Towards a Distributed Quantized Machine Learning Inference with Commodity SoC-FPGAs Using FINN

Mathieu Hannoun[1,2]
[1]Laboratoire ETIS, UMR8051, CY Cergy Paris Universités, ENSEA, CNRS, F-95000 Cergy, France
[2]Madicob, 14 Rue du Petit Albi, 95520 Osny, France

Stéphane Zuckerman, Olivier Romain[1]
[1]Laboratoire ETIS, UMR8051, CY Cergy Paris Universités, ENSEA, CNRS, F-95000 Cergy, France

***Abstract*—Deep Neural Networks (DNNs) have experienced significant growth over the years, accompanied by a corresponding rise in energy consumption due to their escalating demand for computational resources. To mitigate the environmental impact of AI, and address growing concerns over data privacy, a growing trend is to process data locally at the edge rather than relying on large-scale data centers. FPGA-based systems are particularly suited for this kind of applications, with their low power consumption to high parallel computation ratio. The main drawback of commodity FPGAs is their limited hardware resources, constraining the size of the DNNs which can run efficiently on such targets. This paper presents a methodology for distributed DNNs on multiple commodity FPGAs to support models that are usually only suited for larger FPGAs. We are able to support the inference for a MobileNetV1 on six Zedboards with a peak throughput of 118.3 inferences per second for an estimated power consumption of 16.176 Watts.**

Keywords-*Machine Learning, Deep Learning, CNN, FPGA, Edge Computing*

I. Introduction

The inference of deep neural networks (DNNs) and particularly convolutional neural networks (CNNs) at the edge (through edge and fog nodes) has gained in popularity for its energy efficiency, data locality, and data privacy – all growing concerns [1]. This computation can be done by a variety of hardware, *e.g.*: smartphones [2], [3], microcontrollers [4] and FPGAs, leading to the rise Tiny Machine Learning (TinyML) [5], *i.e.*, the deployment of machine learning models on ultra low-power, resource-constrained devices. Microcontrollers are a possible target, but while they fulfill the need for very low-power consumption, they lack the efficient parallel computation capability of FPGAs, while suffering the same drawbacks of having too few resources to support larger models. Performing the inference of large scale DNNs and CNNs require a massive amount of floating-point operations.

As data privacy is an important factor, implementing a system closer to the data source may help avoid resorting to cloud-based solutions. Yet, high-end FPGA are expensive and have a high absolute power consumption (can be over hundreds of watts). In the context of smart buildings, where heterogeneous low-power devices are abundant, commodity FPGAs offer a very promising trade-off: they yield a low power footprint while being able to take advantage of parallelism in a pre-trained neural network model, thus being able to deliver great performances in the context of edge computing. However, commodity FPGAs offer limited hardware resources, making the implementation of DNNs challenging for models larger than a few binarized layers.

Multiple techniques have been developed in recent years to provide a flexible way to implement DNNs on FPGA, such as FINN [6], from weight compression using quantization, up to using only one or two bits for weights and bias [7]. Operations conversion has also been developed to leverage this level of compression [8], [9]. Few works have tackled distributed inference on FPGAs to support larger networks or to speed up computation. Alonso et al. [10] focused on resource partition and optimization for splitting a MobileNetV1 and a Resnet50 across two high-end FPGA boards. It is based on direct FPGA to FPGA communication with 100 Gbps Ethernet links, using VNx IP cores. While this setup yields very high throughput, it is not suited for low-power IoT devices. Some works have explored such hardware. Notably, Fiscaletti et al. [11] used FINN to split a network of binarized CNN on three Pynq boards, but their model splitting was done manually. Jiang et al. [12] have tackled the implementation of a framework for DNN distributed inference on multiple FPGAs. They use it on two boards to speed up DNN inference, but it does not implement the hardware optimizations used by FINN related to quantized neural networks.

In this paper, we address the problem of how to fit DNNs onto several embedded devices on the same network while leveraging FPGAs suitability for hardware acceleration and their low power consumption. In addition, we are interested in the possibility to substitute a dedicated high-end FPGA for a multiple of networked low-power commodity FPGAs. We propose a way to distribute DNN models inference over several FPGAs when a single board is not sufficient to hold the whole DNN. Communications are handled by the CPU, while the actual inference is done by the FPGA.

