-Hardware Design Flow for Adaptive Neural-Network Inference on FPGAs," in *SAMOS*, 2025.
- [19] K. Wang *et al.*, "HAQ: hardware-aware automated quantization with mixed precision," in *IEEE CVPR*, 2019.
- [20] H. Al Haj Ali *et al.*, "Neuromorphic tactile sensing system for textural features classification," *IEEE Sensors Journal*, vol. 24, no. 10, 2024.
- [21] H. Borrás *et al.*, "Open-source FPGA-ML codesign for the mlperf tiny benchmark," *CoRR*, vol. abs/2206.11791, 2022.
- [22] A. Ushiroyama *et al.*, "Convolutional neural network implementations using Vitis AI," in *IEEE CCWC*, Jan. 2022.