Fast inference of Boosted Decision Trees in FPGAs for particle physics

Sioni Summers[a] , **Giuseppe Di Guglielmo**[b] , **Javier Duarte**[c] , **Song Han**[d] , **Philip Harris**[d] , **Sergo Jindariani**[c] , **Edward Kreinar**[e] , **Vladimir Loncar**[a] , **Jennifer Ngadiuba**[a] , **Maurizio Pierini**[a] , **Ryan Rivera**[c] , **Nhan Tran**[c] , **Zhenbin Wu**[f]

[a] *European Organization for Nuclear Research (CERN), CH-1211 Geneva 23, Switzerland*
[b] *Columbia University, New York, NY 10027, USA*
[c] *Fermi National Accelerator Laboratory, Batavia, IL 60510, USA*
[d] *Massachusetts Institute of Technology, Cambridge, MA 02139, USA*
[e] *HawkEye360, Herndon, VA 20170, USA*
[f] *University of Illinois at Chicago, Chicago, IL 60607, USA*

E-mail: hls4ml.help@gmail.com

Abstract: We describe the implementation of boosted decision trees in the `hls4ml` library, which allows to convert a given model into an FPGA firmware through an automatic high-level-synthesis conversion. Thanks to its full on-chip implementation, `hls4ml` allows to perform inference of BDT models. With a typical latency of $O(1\mu s)$, this solution is relevant for fast real-time processing such as the L1 trigger system of a typical collider experiment.

Contents

1 Introduction

Starting with the work of the MiniBooNE collaboration [1, 2], BDTs have been extremely prevalent within the field of High Energy Physics [3], used mainly for regression and classification tasks, both in event reconstruction and subsequent data analysis. In the high-profile discovery of the Higgs boson [4], BDTs were used to increase the sensitivity of the analysis in the decay channel of the Higgs to two photons [3], and have been used significantly in further analyses of Higgs properties.

Recently, Deep Neural Networks (DNNs) have been investigated as an alternative to BDTs for HEP applications [1], due to their superior performance and the increasing availability of parallel processors capable of high throughput training and inference. Despite the large amount of studies showing interesting use cases for DNN applications, the number of DNN models deployed in the central data procession of the LHC experiments during the second LHC run was very limited. This was mainly due to the lack of optimal deployment solutions that would meet the strong constraints of central processing systems (e.g., real-time event selection in the trigger systems), both in terms of latency and resource footprint.

At the LHC experiments, proton collision occur at such a frequency that the full rate of data cannot be stored. With the LHC delivering collisions every 25 ns, the experiments CMS and ATLAS have to deal with tens of terabytes of data produced each second. Each experiment operates an online data reduction system, called the trigger, to filter out only a fraction events for further analysis. Due to the extreme data rates, this processing must necessarily be extremely fast, and since the rejected events can never be recovered, the selection must be extremely robust.

During the LHC Run II, CMS and ATLAS deployed a two-stage trigger system, starting with the Level-1 Trigger (L1T) performing a first selection, with a second High Level Trigger (HLT) performing

[1] For an extensive discussion of use cases, see Ref. [5] and references therein.

a more refined selection. The L1T must process each LHC event, at the full 40 MHz collision rate, and return its decision within approximately 10 μs, the latency for which the event data can be buffered. Due to these constraints, the L1T is implemented using high speed electronics, consisting of ASICs and FPGAs on custom cards, with high-speed optical interconnects.

Recently, we introduced the `hls4ml` library to facilitate the deployment of DNN models on L1T systems [6]. The aim of that work was to establish an automatic workflow to convert a given DNN model into an electronic circuit, emulated on an FPGA through a fully-on-chip firmware implementation. The workflow consists in converting a given architecture into an opportunely written C++ code, which is then converted to an FPGA firmware by a high level synthesis (HLS) conversion software (e.g., Xilinx Vivado **ref here?**). In Ref. [6], we demonstrated how a DNN model for jet identification at the LHC could be compressed and quantized, to run on an FPGA with 75 ns latency.

In this work, we present an extension of the `hls4ml` library to also support BDTs. The main motivation is the need to support the legacy of the LHC Run II: as of today, BDTs are still the most commonly used ML algorithm for LHC experiments. For instance, the LHCb Collaboration makes extensive use of BDTs (as well as neural networks) in their trigger, which runs in software only. To accelerate the computation, a binned BDT method, Bonsai BDT, is used [7].

