KServe inference extension for an FPGA vendor-free ecosystem

Diego Ciangottini[1,*], *Giulio* Bianchini[1,2], *Mirko* Mariotti[1,2], *Daniele* Spiga[1], *Loriano* Storchi[3], and *Giacomo* Surace[1]

[1]INFN - National Institute for Nuclear Physics, Via Alessandro Pascoli, Perugia, Italy
[2]Department of Physics and Geology, Alma Mater Studiorum - University of Perugia, Via Alessandro Pascoli, Perugia, Italy
[3]Department of Pharmacy, Alma Mater Studiorum - University of Chieti, Via dei Vestini, Chieti, Italy

Abstract. Field Programmable Gate Arrays (FPGAs) are playing an increasingly important role in the sampling and data processing industry due to their intrinsically highly parallel architecture, low power consumption, and flexibility to execute custom algorithms. In particular, the use of FPGAs to perform Machine Learning (ML) inference is increasingly growing thanks to the development of High-Level Synthesis (HLS) projects that abstract the complexity of Hardware Description Language (HDL) programming. In this work we will describe our experience extending KServe predictors, an emerging standard for ML model inference as a service on kubernetes. This project will support a custom workflow capable of loading and serving models on-demand on top of FPGAs. A key aspect of the proposed approach is to make the firmware generation, often an obstacle to a widespread FPGA adoption, transparent. We will detail how the proposed system automates both the synthesis of the HDL code and the generation of the firmware, starting from a high-level language and user-friendly machine learning libraries. The ecosystem is then completed with the adoption of a common language for sharing user models and firmwares, that is based on a dedicated Open Container Initiative artifact definition, thus leveraging all the well established practices on managing resources on a container registry.

1 Introduction

Field Programmable Gate Arrays (FPGAs) are becoming an increasingly popular solution in Machine Learning (ML) Inference tasks thanks to their highly parallel architecture, customizability and low power consumption [1]. Unlike standard architectures, such as CPUs and GPUs, FPGAs can be used to design specialized hardware circuits tailored for specific ML algorithms, making them more efficient from a computational and energy consumption point of view. Moreover, FPGAs are well-known for their high parallelization that allows them to perform multiple tasks simultaneously with low latency processing times. This capability makes them suitable for real-time applications, where time-sensitive tasks request fast and efficient computation [2]. In particular for ML inference tasks, where latency, parallelism and energy efficiency play a crucial role [3].

*e-mail: diego.ciangottini@pg.infn.it

Furthermore, FPGAs energy efficiency translates into power savings compared to traditional computing devices. This aspect makes them well-suited for deployment in edge devices where energy consumption often represents a bottleneck.

Despite all the advantages the widespread use of FPGAs has been limited by the complexity in programming them via Hardware Description Language (HDL) and, for this reason, a lot of progress has been made in recent years with the development of high-level frameworks. High-Level Synthesis (HLS) [4] frameworks reduce the barriers to FPGA development, allowing for the generation of firmware directly from high-level code [5, 6]. These tools abstract away the hardware complexities, providing a more user-friendly and efficient approach for developers to create firmware.

For all these reasons the aim of the presented work is to move a step forwards the management of the FPGA provisioning and programming chain via a cloud-based model. In particular, the presented work is the result of the combination of two automation layers: the on-demand FPGA firmware creation for ML inference model via the BondMachine [7, 8] technology and the automatic provisioning of a public endpoint for remote inference via the extension of the well-established KServe software stack.

2 Firmware on-demand with BondMachine

The BondMachine (BM) is an Open Source software ecosystem for the dynamical generation of computer architectures that can be synthesized on FPGA. The primary goal of the BM is to empower users to create a hardware system tailored to their specific task requirements. By providing a high-level description of the problem, such as a Go source code or equivalent, users have the ability to construct the optimal BM architecture, encompassing both the FPGA's hardware implementation through HDL code as well as the corresponding binary code to be executed on the hardware. This approach ensures a seamless and efficient hardware solution customized to the user's precise needs.

From an architectural point of view the BM is made up of a set of customised computational units, potentially all different from each other, interconnected and designed to perform one or more tasks. ML inference is one of many tasks for which a custom BM architecture can be used. For this task, the workflow is simple and similar to those of other well-known software like hls4ml [9, 10]. We are here specifically focusing on Neural Network (NN) models, so that the deployment workflow can be reduced to the following steps:

- The user performs the training of a NN model using Python [11] programming language and the TensorFlow framework [12].
- The trained model is subsequently exported to a JSON file, which is essential for NeuralBond (the BondMachine module capable of interpreting this descriptive network file) in order to generate the corresponding BM architecture for the neural network itself.
- Finally, thanks to all the automation mechanisms developed within the BM framework, the architecture will be synthesized into firmware for a specific board among those supported.

