SURVEY OF FRAMEWORKS FOR INFERENCE OF NEURAL NETWORKS IN SPACE DATA SYSTEMS

Max Ghiglione[1]*, **Vittorio Serra**[1], **Amir Raoofy**[2], **Gabriel Dax**[2], **Carsten Trinitis**[2], **Martin Werner**[2], **Martin Schulz**[2], **Gianluca Furano**[3]

[1]*Airbus Defence & Space GmbH*
[2]*Technical University of Munich*
[3]*European Space Agency*

ABSTRACT

Processing of huge data volumes from remote sensing and Earth Observation (EO) missions requires the development of new generations of space applications and data systems based on Machine Learning (ML). Specifically, the breakthroughs in ML based on Neural Networks (NNs) in the last decade, promise innovative solutions to drive forward and expand the on-board data processing and data systems in the space segment. Therefore, despite their high computational costs, there is an increasing demand for deploying solutions and space services based on NNs. Hence, due to the computational constraints on space missions, efficient hardware solutions for deploying NNs via Field Programmable Gate Array (FPGA)-enabled System on [a] Chip (SoC), in particular Commercial Off-The-Shelf (COTS) solutions, are gaining significant interest.

To investigate this approach, Airbus Defence and Space GmbH is currently leading a research project, Machine Learning Application Benchmark (MLAB), funded by European Space Agency (ESA) General Support Technology Programme. This project aims at developing a ML application benchmark targeting space data systems with a special focus on the deployment of NNs on COTS processing devices.

In this paper, we review frameworks and workflows used for the development of NNs that exhibit a high adherence to the project framework and goals. Specifically, we provide a brief survey of the state-of-the-art tools and frameworks used for the development and deployment of NN models on FPGA-enabled SoCs. Based on the state-of-the-art taxonomy [1] of automated FPGA compilation and design workflows for NN deployment, we classify the deployment frameworks, into Overlay and Dedicated approaches. We further elaborate on these two classes of approaches, compare their characteristics, and discuss implications in exploiting each class in development of space solutions, services, and data systems.
Next, we discuss the deployment tools based on their tradeoffs in various aspects, as well as their ease of programming and openness. Our analyses, discussions, and primary prototypes show that these deployment frameworks provide a wide range of possibilities for deploying NNs to space applications while satisfying the computational restrictions.

*Contact: max.ghiglione@airbus.com

1. INTRODUCTION

During the last two decades, NNs, have gradually gained more popularity in the field of ML due to their generalization capabilities and their possibility to be adapted to various use cases and to be applied to many disparate domains. In particular, Deep Neural Networks (DNNs) experienced significant breakthroughs in the last decade, due to the growing abundance of computational resources, that enabled training and running these models in feasible time frames. Especially, technological advances in High Performance Computing and Cloud services, enabled researchers to develop and train more complex and deeper, hence more accurate NNs on increasingly larger and more sophisticated datasets.

Embedded devices, such as SoCs, are gradually gaining more importance for NN inference, as they enable a mix of data privacy, low latency bounds, and power efficiency and allow for satisfying application-specific constrains. The constrains expressed above are not uniquely limited to earth-bound applications and appear in space-bound applications, as well. In fact, inference in space is only possible on special on-board embedded devices that can deal with challenges almost solely present for space applications, including strict energy budgets, limited compute resources, and severe operating conditions such as exposure to harsh cosmic radiations. In addition, space services and data systems can only rely on on-board processing on such embedded devices for employing inference for modern DNNs in real time because the latency imposed by the roundtrip delay to Earth is a limiting factor. For instance, latency-sensitive tasks such as reliable optical guidance, navigation, and control (GN&C) require on-board inference. Equally importantly, downlink bandwidth limitations do not allow for Cloud-based solutions typically used in other applications, like mobile phones. As an example, bandwidth-heavy EO services can only be efficiently realized by reducing bandwidth

through on-board processing (e.g., inference) on embedded SoCs.

However, these modern DNNs exhibit high computational costs, corresponding to their size and complexity, which render them ill-suited in nature for deployment on on-board embedded devices with limited computational power. Consequently, finding approaches that enable simple, automated, and efficient preparation, and deployment of DNN inference on such devices is a key priority.