Boosted decision trees are still an appealing solution, particularly for use in the earliest processing stages at LHC experiments, thanks to their good performance with relatively low computational cost. The first use case of an ML technique in the L1T of an LHC experiment was a BDT used to perform a regression of muon p_{T} for the CMS L1T endcap muon trigger [8]. The technique gave a three-times reduction in rate for the trigger threshold compared to the previous approach, removing unwanted low p_{T} muons. An external DRAM of 1.2 GB was used as a look-up-table (LUT) to store the pre-computed BDT output for every variation in the input variables. The LUT was filled offline and queried with low latency online. The solution proposed in this paper would allow an on-chip implementation going beyond a full-LUT approach.

Other works have implemented ensembles of Decision Trees for FPGAs [9–13]. These generally target applications of FPGA accelerated inference in a combined CPU-FPGA system, where the relevant performance goals are throughput and energy consumption. Further, the use of external memories and traversal over trees by fetching nodes from memory gives these approaches flexibility and scalability. The work of [9] and [10] in particular is designed to be scalable to very large ensembles in a way that the implementation in this paper is not. In the context of targeting LHC triggers, however, the main performance goal is of extremely low latency, and secondly to maintain a 'reasonable' resource usage.

2 Building boosted decision trees with `hls4ml`

2.1 Case study: jet substructure

In the previous work on translation of neural networks to FPGA firmware with `hls4ml`, the demonstration dataset for discrimination of quarks (q), gluons (g), W and Z bosons, and top (t) jets was presented [6]. The data consist of a set of 16 physics-motivated high-level features, encoding the information on the event jet substructure. With this information at hand, one can distinguish traditional single-prong q and g jets from two- (W and Z) and three-prong jets. This problem is typical

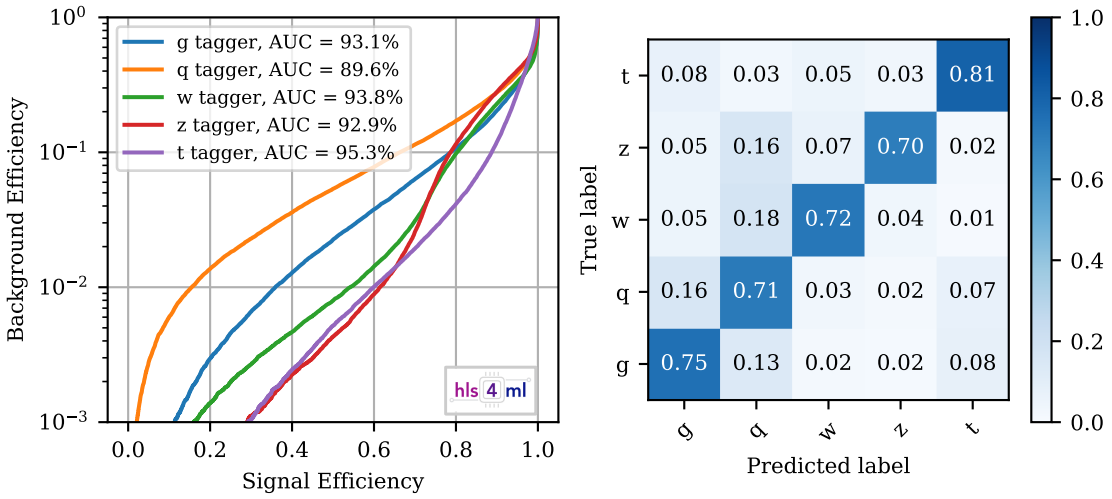

Figure 1: Signal efficiency vs. misidentification rate using a BDT with 100 trees of depth 4 for the five jet classes: gluon, quark, W boson, Z boson, and top quark.

of many searches for physics beyond the standard model at ATLAS and CMS. There is no currently employed algorithm in the L1T systems of these two experiments that exploit this kind of *substructure* information to select events with multi-prong jets.

This dataset provides a benchmark on which to evaluate the classifier performance and its realisation in FPGA implementation as an example application for the L1T. The same dataset is used in this work to prepare a classifier, this time a BDT.