The proposed solution offers a high degree of customization in both hardware and software aspects. In fact, regarding the latter factor, various numerical types have been developed within the BM framework enabling several dynamic adjustments of numerical precision based on specific needs and use cases. Furthermore, the BM framework supports both static 32-bit and 16-bit floating-point data types according to the IEEE754 standard [13], as well as dynamic floating point integration via the FloPoCo library [14], which offers a wide range of numerical precision options. As for hardware optimization, BM allows for pruning or collapsing processors based on requirements. While this may result in a loss of computational

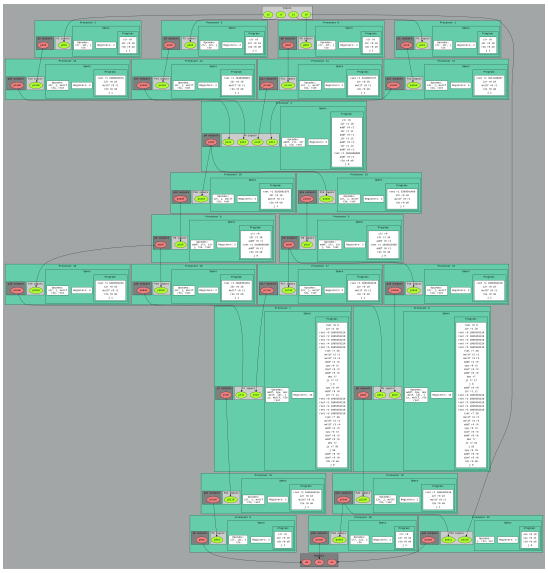

Figure 1. An example of a BM architecture corresponding to a neural network with 4 inputs and 3 outputs

performance on one hand, on the other hand, it implies a reduction in the device's resource usage, that is often a clear bottleneck when integrating complex algorithms on FPGA.

Firmware development often has many challenges: in fact, it not only requires the use of commercial software, which often requires a lot of hardware resources which most of the users cannot access, but also the need of experience in HDL programming.

For the end-users, whose main goal is to solve a high-level computational task and not to focus on low-level aspects, the implementation of their algorithms on a FPGA can be difficult due to their lack of skills and tools. While HLS tools help in the process of converting from high-level code to firmware, they too often require hardware resources that are not available to most users. Because of these difficulties, there exists a compelling and unmet need for an on-demand service that can streamline the firmware generation process for the end-user. By delegating this responsibility entirely to the service, end-users can so fully concentrate on the functionalities and goals of their algorithms. The Firmware-as-a-Service solution we are here presenting will empower users to effortlessly translate their ideas into functional firmware, helping them in implementing their algorithm on FPGA.

The proposed solution consists of two core components: a user-friendly frontend to submit firmware build requests and a backend responsible for processing these requests and generating the firmware. Architecturally, the backend uses the gRPC framework [15], implemented using Python, to provide APIs that facilitate the communication between the frontend and backend systems. The backend service operates on a high-availability host with the necessary hardware resources to handle the firmware generation tasks efficiently. To support this capability, the host is configured with all the packages required for firmware generation. Therefore, it is configured with all the tools from the BondMachine framework, along with its associated dependencies, including the Vivado Design Suite [16].

In Fig. 2 is show the overall workflow implemented with the steps required for bootstrapping the firmware build. Starting from a generic ML model, and before requesting a firmware

build, the client loads the file describing the neural network into a remote storage (step 1 in Fig. 2). The client uploads this file to an S3 bucket and provides its name in the build request (step 2 in Fig. 2)). Once the service verifies the request's parameters for correctness and validity, it proceeds to download the file from the S3 bucket and initiates the firmware build procedure using the specified tool, in this case the BondMachine. Upon completion of the firmware build procedure, which results in the bitstream file, the framework uploads it (step 3 in Fig. 2)), along with the hardware (i.e., hwh) file and a JSON file containing essential build metadata, into an S3 bucket accessible to the end-user. This streamlined process ensures efficient and reliable handling of the NN model related file and firmware generation, facilitating a smooth experience for the client throughout the entire workflow. At the end of the stream, the client will be furnished with a unique identifier (step 4 in Fig. 2). This identifier will grant the client the capability to retrieve both the firmware and its corresponding metadata from the remote storage server (step 5 in Fig. 2).

