Journal of Instrumentation

SISSA

Fast inference of Boosted Decision Trees in FPGAs for particle physics

To cite this article: S. Summers *et al* 2020 *JINST* **15** P05026

Published by IOP Publishing for Sissa Medialab

Received: *February 20, 2020*
Accepted: *April 7, 2020*
Published: *May 29, 2020*

Fast inference of Boosted Decision Trees in FPGAs for particle physics

S. Summers,[a,1] **G. Di Guglielmo,**[b] **J. Duarte,**[c] **P. Harris,**[d] **D. Hoang,**[e] **S. Jindariani,**[f] **E. Kreinar,**[g] **V. Loncar,**[a,h] **J. Ngadiuba,**[a] **M. Pierini,**[a] **D. Rankin,**[d] **N. Tran**[f] **and Z. Wu**[i]

[a] *CERN, Esplanade des Particules 1, Geneva 23 1211, Switzerland*
[b] *Department of Computer Science, Columbia University, 500 West 120 Street, New York, NY 10027, U.S.A.*
[c] *Department of Physics, University of California San Diego, 9500 Gilman Dr., La Jolla, CA 92093, U.S.A.*
[d] *Department of Physics, Massachusetts Institute of Technology, 77 Massachusetts Avenue, Cambridge, MA 02139, U.S.A.*
[e] *Department of Physics, Rhodes College, 2000 North Parkway, Memphis, TN 38112, U.S.A*
[f] *Fermi National Accelerator Laboratory, Batavia, IL 60510, U.S.A.*
[g] *HawkEye360, Herndon, VA 20170, U.S.A.*
[h] *Institute of Physics Belgrade, Pregrevica 118, Belgrade, Serbia*
[i] *Department of Physics, University of Illinois at Chicago, W. Taylor St., Chicago, IL 60607, U.S.A.*

E-mail: sioni.summers@cern.ch

Abstract: We describe the implementation of Boosted Decision Trees in the `hls4ml` library, which allows the translation of a trained model into FPGA firmware through an automated conversion process. Thanks to its fully on-chip implementation, `hls4ml` performs inference of Boosted Decision Tree models with extremely low latency. With a typical latency less than 100 ns, this solution is suitable for FPGA-based real-time processing, such as in the Level-1 Trigger system of a collider experiment. These developments open up prospects for physicists to deploy BDTs in FPGAs for identifying the origin of jets, better reconstructing the energies of muons, and enabling better selection of rare signal processes.

Keywords: Analysis and statistical methods; Data processing methods; Digital electronic circuits; Trigger algorithms

ArXiv ePrint: 2002.02534

[1] Corresponding author.

https://doi.org/10.1088/1748-0221/15/05/P05026

2020 JINST 15 P05026

Contents

1 Introduction **1**

2 Building Boosted Decision Trees with `hls4ml` **3**

3 Implementation and performance **4**
- 3.1 FPGA implementation 4
- 3.2 Varying the precision 5
- 3.3 Performance and cost 6
- 3.4 Resource model 8
- 3.5 Varying clock frequency and precision 9

4 Summary and outlook **11**

2020 JINST 15 P05026

1 Introduction

Starting with the work of the MiniBooNE collaboration [1, 2], Boosted Decision Trees (BDTs) have been extremely prevalent within the field of High Energy Physics (HEP) [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, BDTs were used to increase the sensitivity of the CMS analysis in the decay channel of the Higgs to two photons [4], and have been used significantly in further analyses of Higgs properties.

At the Large Hadron Collider (LHC) experiments, proton collisions 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 of 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 highly robust.

The CMS and ATLAS experiments deploy a two-stage trigger system, starting with the Level-1 Trigger (L1T) performing a first selection, with a second High Level Trigger (HLT) performing 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, 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

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

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 processing of the LHC experiments during previous LHC running 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 computing resource footprint.

Previously, 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, evaluated on an FPGA through a fully-on-chip firmware implementation. The workflow consists of converting a given NN model into an expertly written C++ code, which is then converted to an FPGA firmware by a High Level Synthesis (HLS) tool (e.g., Xilinx Vivado HLS). 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.[2] As shall be seen in the following sections 2 and 3, the BDT implementation in FPGAs is capable of achieving similar performance to a DNN, with a relatively lightweight usage of device resources. The critical FPGA resource for BDTs is Look Up Tables (LUTs), whereas the availability of DSPs for multiplication is the limiting factor for DNNs. Given this, the BDT can be seen as a lightweight solution which is complementary to a DNN.