On the other hand, with the increased use of NNs, the deployment of machine learning on embedded devices has shifted from a Central Processing Unit (CPU)-based approach towards a NNs accelerator-based one. Commercially used boards based on Graphics Processing Units (GPU) or Tensor Processing Units (TPUs) are still adopted in spacecraft only for very short missions in low earth orbit due to their limited reliability and power efficiency. Spacecraft typically employ CPUs for processing applications, which are available as very reliable ASICs, but are highly inefficient for demanding computations operations like NNs inference. Especially, Convolutional Neural Networks (CNNs) are facing serious bottlenecks in terms of processing parallelism and memory bandwidth on classical microcontroller-based processors. For this reason, for high parallelism applications, like beamforming or compression, spacecraft employ dedicated ASICs or, more recently, SRAM or Flash based FPGAs. FPGA manufacturers have in recent years started to offer tools to deploy machine learning algorithms with more ease, as we will describe in this paper. FPGAs are well suited to be employed as a class of devices on which to deploy NNs accelerators, as they allow for Software-Hardware Co-Design and exploit use of suitable Intellectual Property (IP) blocks specifically designed for particular applications or use cases. Additionally, FPGA-based SoCs are energy efficient and are available in radiation tolerant technologies, which renders them a suitable candidate for NN deployment in space missions.

One of the current trends in high performance FPGAs based platforms is an increased focus of their use as part of SoC devices, in which the flexibility of CPUs is combined with the parallelism of programmable logic. Manufacturers are therefore offering either FPGAs with fabric processors or options to implement softcores like RISC-V, LEON or MicroBlaze, which can be used to host software applications. However, NN deployment to FPGA-based SoCs typically follows a different, and often a more intricate, workflow in comparison to traditional CPU/GPU-based deployment scenarios, which imposes additional complexity for developing space data systems benefiting from NNs. The main root cause for this difference lies in the conceptual difference in programming of FPGA-based devices. Additionally, in many cases the margin of benefit for FPGA deployment relies on exploiting fixed-point arithmetic, which enforces additional complexity in deployment, an occurrence that is typically resolved by exploiting quantized and further optimized models. Therefore, vendors and research communities offer deployment frameworks to address these complexities and to simplify and automate the design and compilation of efficient NN solutions for FPGA-enabled SoCs. These tools exhibit a wide range of flexibility, design and deployment automation as well as efficiency, and therefore, there is still a clear need for conceptual, qualitative, and quantitative comparisons of these frameworks, especially considering the particular application needs and requirements in space data systems.

In this paper, therefore we make the following contributions:

- We provide a survey of the state-of-the-art of workflows for developing CNNs and deploying them to space data systems,
- We classify the deployment frameworks based on compilation and design automation flow to Overlay and Dedicated approaches,
- Using the above classification, we analyze and discuss characteristics of each deployment framework, and
- We discuss the framework based on their tradeoffs, ease of programming, and openness.

2. MODEL DEVELOPMENT TARGETING FPGA-ENABLED SOCS FOR SPACE

Typical use cases of NNs for remote sensing missions include classification, object detection and segmentation. Classical models like Residual [Convolutional Neural] Network (ResNet), You Look Only Once (YOLO), and Unet are examples of DNN families that are commonly employed for these tasks. For example, algorithmic solutions targeting the Airbus wind turbine patch classification, Airbus aircraft detection and Airbus ship segmentation datasets are using the above mentioned models, respectively. We use these models as targets for development and deployment, and to assess the capabilities and limitations of various tools in FPGA deployment.

While in the last couple of years, ML model development (i.e., training) and deployment tools targeting FPGAs had a lot of limitations; it can be observed that the development environment is slowly maturing due to the emergence of active development, interest, and support in the community. Specifically, the support for the prominent development frameworks, e.g., TensorFlow, Pytorch, and Caffe, as development frontends, is becoming commonplace in many workflows. Therefore, FPGA-enabled SoC deployment for space can benefit from standard implementations of particular models (ResNet, YOLO, and Unet) in the above mentioned frameworks. Alternatively, it is also possible to customize the algorithms with layers that are more easily supported by inference tools, which provides sufficient flexibility for the deployment and investigation of future AI-oriented approaches.