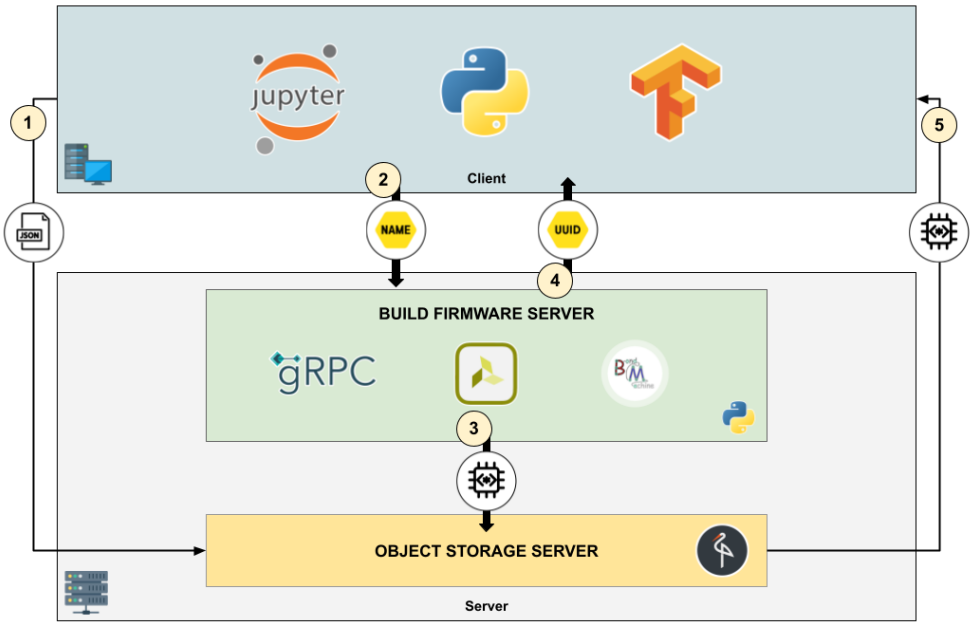

Figure 2. Workflow for the automatic build of a FPGA firmware, starting from a generic ML model, see text for details.

3 Providing ML model remote predictions on FPGA via KServe

Remote machine learning inference is a process where a ML model is hosted on a remote server and is used to perform predictions on data that is sent to the server. It is a suitable solution for a subset of use cases where the model size is too big to be distributed for parallel execution (e.g. a batch system), or when it needs to be shared across multiple users with frequent updates. The extra goal of the approach we are proposing is its capability of managing the access to specialized hardware in a "democratic" and efficient way. That is, developers can build and load their models on a remote storage, and subsequently ask the system to deploy all the needed pieces to get a configured endpoint where to send their input data for the final inference. All of this without a physical access to the machine hosting the FPGA hardware that, in this way, can be shared with other developers in a declarative and cloud-like manner.

Therefore the present work aims to integrate the automated FPGA firmware build, described in the previous section, within the KServe ecosystem. Kserve [17] is an open-source model serving framework that provides a standardized way to deploy and manage ML models in production, and it is part of the KubeFlow [18] ecosystem, a well established platform supporting ML pipeline development. KServe is built to work on top of Kubernetes [19], which makes it a good fit when it comes to abstracting the access to resources with a declarative language. Moreover the choice was lead by a number of features that would allow a generic adoption:

- Model agnostic: support for machine learning models of all types, including TensorFlow, PyTorch [20], and ONNX [21].
- Model versioning: manage different versions of your machine learning models.
- Plugin based: built with extension to custom inference engine in mind (e.g. well define SDK environment for developing new integration).
- Easy WebUI: new model engine can eventually be integrated in a seamless way

As shown in Fig. 3 the idea behind the proposed implementation [22] is to leave the details of the firmware build and load separated. That is motivated by the fact that former step is heavier and more "specialized" than the latter, thus it could need different implementation for different use case. In other words we want users to be able to satisfy two use case:

- Pre-built model: the firmware for a specific board has been already produced and stored on a remote storage. The user just want to load it into an available resource and get predictions back.
- On-demand conversion of a ML model into FPGA: the user has a ML model trained and it requires the automated system (based on the BM framework) to build the firmware for a specific board before loading it into a remote service.