Another motivation for the introduction of BDTs 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].

BDTs remain a particularly appealing solution 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 (BDTs and Random Forests) 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 modest resource usage.

[2]The project code can be accessed at: https://github.com/hls-fpga-machine-learning/hls4ml/tree/bdt.

2020 JINST 15 P05026

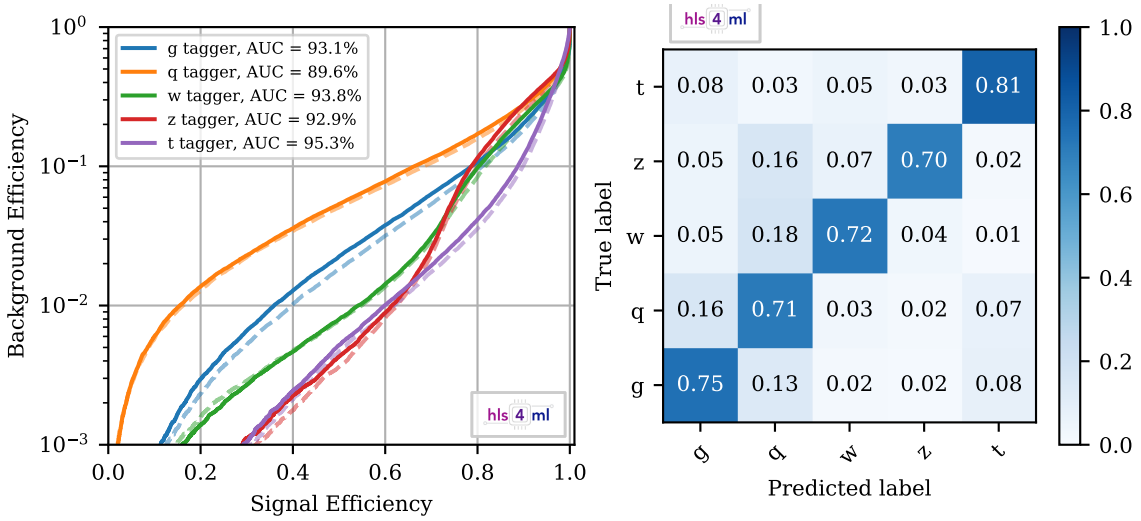

Figure 1. Left: the solid curves show signal efficiency vs. isidentification rate using a BDT with 100 trees of depth 4 for the five jet classes: gluon, quark, W boson, Z boson, and top quark. The dashed curves show the performance of the 3 layer MLP from [6]. Right: confusion matrix for the BDT.

2 Building Boosted Decision Trees with `hls4ml`

In the previous work on translation of neural networks to FPGA firmware with `hls4ml`, we presented a demonstration data set for discrimination of quarks (q), gluons (g), W and Z bosons, and top (t) jets [14]. The data consist of a set of 16 physics-motivated high-level features, representing information of 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 of searches for physics beyond the standard model at ATLAS and CMS. To our knowledge there is no algorithm currently employed in the L1T systems of these two experiments that exploits this kind of *substructure* information to select events with multi-prong jets. This data set provides a benchmark on which to evaluate the classifier performance and its realisation in FPGA implementation as an example application for the L1T. We use the same data set in this work to prepare a classifier, this time a BDT.

We performed the BDT training using the `scikit-learn` package [15], randomly splitting the data set into training (80%) and testing (20%) partitions. A BDT with 100 estimators and a maximum depth of 4 was found to give similar performance to the DNN model trained on the same data set, providing a useful point of comparison. The cross-entropy loss function was used.

The resulting receiver operating characteristic (ROC) curve is shown in figure 1, 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. 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].

The operations used in inference of a BDT are very different from those used for a neural network. While a (fully connected) neural network comprises a series of matrix-vector products and evaluations of non-linear activation functions, the BDT inference involves evaluating decision paths

2020 JINST 15 P05026

over many decision trees. This tree traversal requires comparisons against thresholds, effectively partitioning the feature space. In terms of the number of parameters, the trained BDT with 100 estimators of depth 4, and 5 classes, is summarised by 7,500 threshold values, and 8,000 scores. The fully connected neural network presented in [6], with the same 16 inputs, 5 outputs, and three hidden layers of 64, 32, and 32 neurons, has 4,389 trainable parameters. A BDT is only able to make cuts orthogonal to the feature axes, while the activation functions of a neural network add non-linearity to the classification.