A BDT with 100 trees with a maximum depth of 4 was found to give similar performance to the DNN model trained on the same dataset. The BDT training was performed with the `scikit-learn` package [14]. **we should maybe add here some detail on the training, hyperparameter choice, loss function, and introduce the different BDT scores that define the five classifiers. Should we point to the code on github?**

The resulting receiver operating characteristic (ROC) curve is shown in Figure , displaying the background misidentification efficiency (false-positive rate) as a function of the signal efficiency (true-positive rate) for five jet selectors, defined using the five scores returned by the BDT for the five jet categories. **should we add the confusion matrix?** Overall, the trained BDT reaches state-of-the-art discrimination performance, with a small performance loss with respect to the DNN model of Ref. [6].

should we add some characterization of the model in terms of number of operations that it requires? Maybe comparing it to the DNN to show the trade-off between performance and resources?

This model is used as a benchmark example to show how to use the `hls4ml` to derive an FPGA firmware implementation.

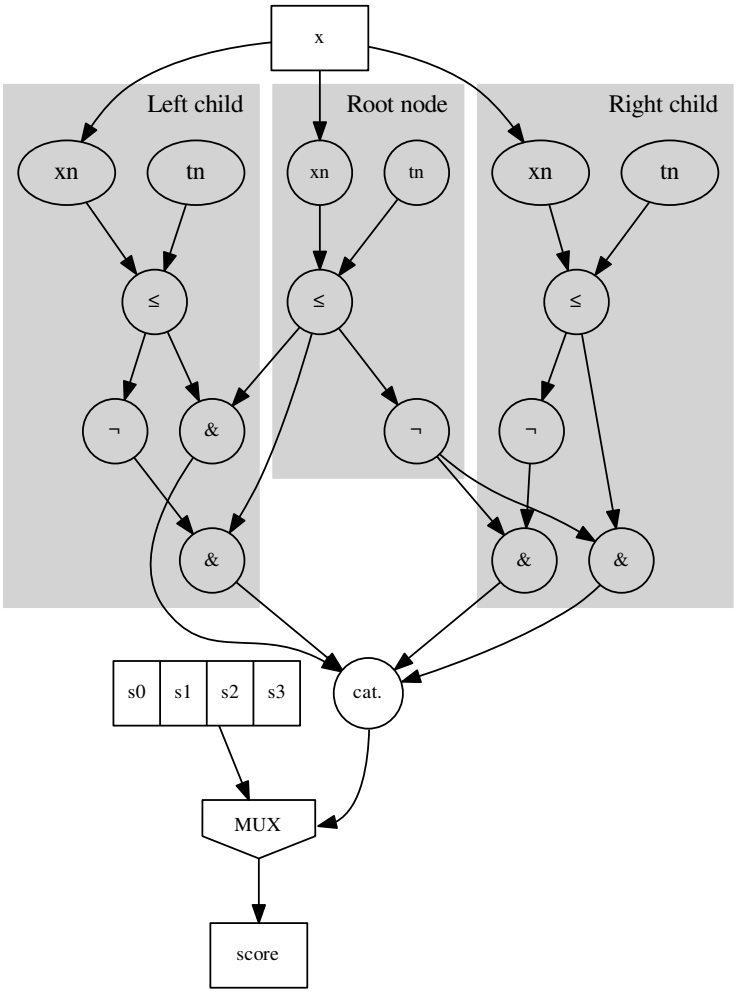

Figure 2: Schematic of the implementation of decision trees in `hls4ml`, showing a single tree with depth of 2. x are the node features, and t the thresholds. The '¬' is the unary 'not' operator, and '&' the binary 'and'. The Boolean leaf activations are concatenated and used to address a look-up-table of output scores.

3 Implementation and Performance

3.1 FPGA Implementation

Decision Trees in `hls4ml` are implemented as an unrolled tree of decisions, as illustrated in Figure 2. Each node in the tree performs a comparison of one of the input features against a constant threshold,

learned in training, and statically fixed during FPGA synthesis. Nodes pass the results of the comparison (true or false) to their children. The decision path is then encoded by the series of Boolean values propagated along the nodes. By construction, only a single leaf node can be activated, and the index of the active leaf is use to address a small look-up-table containing the tree scores for each path.

The score of the BDT ensemble is the sum of score of the decision trees. Since each decision tree is independent, a high degree of parallelisation is possible in the FPGA. The sum is performed with a balanced add-tree. The implementation of BDTs in `hls4ml` targets low latency applications, such as LHC hardware triggers, by executing all trees, and all decisions within each tree, in parallel.