The workflow is thought to be as much transparent as possible to the final user, that can interact from both native Kubernetes command line or via a WebUI, where the barrier provided by Kubernetes specific description language is completely hidden. In fact, a set of APIs that follows the KServe spec has been implemented to make any remote machine able to manage the steps required to get a running service ready to execute prediction on a FPGA board of choice. The new KServe engine will be immediately available to the user as soon as the deployment of the customized inference pod will start forwarding the correct information to the remote machines hosting the FPGA and the auto-build tools. The user is then prompted with a minimum set of parameters: name of the model to be deployed; firmware auto-build engine name (optional: "bondmachine" default); URL of the ML model data (currently only

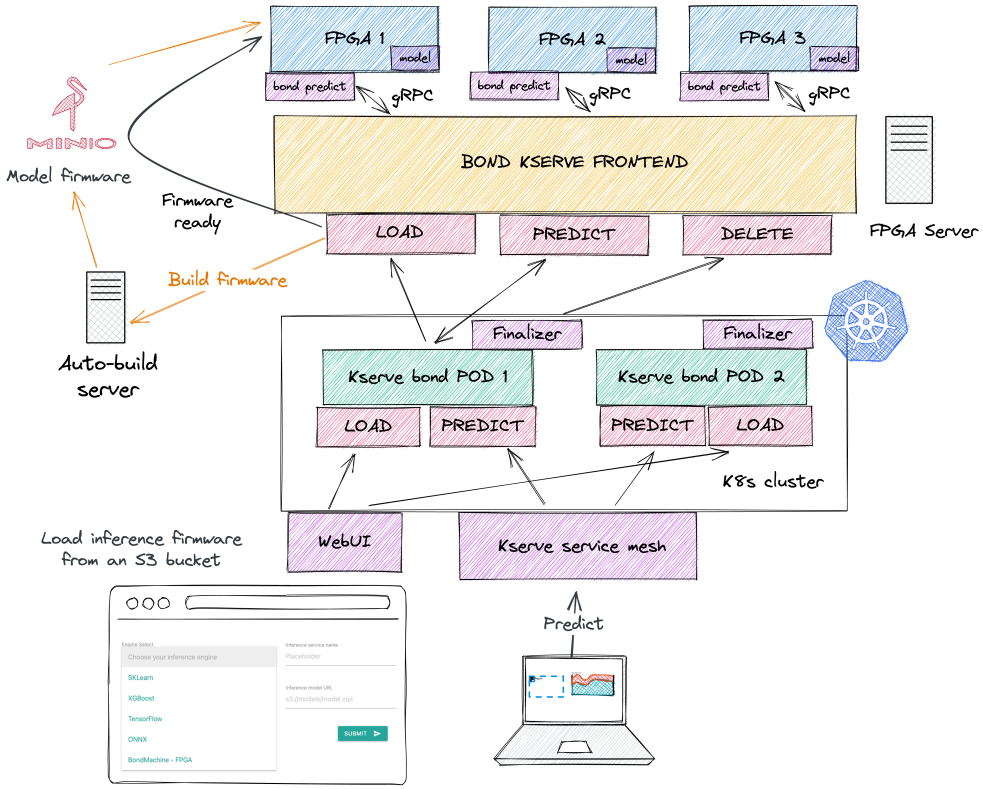

Figure 3. Implementation schema of the different layer of inference-as-a-service stack, including Kserve WebUI and custom inference pod, BondMachine predictor frontend and auto-build module.

HTTP and S3 protocols are supported). After that the user can monitor the status of the deployment from the very same WebUI interface and obtain all the needed information to contact the deployed service for the inference.

An end-to-end deployment of the presented system has been tested running KServe inference services for a demo ML model, which is a fully connected Multi-Layer Perceptron neural network formed by a hidden layer trained on the banknote-authentication dataset taken from OpenML [23].

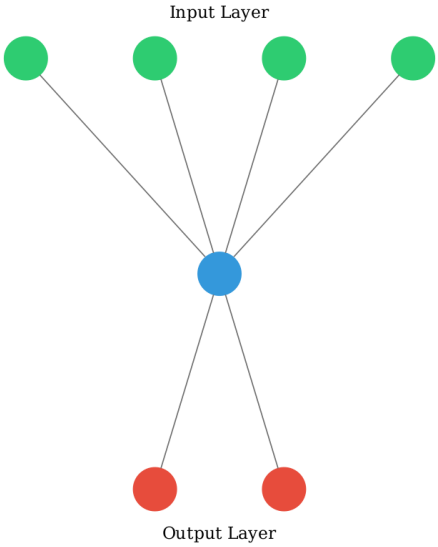

Figure 4. The NN architecture schema of the model used for the validation tests.

Both the automatic build workflow and the direct load of a pre-built firmware have been validate on a heterogeneous set of small/development board but also on ML dedicated ones: namely ebaz4205, zedboard, and alveo u50.