We use this model as a benchmark example to show the use of `hls4ml` to derive an FPGA firmware implementation.

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. These thresholds are statically fixed in the logic of the FPGA firmware, rather than being fetched from an external memory. 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. These scores map to the probability that the given input features correspond to a certain class. For multiclass classification, each 'estimator' uses as many decision trees as the number of classes, while only one tree per estimator would be used for a binary classification problem.

The score of the BDT ensemble is the sum of scores of all 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 adder tree, reducing the scores to their sum in a pair-wise tree structure. 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.

We developed two code implementations, both targeting the architecture described. The first uses Xilinx's Vivado HLS, written in C++, and the second is developed at the Register-Transfer Level (RTL), using VHDL.[3] Generally, an RTL 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 RTL 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` [17], and `TMVA` [18] 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] For an introduction to FPGA concepts and terminology, see ref. [16].

2020 JINST 15 P05026

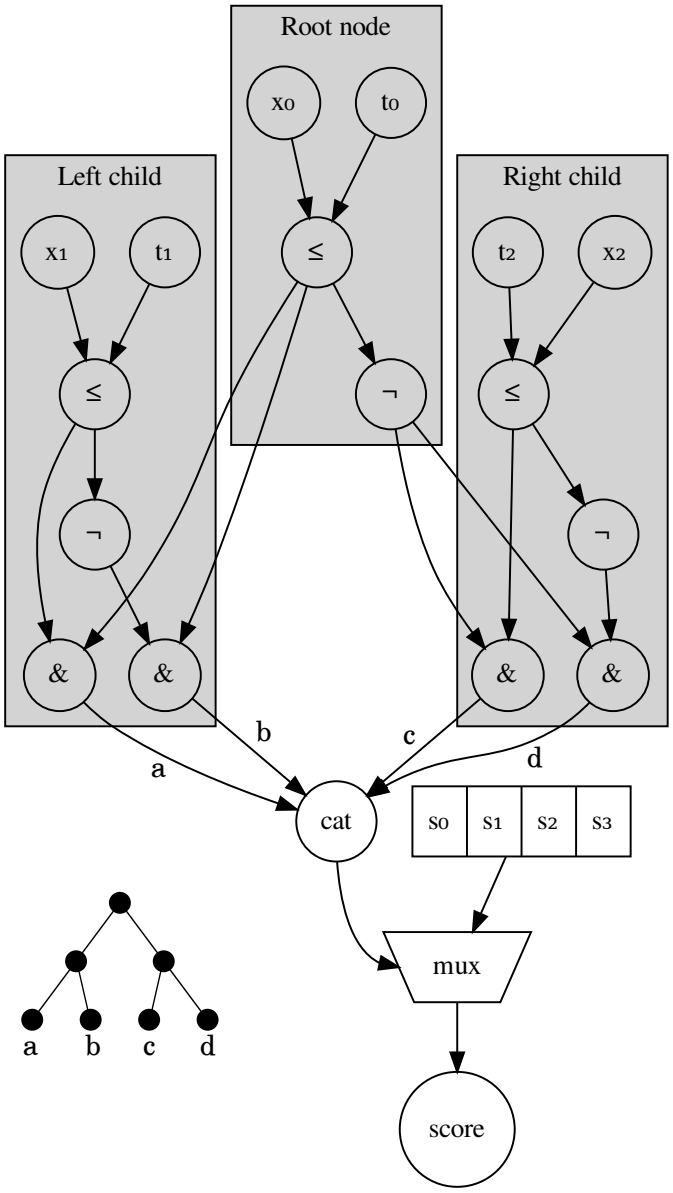

Figure 2. Schematic of the implementation of decision trees in `hls4ml`, showing a single tree with depth of 2. The x are the node features, and t the thresholds. The '¬' is the unary 'not' operator, and '&' the binary 'and'. The Boolean leaf activations are concatenated and used to address a look-up-table of output scores. The labels 'a', 'b', 'c', and 'd' on the schematic correspond to the respective labelled leaf nodes of the tree represented at the bottom left.

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.