II. Methodology

Our approach enables the deployment of DNNs, typically suited for high-end FPGAs (*e.g.*, Alveo U250) due to their size, onto commodity FPGAs by partitioning the model into multiple sub-models. These sub-models are distributed across a network of SoC-FPGAs.

Our methodology provides a flexible and scalable approach to FPGA-based inference, enabling larger models to run on low-

Manuscript received April 30, 2025; revised July 24, 2025; accepted August 13, 2025. Published September 2, 2025.
Issue category: Special Issue on DSD/SEAA 2025 on Works in Progress (WiP) Session, Salerno, Italy, Sept. 2025
Paper category: Short
DOI: doi.org/10.64552/wipiec.v11i1.76

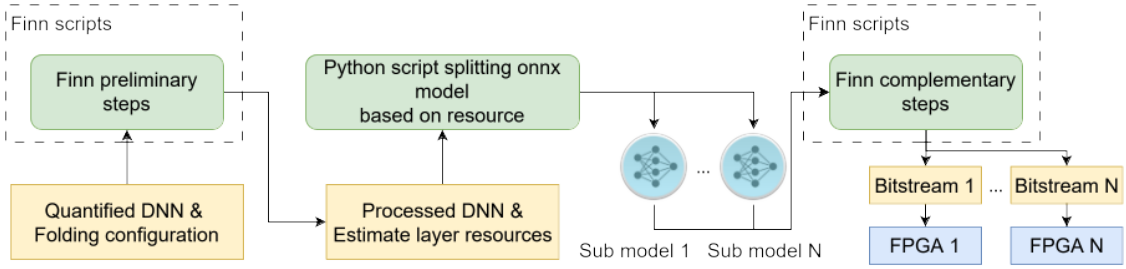


Fig. 1: DNN splitting process. Complete pipeline from quantified DNN to FPGA implementation.

cost hardware. While it does not yet match the performance of high-end FPGAs, it represents a step toward more accessible and efficient distributed inference solutions. Our work uses FINN as a basis for hardware implementation to run inferences. FINN can convert a DNN model to target an FPGA. It will take an ONNX model representing the DNN architecture and its weights, convert it to multiple intermediate representations and output a bitstream file, including the model and custom Direct Memory Access (DMA) engines. However, the intended use is to take a complete model to produce a configuration that will fit in a single FPGA. Our goal is to partition the model into smaller sub-models that can each fit into a commodity FPGA and communicate with each other to complete the overall model. To achieve this, we split the ONNX model representing the DNN, use FINN to generate a bitstream from each sub-model, use those bitstreams to configure each board, and establish communication with every board to run inferences.

Network Splitting Strategy

To allow medium size DNN models to fit on low-power FPGAs, it is necessary to split them to fit resource constraints. FINN [9] is used to automatically generate a preprocessed model and a per-layer resource estimation. The resulting split is based on those models. The objective of our automated splitter is to maximize resource occupancy to reduce the number of bitstreams. Fig. 1 provides an overview of our splitting process. Currently, only the network splitting is automatic. FINN's preliminary and complementary steps are still launched manually.

To find a suitable cut, the ONNX graph is traversed, starting from the first layer. For each node, the required resources estimate (*i.e.*, LUTs, BRAMs, DSPs) are added, until they exceed available resources for a given board. For now, the only objective of our automated splitter is to maximize resource occupancy to reduce the number of bitstream files (in the future, other metrics and objectives may be targeted, such as the amount of data transmitted between layers to reduce network traffic). The original ONNX model is then split into several ONNX submodels. The resulting models are then fed to FINN to generate FPGA design as well as a bitstream for each sub-model. We modified FINN to support the Zedboard for bitstream generation (since we do not use the Pynq OS, we have no need for the Pynq overlay). For now we target identical boards, but in the future we will have different types of SoC-FPGA systems, and some submodel may only fit on specific boards, or alternatively, a given sub-model may be allocated differently resource-wise (to favor resource utilization, limit power consumption, *etc.*) depending on the target hardware for a given submodel. A more complex automatic solver will be required then.

III. Experimental Results

A. Experimental Testbed

The overall system is architected as a pipeline, with each SoC-FPGA board running its own client/server software. The client part sends data to the next board while the server part waits and processes data from the previous board (see Fig. 2). Submodels are generated beforehand, as explained in Section II. Each board embeds all submodels (including Initial weights), stored locally as bitstreams. A client can distribute a DNN on-the-fly by ordering a selected board to reconfigure its Programmable Logic (PL) into the desired submodel.