Two code implementations are provided, both targeting the architecture described. The first uses Xilinx's Vivado HLS, written in C++, and the second uses VHDL. The VHDL implementation does not benefit from some of the features of Vivado HLS, such as automatic pipelining depending on the target clock frequency, and easy loop rolling/un-rolling. However, the VHDL implementation synthesises to more reliable results for 'large' BDTs, as will be seen in the Section 3.3. Both implementations are fully pipelined, capable of an 'initiation interval' of 1 clock cycle.

A trained BDT, with specific features, thresholds and scores for each tree, can be evaluated with the FPGA implementation described above using `hls4ml`. Models trained and exported from the `scikit-learn`, `xgboost`, and `TMVA` packages are supported. From the FPGA code produced, which is either using Vivado HLS or VHDL, the user is then able to run the usual FPGA Vendor workflow to integrate the BDT into a specific project and compile to a bitfile.

3.2 Varying the Precision

The generic, programmable logic cells in FPGAs support completely customised data representations. Floating point types are supported, but generally require more resources, latency, and achieve lower clock frequencies than integer types. The fixed point representation uses integer operations, but with a radix point in the number to represent fractional values. In the FPGA, any bitwidth and radix position may be used. A narrower bitwidth will enable smaller resource usage.

The trade-off for using narrower bitwidths is a loss of precision. The loss of discriminating power is investigated by measuring the ratio of the AUC obtained testing with fixed point representation to the AUC from the original floating point, and shown in Figure 3 as a function of the bitwidth, for the benchmark jet-classification BDT introduced in Section 2.1. The number of integer bits was kept at 4 for all bitwidths, as required by the range of the features and scores in the data. A significant reduction in AUC is seen for the smallest width of 6, which allows no fractional bits. The AUC with fixed point variables reaches 99% of the AUC with floating point for all taggers with 11 bits. The consequences in terms of resource savings are discussed in the next section.

3.3 Performance

The FPGA resource utilisation and inference latency is studied using BDTs trained on the jet classification task described in Section 2.1. These performance metrics are expected to vary with the number of trees and their depth. Other hyperparameters, while having an impact on the classification performance, do not affect these FPGA performance metrics. All HLS evaluations of BDTs were

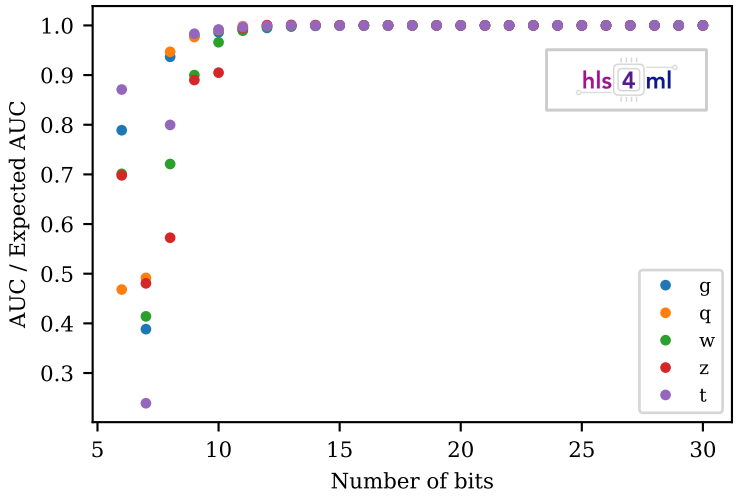

Figure 3: The ratio of Area Under the Curve (AUC) obtained from the fixed point implementation to the AUC expected from the floating point software, as a function of the fixed point bitwidth used, for each of the five tag categories. The ratio saturates at around 15 bits.

built for a Xilinx `vu9p-flgb2104-2L-e` FPGA at 200 MHz target clock frequency. All features, thresholds, and scores were encoded with 18 bits, which is sufficient to achieve identical classification results to the `scikit-learn` original, as will be shown in Section 3.2.

Resource	LUTs	FFs	DSPs	BRAMs
Number Used	96148	42802	0	0
Percentage of VU9P	8.1	1.8	0	0

Table 1: Resource usage of the BDT with 100 trees of depth 4.