2020 JINST 15 P05026

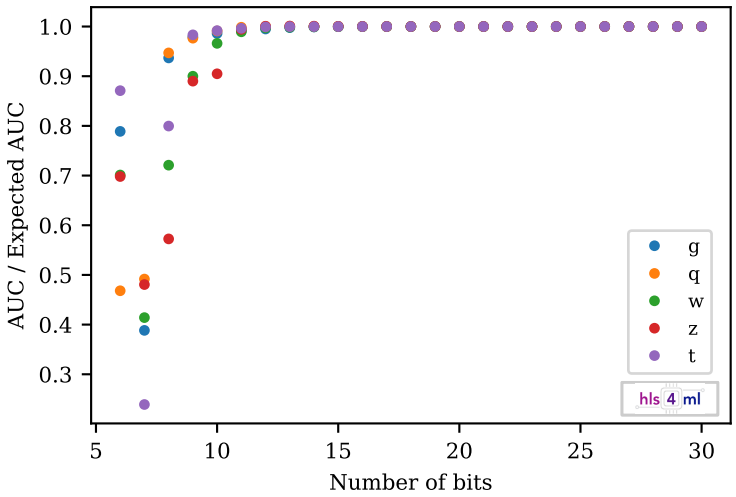

Figure 3. The ratio of Area Under the Curve (AUC) obtained from the fixed point implementation to the AUC expected from the floating point software, as a function of the fixed point bitwidth used, for each of the five tag categories. The ratio saturates at around 15 bits.

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 area under the ROC curve (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. 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 to avoid overflow. A significant reduction in AUC is seen for bitwidths of 10 and below. The AUC with fixed point variables reaches 99% of the AUC with floating point for all taggers with 11 bits. Below 10 bits the behaviour becomes unstable, as numerical rounding effects cause unpredictable misclassification. The consequences of the bitwidth on resource usage are discussed in the next section.

3.3 Performance and cost

We studied the FPGA resource utilisation and inference latency using BDTs trained on the jet classification task described in section 2. These 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 built for a Xilinx `vu9p-flgb2104-2L-e` FPGA at 200 MHz target clock frequency. FPGAs of this size or similar could be used in future LHC upgrades, and would generally be used to execute several algorithms (including feature pre-processing) as well as any ML inference. All features, thresholds, and scores were encoded with 18 bits, which is sufficient to achieve identical classification results to the `scikit-learn` original, as was shown in section 3.2.

The resource utilisation of LUTs, FFs, DSPs, and BRAMs for the benchmark BDT with 100 estimators and a depth of 4 is shown in table 1. This utilisation is reported after running the logic synthesis step with the VHDL implementation. The inference latency for this ensemble is 12 clock cycles, corresponding to 60 ns execution time at the chosen target clock frequency. This is compatible with the requirements for use in the L1T system.