Each board can be queried independently by the orchestrator. Communications are thus handled by a TCP server written in C++, and running on the Processing System (PS) side, as shown in Fig. 2. The server directly interacts with the PL part to trigger network inferences through the CPU.

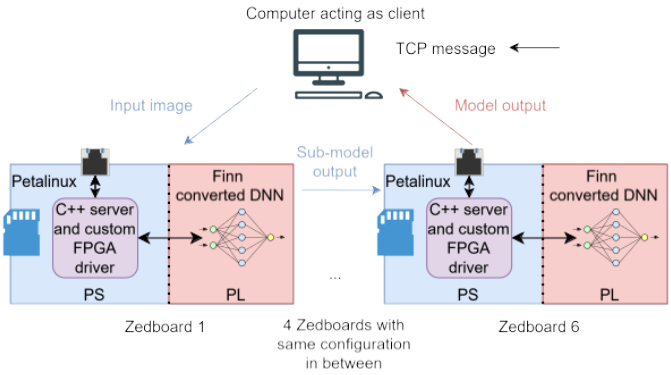


Fig. 2: Example of a full communication for a DNN divided into six bitstreams, from an image input to the model classification (Sub model 6 output), from right to left in a sequential manner.

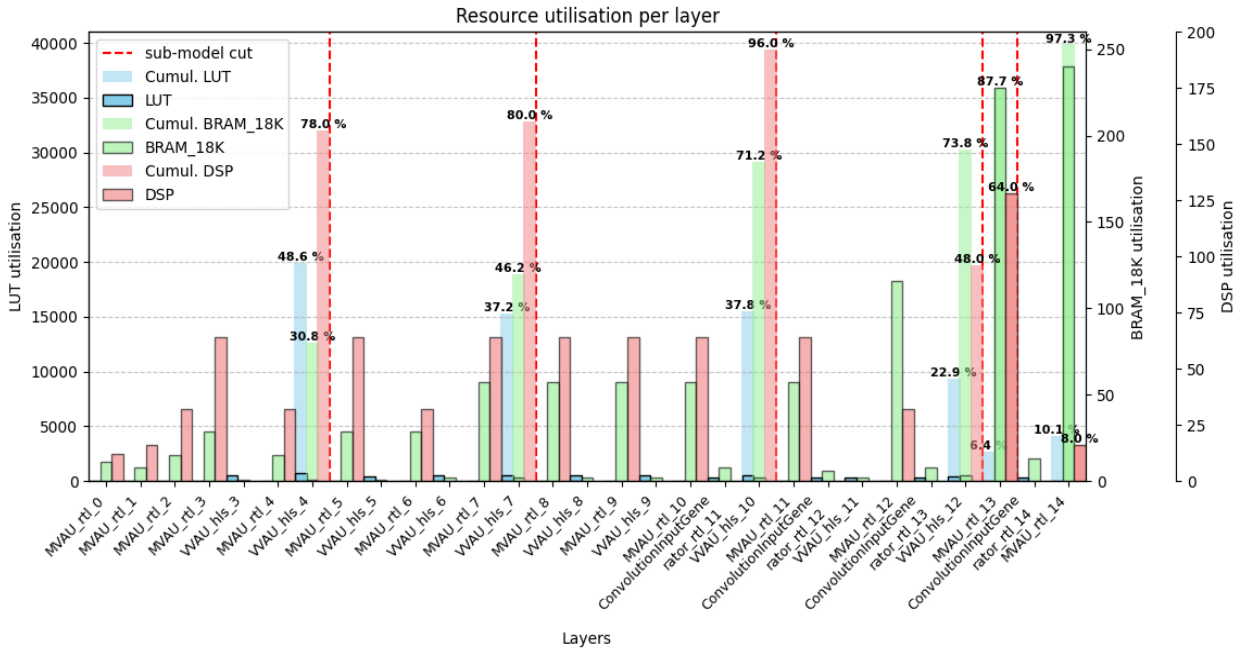


Fig. rs not
requ in this
case.