Regardless of the deployment workflow and the efficiency of the developed solutions, the standard models that we target in this project are implemented in at least one of the above-mentioned frameworks, which simplifies deployment to COTS systems significantly.

While using the same frontend for model development sounds very promising, in comparison to CPU or GPU deployment, the development workflow for FPGAs has additional overheads that imply a supplemental development effort:

- Deploying a model on FPGAs requires conducting an investigation regarding the supported layers and software components for the selected model within the targeted deployment framework. The state-of-the-art approach for dealing with this limitation relies on either manual and rigorous investigation of NN layers or analysis of errors in downstream deployment workflow.
- Specific versions of the mentioned frameworks need to be exploited as the downstream deployment tools might impose strict version requirements. This implies that using the latest features in the development frameworks, e.g., TensorFlow, as well as the latest models, layers, and formats might not always be a viable option.
- The deployment workflows require to conduct the development and training of the model in a custom frontend framework. For instance, deployment of NN using the Fast, Scalable Quantized Neural Network Inference on FPGAs (FINN) framework—which uses an automated Hardware (HW)/Software (SW) co-design process for deployment of NN on COTS systems—requires porting and training the NN to Brevitas. In such cases, the development and deployment are fully coupled processes, and an extra effort for porting the model into the custom framework is expected in exchange for potential efficiency benefits.

In summary, in many development and deployment frameworks, layer support, model format support, and versioning consistency issues might require redesign, simplification and modification, even re-implementation and eventually re-training of the models in the development framework.

3. FRAMEWORKS FOR DEPLOYING NNS TO FPGAS

The implementation of NNs on FPGAs is a complex task. FPGA-based systems are conventionally programmed using Register-Transfer Level (RTL) and Hardware Description Language (HDL) design tools. However, programming in such low levels of abstraction might be, in many cases (including ML), cumbersome. Additionally, ML developers may lack the knowledge of programming in this level of abstraction, and learning the latter would require an additional effort, which in many cases, might imply dealing with steep learning curves. Therefore, the state-of-the-art deployment tools are designed to provide higher levels of abstraction for programming. Consequently, the deployment workflows typically rely on model development in higher-level interfaces (e.g., TensorFlow in Python) which are compiled and deployed using lower abstractions, that then exploit High Level Synthesis (HLS), model-based autocoding approaches and/or pre-synthesized designs. This deployment and programming scheme are significantly productive as they require minimum RTL-level hardware design knowledge.

The usual workflow starts within the frontend, such as Tensorflow or Pytorch, in order to develop a Deep Learning (DL) application, e.g., aircraft detection or ship detection in a high-level programming tool.
The developed model is then passed through the design and deployment frameworks. This typically involves various steps including quantization of the model, pruning of the graph, and several optimization methods for deployment of the networks on an FPGA.

Next, an inference driver program is prepared using a cross-compilation toolchain and is potentially further optimized with the help of just-in-time (JIT) compilation.
Deployment of the inference model on the target platforms includes setting up the platform themselves, including the Operating System (OS) images, drivers, and runtime libraries to support execution of the application on an FPGA. The new development flows introduced by these frameworks pose major challenges to the space industry, as processes, checklists and standards not always match what can be used in other industries. This will be discussed further in the next paragraphs.

The complexity of NN deployment to FPGAs is not limited to the development of the actual design. Limitations of resources on edge SoCs, and specifically energy consumption and latency requirements of inference in space applications, call for additional steps in the deployment pipeline. Therefore, the deployment frameworks also deal with the additional complexity to employ fixed point arithmetic and sparsity and optimize the networks. However, the complexity of the task is conquered by exploiting automation for deployment in frameworks. Various frameworks proposed by different vendors and researchers take different approaches to automate parts (or all) of the deployment pipeline. With this perspective in mind, we give an overview of the following tools available for NN deployment to FPGA-based SoCs for space applications.

The frameworks and tools that will be compared in this paper are Vitis AI, TVM VTA, Matlab DLP, FINN, hls4ml and Vectorblox. The selection covers the most mature tools, and tries to cover all vendors of processing platforms commonly employed in space for high performance processing. Two main categories are identified and described in the next section. Other tools that have

been successfully employed for neural network inference are mostly model-based tools that involve autocoders, like Matlab C and HDL coders, or high level synthesis tools like Catapult HLS. While certainly a very promising method, they typically involve a high expert knowledge and cannot fully be included. For the purpose of this paper they are considered as standard design tools that can improve the design cycles, but are not specific NNs and are therefore not considered.