The resource utilisation for the benchmark BDT with 100 trees and a depth of 4 is shown in Table 1, where the utilization of look up tables (LUTs), flip-flops (FFs), digital signal processing units (DSPs), and block random access memories (BRAMs). The inference latency for this ensemble is 12 clock cycles, corresponding to 60 nsec execution time at the chosen target clock frequency. This **.... ???? I guess some text is missing here.**

Figure 4 shows the variation with the number of estimators of the BDT, with the depth fixed at 3. For the five-class task of the jet classification dataset, each estimator uses 5 trees - one per class. Only one tree per estimator would be used for a binary classification problem. For the VHDL implementation, the utilisation is reported after synthesis with Vivado. For the HLS implementation, the Vivado HLS resource estimate is reported, as well as the result after synthesising the produced RTL with Vivado. The HLS estimate of LUT and FF usage tend to be larger than the eventual usage after the full synthesis and implementation workflow.

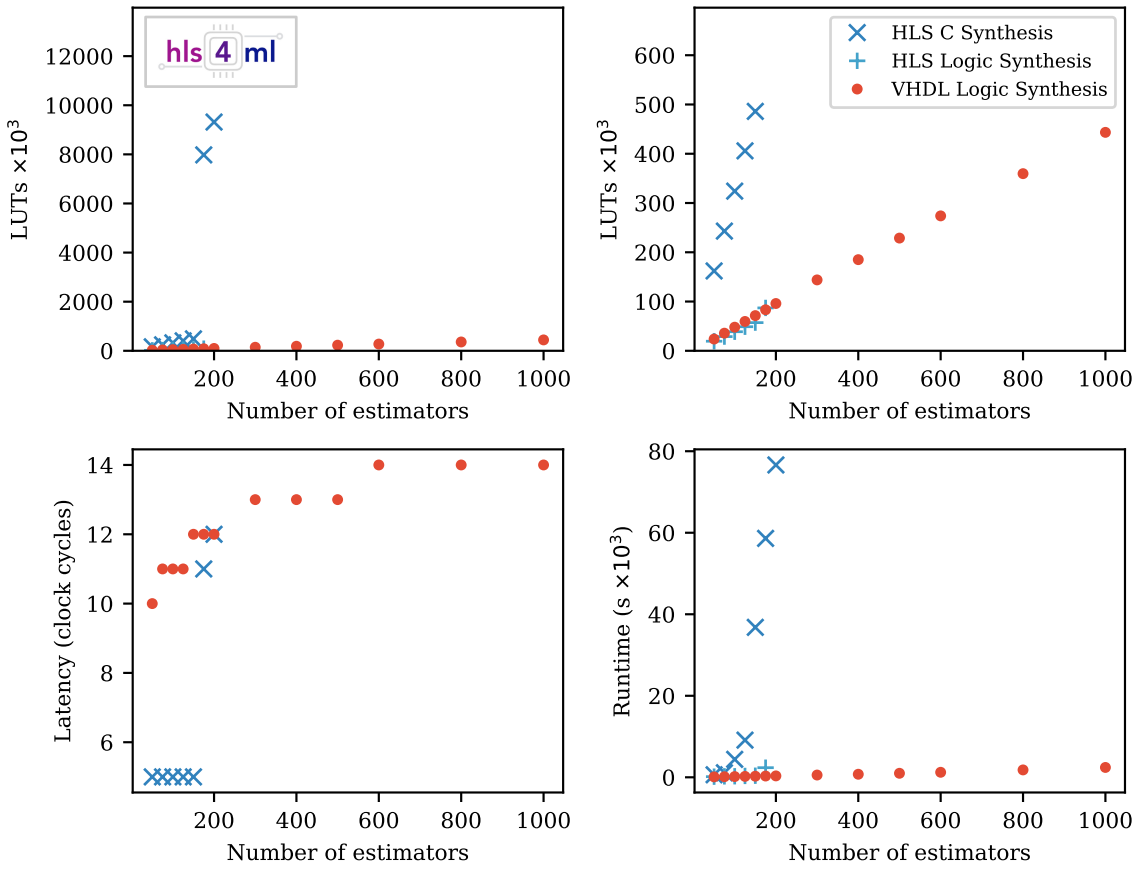

Figure 4: Dependence of LUT usage, inference latency, and synthesis time on the number of estimators (trees) of the BDT, with depth fixed at 3.

Up to $n_{\text{estimators}} = 150$, the LUT utilisation for both implementations increases linearly, with the HLS implementation using slightly fewer than the VHDL version (referring to the utilisation reported after Vivado synthesis). With $n_{\text{estimators}} > 150$, the LUT usage of the HLS implementation increases dramatically, and the Vivado synthesis of the produced RTL also yields poor results. In this regime, the LUT usage of the VHDL implementation continues to increase linearly with $n_{\text{estimators}}$.