DNNs are fully executed on the FPGA fabric, with no contribution from the CPU: the ARM processor is only used for data transfers: network communications, input data buffering, and moving data to and from the PL. It currently runs on two threads, one listening for incoming data and storing them in a queue, while the second pops data items from the queue, passes them to the PL and sends the output to the next board. For this study, our system is comprised of 6 Zedboards (using Zynq XC7Z020) connected to a 1 Gbps Ethernet switch. The Zynq is a heterogeneous system with a dual-core processor (ARM Cortex-A9) coupled with an FPGA (XC7Z020-CLG484-1). Each board is set up with an identical version of Petalinux v2024.1. A laptop acts as an orchestrator for initial configuration and as a client sending data to the first board and receiving model inference output from the last (see Fig. 2). Time measurement has been taken client-side to account for all network transit. Power estimations are given by Vivado after the bitstream synthesis stage. In our case, we take the sum of each PS and PL estimates for each board.

B. *MobileNetV1*

MobileNetV1 [13] is a CNN with 3.22 million parameters. The DNN is quantized to 4 bits for weights and activations. We are using the version made available by Xilinx on their FINN example git repository. It has been trained on the ImageNet dataset [14] using input images with a resolution of 224×224×3 pixels. Using our splitting automation script, it resulted in 6 bitstreams which can each fit in a single XC7Z020. In FINN, the folding configuration is the per-layer selection of processing-element (PE) and SIMD-lane counts that affect the level of parallelization of each layer, trading resource use against throughput. Default folding configuration was used, except for the first and last layer of every submodel where, respectively, SIMD and PE were set to two. This is due to the 4 bits quantization of the MobileNetV1. This allows to pack multiple data words into a single byte at DMA level to avoid having half empty bytes at the sub-model output. This halves the network communication overhead. The number of layers in those six submodels is very disparate (31 layers for the first submodel, only 5 for the last submodel). This is due to the high variability in resource consumption between each layer, as shown in Fig. 3.

TABLE 1 COMPARISON TO EXISTING WORKS ON TINY ML ACCELERATION

Work	Hardware	Model	Number of boards	Peak FPS	Total estimate power (W)	FPS/W
Ours	Zedboard	MobileNetV1	6	118.3	16.176	7.31
[15]	Zynq ZCU104	MobileNetV1 + SSD	1	17.85	N/A	N/A
[10]	Alveo U280	MobileNetV1	1	3741	N/A	N/A
[10]	Alveo U280	MobileNetV1	2	5923	152.3	38.82

The disparity is also present in the type of resources used: For this model, using the default folding configuration, targeting a single Zedboard, the total estimated resources yield 126% of its LUTs, 340% of its DSPs and 378% of its BRAMs.

Fig. 4 breaks down the computation and network cost in time. There we can observe that transfer times vary widely. It is due to the difference in input/output sizes of the submodels (up to 100 kb per inference between submodel 1 and 2). Submodels inferences time also widely vary from 6754 µs for submodel 1, to 3422 µs for submodel 6. This could be leveraged with submodels duplication and parallel processing using more boards.

Using our pipelined architecture we are able to achieve 118.3 peak FPS for a batch size of 1 using our neural network splitting technique. Hence, we achieve a throughput sufficient for real-time video classification while maintaining reasonable power consumption.

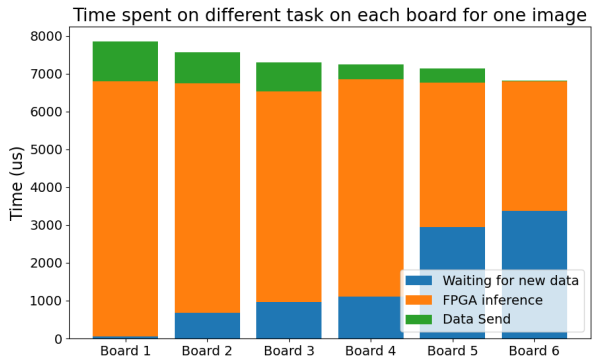


Fig. 4: Average time per task for one inference for each board, waiting/receiving new data, FPGA inference and sending data. In blue, waiting for new data from previous board/client, in orange, data transfer from PS to PL and submodel inference on the FPGA fabric, in green network transfer from the board to the next board/client

IV. Conclusion

In this paper, we have presented a methodology to partition neural networks across several commodity SoC-FPGA systems. This solution supports an arbitrary number of bitstreams and boards. Our first results are promising with a peak throughput of 118.3 inferences per second for MobileNetV1. We stand at a compromise between throughput and power requirements.