4. FRAMEWORK CATEGORIES

Two conceptually different classes of approaches for the deployment of NN on hardware exist:

- Dedicated Design Tools
  (from now on addressed as "dedicated" tools)
- Software Overlay Frameworks
  (from now on addressed as "overlay" tools)

Dedicated design tools have the goal of allowing users a full integration of the workflow, from training to deployment on hardware inside a single framework, while still maintaing a good level of design space exploration options in terms of quantization and hardware mapping. They are typically fully deployed in programmable logic, meaning that they require interfaces or control software only to provide the input data and fetch the results. As the NN layers and weights are stored in hardware, the tool often shows limitations both in terms of model support and training frontends.
From the frontend point of view, the overlay approaches are more mature and support most of the popular DL programming tools, such as Tensorflow and Pytorch. This is because they are first converted in a software representation and then the supported layers are accelerated using processing units, interfaced utilizing specific drivers. On the other hand, only dedicated approaches (specifically FINN) support custom deployment scenarios with quantization- and fault-aware training, which are more suitable for space-bound applications in order to realize high accuracy and fault-tolerance, given the limited computing and memory resources.
As Vitis Artificial Intelligence (AI) is the most mature overlay tool and FINN supports the custom deployment scenarios, in our publication we mainly focus on investigation of Vitis AI and FINN.

As overlay tools typically use programmable logic accelerators, they are typically bound by the vendor to specific processing platforms. Therefore, it is to be noted that Vitis AI is a proprietary tool for Xilinx FPGAs, while Vectorblox is the proprietary Microsemi counterpart. Matlab DLP (which allows the inference of a MathWorks proprietary IP) supports both Xilinx and Intel FPGAs, while Microsemi FPGAs are currently not supported. TVM VTA, instead, is an open source tool, that allows C++ inference and to accelerate the algorithm implementation using vendor specific IPs from Xilinx or Intel. The latter is summarized in Table 1.

Table 1: Frameworks and categories evaluated in this survey

Design Flow	**Support**	**Development**	**Type**
Vitis AI	Xilinx	Proprietary	Overlay
TVM VTA	Xilinx, Intel	Scientific	Overlay
Matlab DLP	Xilinx, Intel	Commercial	Overlay
FINN	Xilinx	Scientific	Dedicated
hls4ml	Xilinx	Open-source	Dedicated
Vectorblox	Microsemi	Proprietary	Overlay

4.1. Dedicated Approaches

In dedicated solutions, IP-cores from a library are configured and connected to produce a specific hardware architecture for every NN. The deployment for dedicated tools typically does not rely on a compiler. The generator of the IP core is not an optimizing compiler, in the sense that it applies generalized transformation passes, but it is able to project a high-level representation in a space closer to the hardware. An example of this is constituted by the FINN-workflow, developed by Xilinx. In standard scenarios, IP core generated by the tools is used solely by one NN. This approach does not result in flexible, generic and reusable solutions, as the generated hardware design is specifically tailored to the NN architecture, thus intrinsically hindering a reprogramming of the accelerator during runtime, as the whole programmable logic has to be reconfigured. On the other hand, this approach is less dependent upon runtime libraries on the processor side, and therefore the accelerator is fully able to read and write to external memories during runtime, without any signaling or driving needs from the host (OS).
Hls4ml and FINN, with its libraries FINN-L (for recurrent LSTM Networks) and FINN-R (for Residual Networks) can be classified among the dedicated hardware solutions. Tools like (Matlab) HDL coder and Catapult HLS, as already introduced previously, are very similar as they allow also for a model that is fully computed in hardware, but have very different design flows as they are not dedicated specifically for neural networks.

The main differences with regard to overlay methods are related to the fact that in such tools the layers are typically converted to an HLS design, allowing for the composition of a stitched-IP for the whole network. Further optimization also happens during synthesis, which can cross layer boundaries.
Dedicated accelerators typically still require to add software or DMAs in order to acquire the data to be processed by the model, but proprietary libraries bound to an operating system, which are typical of overlay approaches can be avoided. While this means that addional design effort in terms of application wrappers is required for this tools, avoiding proprietary libraries is seen as a great advantage, especially in the space sector.