2020 JINST 15 P05026

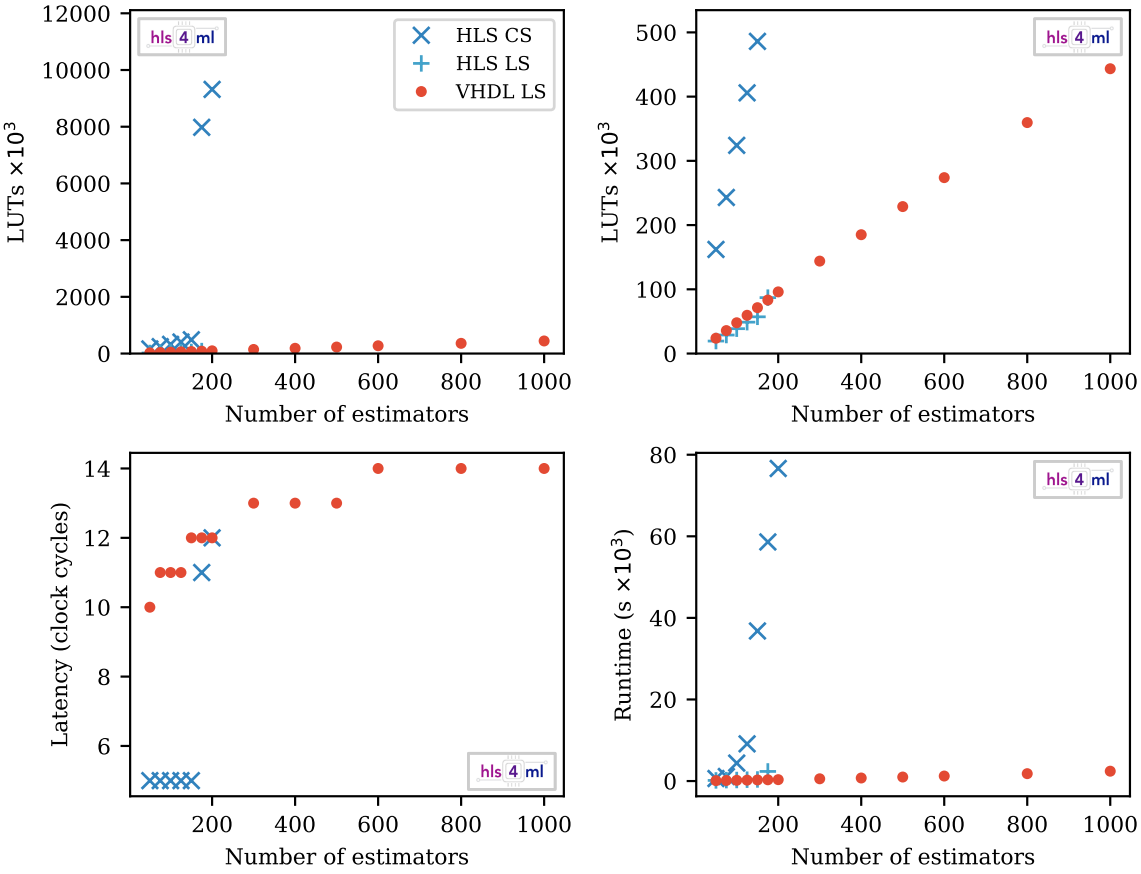

Figure 4. Dependence of LUT usage (top row), inference latency (bottom left), and synthesis time (bottom right) on the number of estimators of the BDT, with depth fixed at 3. The top right plot is a view of the same data as the top left, with reduced range. Different stages of synthesis are shown: C-Synthesis estimate of HLS (HLS CS), utilisation report after Logic Synthesis step of RTL produced by HLS (HLS LS), and utilisation report after Logic Synthesis of the VHDL implementation (VHDL LS).

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

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

Figure 4 shows the variation in resource usage with the number of estimators, n_e, of the BDT, with the depth fixed at 3. Each estimator uses as many trees as the number of classes, in this case of the jet classification dataset, five. Only one tree per estimator would be used for a binary classification problem, reducing the resource cost by a factor five in such cases. For the VHDL implementation, the utilisation is reported after logic synthesis with Vivado. For the HLS implementation, the Vivado HLS resource estimate after C-synthesis is reported, as well as the result after executing logic synthesis on 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. Pipelining of the HLS implementation is determined by the Vivado HLS compiler during

2020 JINST 15 P05026

the C-synthesis step, placing registers optimally to achieve timing closure. The latency of the HLS implementation is set during this step, and not affected by later execution of logic synthesis.

Up to $n_e = 150$, 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 $n_e > 150$, 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_e.

The inference latency of the VHDL implementation increases logarithmically with n_e, as the depth of the balanced adder tree used to sum tree score increases. The HLS implementation inference latency is more constant, as HLS packs the adder tree into a single cycle for most ensemble sizes. For $n_e > 150$ 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_e for the VHDL implementation, taking 40 minutes for the 1000 estimators ensemble. The HLS C synthesis time increases 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_e fixed at 10. The LUT usage increases exponentially with depth, with each additional layer in the trees adding as many nodes as there are 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 for a depth of 10 taking 27 hours.

3.4 Resource model

Given the expected scaling of LUTs with model hyperparameters — linear with n_e and exponentially (base two) with depth — we analytically describe the resource usage using the following relation:

$$r = k_0 \cdot n_e + k_1 \cdot n_e \cdot 2^d,$$

where r is the resource usage (LUTs), n_e the number of estimators, d the tree depth, and k_0, k_1 are unknown constants. The term linear in n_e represents the resource of the adder-tree which grows with the number of trees. The term linear in n_e and exponential with d represents the logic used for the trees, of which there are n_e, while the number of decision nodes doubles at each layer in depth. Other hyperparameters — such as the loss function, learning rate, and number of features — may impact the classification performance of the model, but would not affect the resource usage. A fit to the measurements of trained and synthesised BDTs using the VHDL implementation was performed, yielding:

$$r = 22 \cdot n_e + 53 \cdot n_e \cdot 2^d.$$

2020 JINST 15 P05026

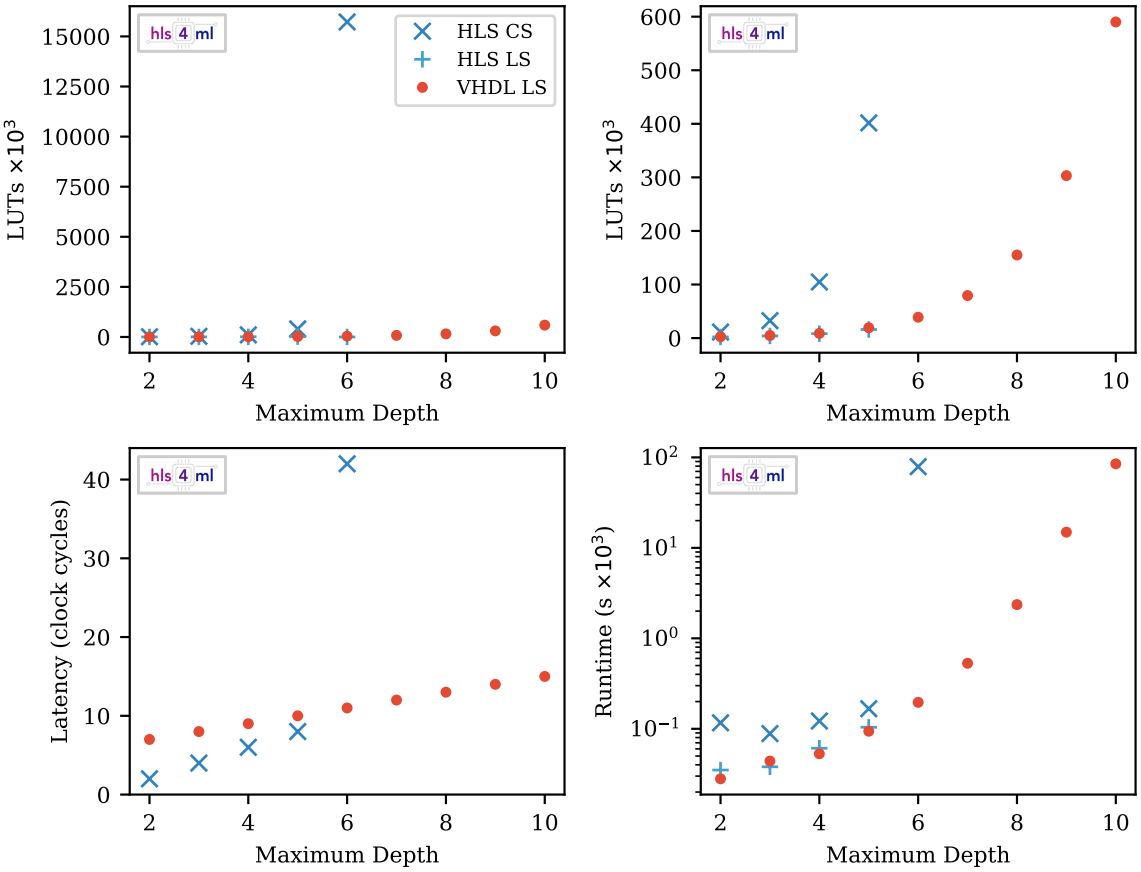

Figure 5. Dependence of LUT usage (top row), inference latency (bottom left), and synthesis time (bottom right) on the maximum depth of the BDT, with 10 estimators. The top right plot is a view of the same data as the top left, with reduced range. Different stages of synthesis are shown: C-Synthesis estimate of HLS (HLS CS), utilisation report after Logic Synthesis step of RTL produced by HLS (HLS LS), and utilisation report after Logic Synthesis of the VHDL implementation (VHDL LS).

All features, thresholds and scores were encoded with 18 bits. Figure 6 shows this scaling model over the measured BDT results used for the single-parameter scans in figures 4 and 5, showing good agreement.