It is difficult to compare pure performances between works since most papers use different hardware, slightly different models, may use an object-detection head (SSD) in conjunction, and/or an altogether reworked MobileNet architecture. Our solution is suited to local processing of data with its fairly small power consumption, with a throughput allowing for real-time image processing. We believe it is suitable for environments seeking data privacy and that do not want to rely on cloud services.

Future work includes partitioning larger neural networks to map onto FPGA chips, forming a heterogeneous system of FPGAs to accommodate particularly large layers. This will imply exploring the various tradeoffs to partition such networks, *e.g.*, maximal resource usage per board, the type of resources used (*e.g.*, BRAM vs. LUT-RAM, *etc.*), as well as automatic generation of folding configurations.

References

[1] H. F. Atlam, R. J. Walters, *et al.*, Fog Computing and the Internet of Things: A Review.Big Data and Cognitive Computing. vol. 2, p. 10, June 2018. Number: 2 Publisher: Multidisciplinary Digital Publishing Institute.

[2] H. Li, G. Shou, Y. Hu, *et al.*, "Mobile Edge Computing: Progress and Challenges," in 2016 4th IEEE International Conference on Mobile Cloud Computing, Services, and Engineering (MobileCloud), pp. 83–84, Mar. 2016.

[3] T. Zebin, P. J. Scully, *et al.*, "Design and Implementation of a Convolutional Neural Network on an Edge Computing Smartphone for Human Activity Recognition," IEEE Access, vol. 7, pp. 133509–133520, 2019.

[4] M. Merenda, C. Porcaro, *et al.*, "Edge Machine Learning for AI-Enabled IoT Devices: A Review," Sensors, vol. 20, p. 2533, Jan. 2020. Number: 9 Publisher: Multidisciplinary Digital Publishing Institute.

[5] N. N. Alajlan and D. M. Ibrahim, "TinyML: Enabling of Inference Deep Learning Models on Ultra-Low-Power IoT Edge Devices for AI Applications," Micromachines, vol. 13, p. 851, June 2022. Number: 6 Publisher: Multidisciplinary Digital Publishing Institute.

[6] Y. Umuroglu, N. J. Fraser, *et al.*, "FINN: A Framework for Fast, Scalable Binarized Neural Network Inference," in Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, FPGA '17, (New York, NY, USA), p. 65–74, Association for Computing Machinery, 2017.

[7] M. Courbariaux, I. Hubara, *et al.*, "Binarized Neural Networks: Training Deep Neural Networks with Weights and Activations Constrained to +1 or -1," Mar. 2016. arXiv:1602.02830.

[8] M. Rastegari, V. Ordonez, *et al.*, "XNOR-Net: ImageNet Classification Using Binary Convolutional Neural Networks," in Computer Vision – ECCV 2016 (B. Leibe, J. Matas, N. Sebe, and M. Welling, eds.), (Cham), pp. 525–542, Springer International Publishing, 2016.

[9] M. Blott, T. B. Preußer, *et al.*, "FINN-R: An end-to-end deep-learning framework for fast exploration of quantized neural networks,"ACM Transactions on Reconfigurable Technology and Systems (TRETS),vol. 11, no. 3, pp. 1–23, 2018.

[10] T. Alonso, L. Petrica, *et al.*, "Elastic-df: Scaling performance of dnn inference in fpga clouds through automatic partitioning," ACM Trans. Reconfigurable Technol. Syst., vol. 15, Dec. 2021.

[11] G. Fiscaletti, M. Speziali, *et al.*, "BNNsplit: Binarized neural networks for embedded distributed FPGA-based computing systems," in 2020 Design, Automation & Test in Europe Conference & Exhibition (DATE), pp. 975–978, 2020.

[12] W. Jiang, E. H.-M. Sha, *et al.*, "Achieving Super-Linear Speedup across Multi-FPGA for Real-Time DNN Inference," ACM Trans. Embed. Comput. Syst., vol. 18, pp. 67:167:23, Oct. 2019.

[13] A. G. Howard, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," arXiv preprint arXiv:1704.04861, 2017.

[14] J. Deng, W. Dong, *et al.*, "Imagenet: A large-scale hierarchical image database," in 2009 IEEE conference on computer vision and pattern recognition, pp. 248–255, Ieee, 2009.

[15] A. Sharma, V. Singh, *et al.*, "Implementation of CNN on Zynq based FPGA for Real-time Object Detection," IEEE Internet of Things Journal, pp. 1–7, July 2019.