4 Conclusions and plans

In the presented work we demonstrated the possibility to integrate, within one of the most adopted inference-as-a-service platform, a custom logic to make ML prediction on FPGA as easy as possible for a developer. The proposed solution is also able to grant the access to that specialized hardware to multiple users using a cloud-native paradigm.

The natural next step of the presented project is to enhance the ecosystem with additional features; mostly aiming for a cloud-native version control of the models with their firmware and to optimize the resource bookkeeping. Specifically, introducing a Kubernetes resource provider plugin that is aware of the model loaded into the board. This will make the re-use of the same model more efficient across different users.

We aim also to enable interactive development, i.e., load a complete IDE (JupyterLab [24]) with an FPGA attached to, instead of a inference endpoint. Secondly, a monitoring for inference time at different layers (i.e., board cycles, FPGA server and predictor machine and more) is needed. This is particularly useful when optimizing the inference times of a model. Lastly, we will enable the user to store FPGA model as OCI artifact. Thus, the user should be able to define an OCI artifact spec, that will allow for storing model on any compatible container registries together with bitstreams for different boards (this soporific feature it is currently in testing stage).

References

[1] W.F. Samayoa, M.L. Crespo, A. Cicuttin, S. Carrato, IEEE Access (2023)

[2] J. Romoth, M. Porrmann, U. Rückert (2017)

[3] R. Wu, X. Guo, J. Du, J. Li, Electronics **10**, 1025 (2021)

[4] P. Coussy, A. Morawiec, *High-Level Synthesis: From Algorithm to Digital Circuits* (2008)

[5] R.S. Molina, V. Gil-Costa, M.L. Crespo, G. Ramponi, IEEE Access **10**, 90429 (2022)

[6] *Vivado Design Suite User Guide - High-Level Synthesis*, Xilinx Inc. (2020)

[7] M. Mariotti, D. Magalotti, D. Spiga, L. Storchi, Parallel Computing **109**, 102873 (2022)

[8] M. Mariotti, L. Storchi, D. Spiga, D. Salomonie, T. Boccalif, D. Bonacorsid, *The Bond-Machine toolkit: Enabling Machine Learning on FPGA*, in *International Symposium on Grids & Clouds 2019* (2019), p. 20

[9] J. Duarte et al., JINST **13**, P07027 (2018), 1804.06913

[10] FastML Team, *fastmachinelearning/hls4ml* (2023), https://github.com/fastmachinelearning/hls4ml

[11] G. Van Rossum, F.L. Drake, *Python 3 Reference Manual* (CreateSpace, Scotts Valley, CA, 2009), ISBN 1441412697

[12] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G.S. Corrado, A. Davis, J. Dean, M. Devin et al., *TensorFlow: Large-scale machine learning on heterogeneous systems* (2015), software available from tensorflow.org, https://www.tensorflow.org/

[13] IEEE Std 754-2019 (Revision of IEEE 754-2008) pp. 1–84 (2019)

[14] F. de Dinechin, B. Pasca, IEEE Design & Test of Computers **28**, 18 (2011)

[15] A. Kumar, J. Kolhe, S. Ghemawat, L. Ryan, Internet-Draft draft-kumar-rtgwg-grpc-protocol-00, Internet Engineering Task Force (2016), work in Progress, https://datatracker.ietf.org/doc/draft-kumar-rtgwg-grpc-protocol/00/

[16] T. Feist, White Paper **5**, 30 (2012)

[17] *Highly scalable and standards based model inference platform on kubernetes for trusted ai*, https://kserve.github.io/website

[18] *The machine learning toolkit for kubernetes*, https://www.kubeflow.org/

[19] *An open-source system for automating deployment, scaling, and management of containerized applications*, https://kubernetes.io/

[20] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga et al., in *Advances in Neural Information Processing Systems 32* (Curran Associates, Inc., 2019), pp. 8024–8035, http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf

[21] J. Bai, F. Lu, K. Zhang et al., *Onnx: Open neural network exchange*, https://github.com/onnx/onnx (2019)

[22] D. Ciangottini, L. Storchi, M. Mariotti, G. Bianchini, G. Surace, D. Spiga, *KServe inference extension for a FPGA vendor-free ecosystem* (2023), https://github.com/BondMachineHQ/kserve-bond- extension, https://doi.org/10.5281/zenodo.8365556

[23] M. Feurer, J.N. van Rijn, A. Kadra, P. Gijsbers, N. Mallik, S. Ravi, A. Mueller, J. Vanschoren, F. Hutter, arXiv **1911.02490** (2019)

[24] *Free software, open standards, and web services for interactive computing across all programming languages*, https://jupyter.org/