3.5 Varying clock frequency and precision

Vivado HLS automatically pipelines FPGA designs, according to the target clock period specified by the developer. When using the HLS workflow, the `hls4ml` library allows the user to choose a target clock period for the BDT model. Generally, a faster target clock frequency requires more pipeline stages, so more clock cycles will be needed to perform the inference. The left plot of figure 7 shows the pipeline depth increasing with target clock frequency from 6 clock cycles at 100 MHz to 29 cycles at 500 MHz. The single inference latency in nanoseconds (the product of the latency in clock cycles and the clock period) is relatively constant with the target clock frequency. The lowest single inference latency is 52 ns at 250 MHz while the highest is 62.2 ns at 450 MHz. Using a higher clock frequency will achieve overall faster inference when classifying several input feature vectors, since the initiation interval is 1 in all cases.

2020 JINST 15 P05026

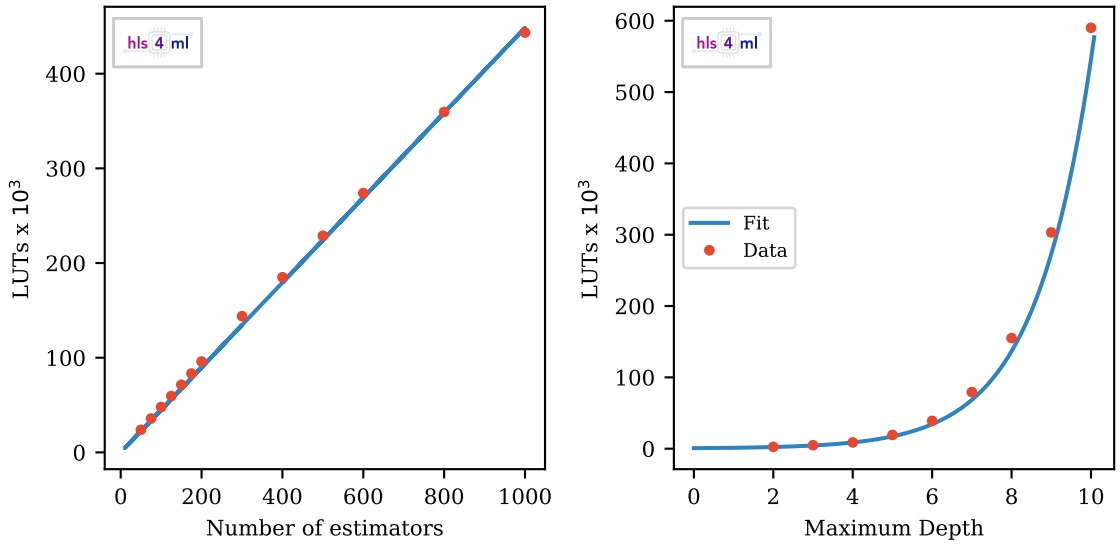

Figure 6. Comparison of LUT usage between measured results (points) and resource usage model (curve). Left: as a function of number of estimators with depth fixed at 4. Right: as a function of the depth, with number of estimators fixed at 10.