4.2. Overlay Approaches

While the dedicated implementation approaches rely on low-level design and synthesis tools, overlay architectures consist of one or more programmable accelerator cores instantiated in the FPGA fabric to improve the execution time of tensor operation instructions. Overlay approaches adopt the coprocessor-based architecture, where a heterogeneous flow is used to drive the inference computation and accelerate it on an FPGA fabric that functions as an accelerator. The instantiation of a programmable computation engine is controlled by the host processor. The computational engine is able to execute several NN layers in parallel. Usually it consists of a controller and Processing Elements (PE) arrays. A symbiotic relationship exists between the host and the hardware accelerator, which receives instructions from the host and performs them accordingly. The latter involves scheduling single operations in the PE-Array and loading the respective parameters, prior to the beginning of the real computation. In spite of the fact that in many deployment scenarios, existing accelerator designs (e.g., provided by vendors) can be employed, further tuning of the overlay design is often possible and promising: the overlay designs are usually parameterized and can be improved based on the demands of the particular inference workload. This process requires further analysis of the overlay design and the demands of running NNs and resynthesis of the final accelerator implementation. Furthermore, in contrast to dedicated approaches, programming of overlay architectures relies on compilation of the NN to generate instructions to be executed on the cores. It is worthy of notice that, inference execution is typically managed by a trigger application running on the host with the support of runtime libraries and drivers in the operating system.
Vitis AI, Versatile Tensor Accelerator (VTA), and Matlab DL HDL toolbox can be characterized as Overlay approaches.

The degree of flexibility that these architectures present is elevated, due to the fact that if the computation system proofs ill-suited, or unable, to execute a certain operation, as it might be caused by constrains of various nature, the computation can be executed on the accelerator.

4.3. Discussion

Summarizing, all the Overlay approaches support Tensorflow and Pytorch model formats as inputs. Additionally, the deployment of Open Neural Network Exchange (ONNX) models is also straightforward for these options, as it can benefit from standard conversion tools. However, as we discussed before, supporting the model format does not necessarily mean that the deployment of all models with such formats to COTS SoCs is possible: for instance, a particular TensorFlow (.pb) model can encounter problems due to layer support in, e.g., Vitis AI. Table 2 summarizes the nuances of layer-support and quantization offered by every of the analysed tools.

On the other hand, the Dedicated approaches are less flexible in model support: FINN only supports Pytorch (.pt) models that are created using Brevitas as a frontend. High Level Syntheis for Machine Learning (hls4ml), also only supports Keras (.h5) models that are generated with QKeras. In the case of dedicated approaches, the application developer does not need to worry about the support of network layers by the downstream deployment workflow, as the model is trained in the compatible frontend. However, this implies an higher porting effort, as discussed before. In typical development scenarios this implies that when custom models are employed, these have to be redesigned in a new training front like PyTorch.

The output in each deployment workflow is in a unique format, with the exception of VTA and Vitis AI. This is rooted in the fact that VTA uses a similar workflow to Vitis AI. The output model of Vitis AI is an xmodel file which includes the serialized graph object. This object is combined with the Deep [learning] Processing Unit (DPU) IP at a later stage for execution.
Matlab stores the resulting models in a specifically tailored file format, whereas FINN stores the output model as a customized ONNX file with an extension for quantization annotations to support storage of integers with small precision.
The output of hls4ml is an HLS code representation that is later synthesized and deployed.

Overall, regardless of whether a tool is using an Overlay or Dedicated approach for design, the output format is not portable to other platforms.
Overlay approaches use a binary file format that typically is a container for both the host and accelerator specific codes to facilitate execution on SoC platforms. This implies that Overlay approaches would require running an OS along the required drivers and runtime libraries to command the inference acceleration on the FPGA fabric. However, dedicated approaches rely on pure bitstreams, which can be executed without the need for an OS.

5. SUMMARY

This paper presented an overview on the different categories of frameworks for NN inference. In the first place, the need for such tools to assist the deployment of ML applications on space data systems is introduced, with FPGA based SoC being the most promising chips for NN inference on spacecraft. An analysis is then performed on development and deployment of NNs and different frameworks available. The tools are categorized in software overlay and dedicated design tools to identify similarities and differences. The advantages and disadvantages of each workflow are described, with the goal of being able to identify a suitable framework for each use case in future spacecraft missions employing artificial intelligence.