The inference latency of the VHDL implementation increases logarithmically with $n_{\text{estimators}}$, as the depth of the balanced add-tree used to sum tree score increases. The HLS implementation latency is more constant, as HLS packs the add-tree into a single cycle for most ensemble sizes. For $n_{\text{estimators}} > 150$ the latency of the HLS result increases significantly. The VHDL implementation latency is typically longer than the latency achieved by the HLS. The VHDL is pipelined to achieve timing closure at higher clock frequencies than the 200 MHz target used for the HLS.

The time taken to synthesise the BDT increases linearly with $n_{\text{estimators}}$ for the VHDL imple-

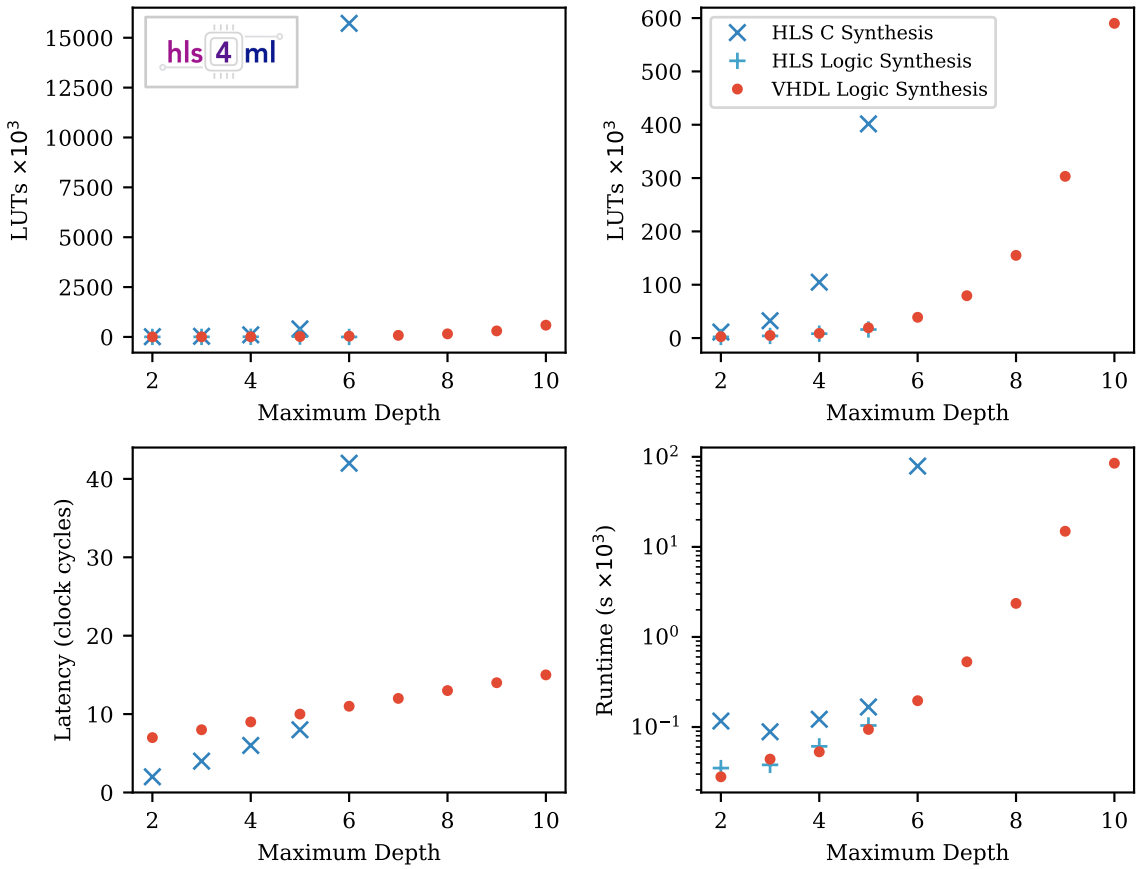

Figure 5: Dependence of LUT usage, inference latency, and synthesis time on the maximum depth of the BDT, with 10 estimators.

mentation, taking 40 minutes for the 1000 estimators ensemble. The HLS C synthesis time increase exponentially with the number of estimators, with synthesis for 200 estimators taking 21 hours. Vivado synthesis times for the HLS RTL output are significantly faster than the HLS C Synthesis which must run before, and increase linearly with the number of estimators.