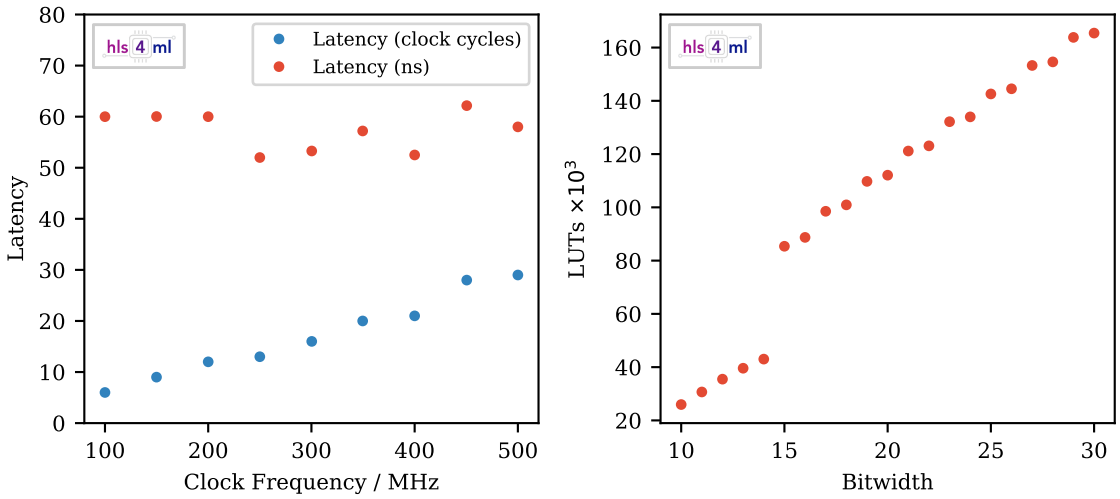

Figure 7. Left: latency — in clock cycles and nanoseconds — of the benchmark BDT model with 100 estimators of depth 4, as a function of the target clock frequency. Right: resource usage as a function of the bitwidth used for all features, thresholds and scores.

The variation of resource usage with the bitwidth is shown in the right plot of figure 7 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.

The benchmark model, as well as scans over the number of estimators and maximum depth, were evaluated using 18 bits. From figure 3 this can be seen to be comfortably sufficient to give numerically equivalent results to the CPU evaluation of the model. Using 14 bits, the ratio between Area Under the ROC Curve (AUC) achieved with the FPGA versus CPU inference is above 99.9%

2020 JINST 15 P05026

for all classes. For many applications, this performance will be adequate, and the resource saving seen in figure 7 from using 14 bits or below can be taken advantage of. In other cases, for example selecting rare signals with a large background, the extra precision from using more bits may be desirable to maintain reliable, stable performance not susceptible to numeric effects.

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 ns at a clock frequency of 200 MHz). We discussed the dependence of the FPGA resource usage and inference latency upon the model hyperparemeters, presenting a model which predicts the resource usage well. We compared 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]. Compared to a Neural Network applied to the same problem, a BDT is able to achieve very similar performance, with a comparable inference latency. The implementation of BDTs in the FPGA utilises LUTs most heavily, while the Neural Network predominantly uses DSPs. This functionality of the `hls4ml` library could support an efficient deployment of algorithms analogous to that described in ref. [8], which took data at the CMS experiment during the LHC Run II.

Acknowledgments

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. M.P., S.S., V.L. nd 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). 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. 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.* **A 543** (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.* **A 555** (2005) 370 [physics/0508045].

[3] A. Radovic et al., *Machine learning at the energy and intensity frontiers of particle physics*, *Nature* **560** (2018) 41.

[4] CMS collaboration, *Observation of a New Boson at a Mass of* 125 *GeV with the CMS Experiment at the LHC*, *Phys. Lett.* **B 716** (2012) 30 [arXiv:1207.7235].

2020 JINST 15 P05026

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

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

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

[8] CMS collaboration, *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 proceedings of the *2017 27th International Conference on Field Programmable Logic and Applications (FPL)*, Ghent, Belgium, 4–8 September 2017, pp. 1–8.

[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.

[11] M. Barbareschi, S. Del Prete, F. Gargiulo, A. Mazzeo and C. Sansone, *Decision Tree-Based Multiple Classifier Systems: An FPGA Perspective*, *Lect. Notes Comput. Sci.* **9132** (2015) 194.

[12] S. Buschjäger and K. Morik, *Decision tree and random forest implementations for fast filtering of sensor data*, *IEEE Trans. Circuits Syst. I Regul. Pap.* **65** (2018) 209.

[13] R. Kułaga and M. Gorgon, *FPGA implementation of decision trees and tree ensembles for character recognition in Vivado HLS*, *Image Process. Commun.* **19** (2014) 71.

[14] M. Pierini, J.M. Duarte, N. Tran and M. Freytsis, *HLS4ML LHC Jet dataset (150 particles)*, (2020), https://doi.org/10.5281/zenodo.3602260.

[15] F. Pedregosa et al., *Scikit-learn: Machine learning in Python*, *J. Mach. Learn. Res.* **12** (2011) 2825 [arXiv:1201.0490].

[16] National Instruments, *FPGA Fundamentals*, (2019) https://www.ni.com/it-it/innovations/white-papers/08/fpga-fundamentals.html.

[17] T. Chen and C. Guestrin, *XGBoost: A scalable tree boosting system*, in proceedings of the *22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '16)*, San Francisco, CA, U.S.A., 13–17 August 2016, pp. 785–794.

[18] A. Hocker et al., *TMVA — Toolkit for Multivariate Data Analysis*, physics/0703039.

2020 JINST 15 P05026