Table 2: Layer and quantization support

Design Flow	Convolutional	Residual	Recurrent	Skip	Quantization
Vitis AI	Hardware	Hardware	Software	Yes	Fixed (8 bit)
TVM VTA	Hardware	Hardware	Software	Yes	Fixed
Matlab DLP	Hardware	Hardware	(Expected)	Yes	Fixed
FINN	Hardware	Hardware	(Discontinued)	No	Custom
hls4ml	Hardware	Hardware	No	No	Custom
Vectorblox	Hardware	Hardware	No	Yes	Fixed (32 bit)

REFERENCES

1. P. Plagwitz, F. Hannig, M. Strbel, C. Strohmeyer, and J. Teich, "A safari through fpga-based neural network compilation and design automation flows," in *2021 IEEE 29th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, 2021, pp. 10–19.

2. "GitHub - onnx/models: A collection of pre-trained, state-of-the-art models in the ONNX format," https://github.com/onnx/models, [Online], visited on 2022-01-30. [Online]. Available: https://github.com/onnx/models

3. M. Blott, T. B. Preuber, N. J. Fraser, G. Gambardella, K. O'Brien, Y. Umuroglu, M. Leeser, and K. Vissers, "FINN-R: An End-to-End Deep-Learning Framework for Fast Exploration of Quantized Neural Networks," *ACM Transactions on Reconfigurable Technology and Systems*, vol. 11, no. 3, sep 2018. [Online]. Available: https://arxiv.org/abs/1809.04570v1

4. MathWorks, "HDL Coder - MATLAB & Simulink," https://de.mathworks.com/products/hdl-coder.html, [Online], visited on 2022-01-30. [Online]. Available: https://de.mathworks.com/products/hdl-coder.html

5. Y. Umuroglu, N. J. Fraser, G. Gambardella, M. Blott, P. Leong, M. Jahre, and K. Vissers, "FINN: A Framework for Fast, Scalable Binarized Neural Network Inference," *FPGA 2017 - Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, pp. 65–74, dec 2016. [Online]. Available: http://arxiv.org/abs/1612.07119http://dx.doi.org/10.1145/3020078.3021744

6. MathWorks, "Deep Learning Processor IP Core - MATLAB & Simulink - MathWorks Deutschland," https://de.mathworks.com/help/deep-learning-hdl/ug/deep-learning-processor-ip-core.html, [Online], visited on 2022-01-30. [Online]. Available: https://de.mathworks.com/help/deep-learning-hdl/ug/deep-learning-processor-ip-core.html

7. A. Shawahna, S. M. Sait, and A. El-Maleh, "FPGA-based Accelerators of Deep Learning Networks for Learning and Classification: A Review," *IEEE Access*, vol. 7, pp. 7823–7859, jan 2019. [Online]. Available: http://arxiv.org/abs/1901.00121http://dx.doi.org/10.1109/ACCESS.2018.2890150

8. V. Rybalkin, A. Pappalardo, M. M. Ghaffar, G. Gambardella, N. Wehn, and M. Blott, "FINN-L: Library Extensions and Design Trade-off Analysis for Variable Precision LSTM Networks on FPGAs," *Proceedings - 2018 International Conference on Field-Programmable Logic and Applications, FPL 2018*, pp. 89–96, jul 2018. [Online]. Available: https://arxiv.org/abs/1807.04093v1

9. T. Moreau, T. Chen, L. Vega, J. Roesch, E. Yan, L. Zheng, J. Fromm, Z. Jiang, L. Ceze, C. Guestrin, and A. Krishnamurthy, "A Hardware-Software Blueprint for Flexible Deep Learning Specialization," *IEEE Micro*, vol. 39, no. 5, pp. 8–16, jul 2018. [Online]. Available: https://arxiv.org/abs/1807.04188v3

10. Xilinx, "Vitis AI," https://www.xilinx.com/products/design-tools/vitis/vitis-ai.html, [Online], visited on 2022-01-30. [Online]. Available: https://www.xilinx.com/products/design-tools/vitis/vitis-ai.html