Figure 5 shows the dependence of the same FPGA performance metrics on the maximum depth of the BDT, with $n_{\text{estimators}}$ fixed at 10. The LUT usage increases exponentially with depth, with each additional layer in the trees adding as many nodes as there above it. As before, the HLS estimate of the LUTs is high compared with the report after synthesising the produced RTL with Vivado. The LUT usage of the VHDL and Vivado-synthesized HLS are very similar, until at maximum depth of 6, the HLS implementation resource usage suddenly increases. At the same point, the latency and synthesis time drastically increase. The latency of the VHDL implementation increases linearly, with one extra clock cycle per depth. Synthesis time increases exponentially with depth, with the synthesis

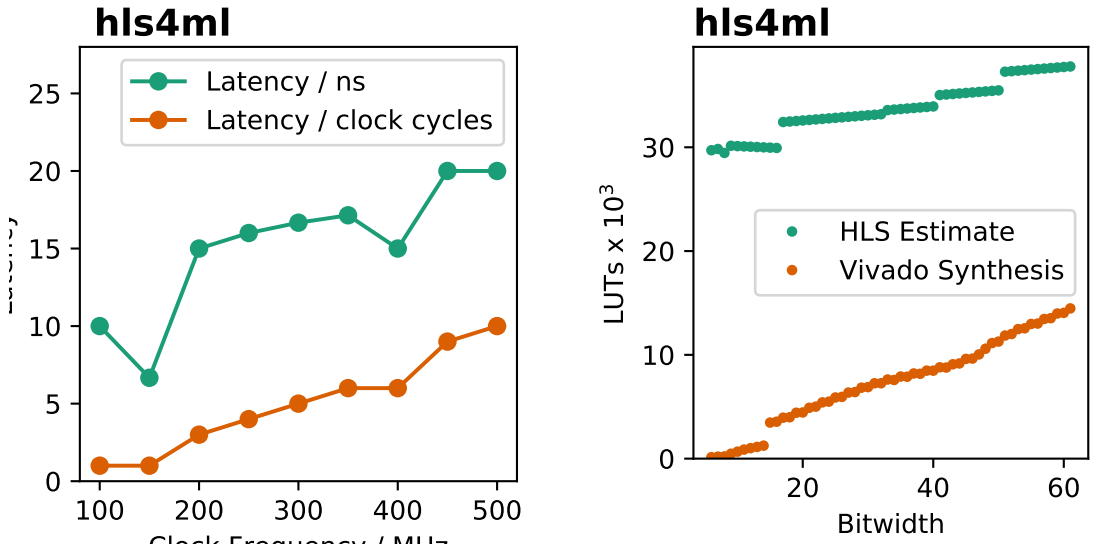

Figure 6: Left: Latency – in clock cycles and nanoseconds – of a jet tagger BDT with 25 trees of depth 3, as a function of the target clock frequency. Right:Resource usage as a function of the bitwidth used for all features, thresholds and scores.**there is some issue with the left image (axis is cut)**

for a depth of 10 taking 27 hours.

Vivado HLS automatically pipelines FPGA designs, according to a target clock period specified by the developer. The `hls4ml` library allows the user to choose a target clock period for the design, when using the HLS backend. Since the design is pipelined, a higher clock frequency allows for a greater inference throughput. Generally, a faster target clock frequency requires more pipeline stages, so it takes more clock cycles to perform inference. **TO BE UPDATED** The left plot of Figure 6 shows the pipeline depth increasing with target clock frequency from 1 clock cycle at 100 MHz to 10 cycles at 500 MHz. The fastest single inference time is 6.67 ns at 150 MHz, and the slowest is 20.0 ns at 500 MHz, although the higher clock frequency will give the best performance when classifying multiple input vectors.

The variation of resource usage with the bitwidth is shown in the right plot of Figure 6 for LUTs, the dominant resource used for BDTs. Four integer bits were used in all cases, as in Figure 3. The increase in resource usage with bitwidth is approximately linear, but with a significant step change transitioning from 14 to 15 bits.

4 Summary and Outlook

We presented the implementation of BDT conversion to FPGA firmware in the `hls4ml` library. Taking as an example a multiclass classification problem from high energy physics (the identification of boosted jets based on substructure information), we show how a state-of-the-art algorithm could be deployed on an FPGA with a typical inference time of 12 clock cycles (i.e., 60 nsec at a clock frequency of 200 MHz). We discuss the optimization options provided by the library, and they

could be used to reduce the resource consumption. We compare an HLS-based implementation to a VHDL one, as a function of the model size. Both the workflows are supported in `hls4ml`. The presented workflow provides a resource effective alternative to Neural Network deployment, which we discussed in a previous publication [6]. This functionality of the `hls4ml` library could support an efficient deployment of algorithms analogues to that described in Ref. [8], which took data at the CMS experiment during the LHC Run II.

Acknowledgements

We acknowledge the Fast Machine Learning collective as an open community of multi-domain experts and collaborators. This community was important for the development of this project.

J. D., B. K., S. J., R. R., and N. T. are supported by Fermi Research Alliance, LLC under Contract No. DE-AC02-07CH11359 with the U.S. Department of Energy, Office of Science, Office of High Energy Physics. P.H. is supported by a Massachusetts Institute of Technology University grant. M. P., S. S., V. L. and J. N. are supported by the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation program (grant agreement n^o 772369). Z. W. is supported by the National Science Foundation under Grants No. 1606321 and 115164.

References

[1] B. P. Roe, H.-J. Yang, J. Zhu, Y. Liu, I. Stancu and G. McGregor, *Boosted decision trees, an alternative to artificial neural networks*, *Nucl. Instrum. Meth.* **A543** (2005) 577 [physics/0408124].

[2] H.-J. Yang, B. P. Roe and J. Zhu, *Studies of boosted decision trees for MiniBooNE particle identification*, *Nucl. Instrum. Meth.* **A555** (2005) 370 [physics/0508045].

[3] A. Radovic, M. Williams, D. Rousseau, M. Kagan, D. Bonacorsi, A. Himmel et al., *Machine learning at the energy and intensity frontiers of particle physics*, *Nature* **560** (2018) 41.

[4] CMS Collaboration, S. Chatrchyan et al., *Observation of a New Boson at a Mass of 125 GeV with the CMS Experiment at the LHC*, *Phys. Lett.* **B716** (2012) 30 [1207.7235].

[5] D. Guest, K. Cranmer and D. Whiteson, *Deep Learning and its Application to LHC Physics*, *Ann. Rev. Nucl. Part. Sci.* **68** (2018) 161 [1806.11484].

[6] J. Duarte et al., *Fast inference of deep neural networks in FPGAs for particle physics*, *JINST* **13** (2018) P07027 [1804.06913].

[7] V. V. Gligorov and M. Williams, *Efficient, reliable and fast high-level triggering using a bonsai boosted decision tree*, *JINST* **8** (2013) P02013 [1210.6861].

[8] CMS Collaboration, D. Acosta et al., *Boosted Decision Trees in the Level-1 Muon Endcap Trigger at CMS*, *J. Phys. Conf. Ser.* **1085** (2018) 042042.

[9] M. Owaida, H. Zhang, C. Zhang and G. Alonso, *Scalable inference of decision tree ensembles: Flexible design for CPU-FPGA platforms*, in *2017 27th International Conference on Field Programmable Logic and Applications (FPL)*, pp. 1–8, Sep., 2017 DOI.

[10] M. Owaida, A. Kulkarni and G. Alonso, *Distributed Inference over Decision Tree Ensembles on Clusters of FPGAs*, *ACM Trans. Reconfigurable Technol. Syst.* **12** (2019) 17:1.

[11] M. Barbareschi, S. Del Prete, F. Gargiulo, A. Mazzeo and C. Sansone, *Decision Tree-Based Multiple Classifier Systems: An FPGA Perspective*, 06, 2015 DOI.

[12] S. Buschjäger and K. Morik, *Decision Tree and Random Forest Implementations for Fast Filtering of Sensor Data*, *IEEE Transactions on Circuits and Systems I: Regular Papers* **65** (2018) 209.

[13] R. Kułaga and M. Gorgon, *FPGA Implementation of Decision Trees and Tree Ensembles for Character Recognition in Vivado Hls*, *Image Processing & Communications* **19** (2014) .

[14] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel et al., *Scikit-learn: Machine Learning in Python*, *Journal of Machine Learning Research* **12** (2011) 2825.