A Flexible HLS Hoeffding Tree Implementation for Runtime Learning on FPGA

Luís Miguel Sousa
Faculty of Engineering
University of Porto
Porto, Portugal
lm.sousa@fe.up.pt

Nuno Paulino, João Canas Ferreira, João Bispo
INESC-TEC and Faculty of Engineering
University of Porto
Porto, Portugal
nuno.m.paulino@inesctec.pt, jcf@fe.up.pt, jbispo@fe.up.pt

2022 IEEE 21st Mediterranean Electrotechnical Conference (MELECON) | 978-1-6654-4280-0/22/$31.00 ©2022 IEEE | DOI: 10.1109/MELECON53508.2022.9843092

***Abstract*—Decision trees are often preferred when implementing Machine Learning in embedded systems for their simplicity and scalability. Hoeffding Trees are a type of Decision Trees that take advantage of the Hoeffding Bound to allow them to learn patterns in data without having to continuously store the data samples for future reprocessing. This makes them especially suitable for deployment on embedded devices. In this work we highlight the features of a HLS implementation of the Hoeffding Tree. The implementation parameters include the feature size of the samples (D), the number of output classes (K), and the maximum number of nodes to which the tree is allowed to grow (Nd). We target a Xilinx MPSoC ZCU102, and evaluate: the design's resource requirements and clock frequency for different numbers of classes and feature size, the execution time on several synthetic datasets of varying sizes (N) and the execution time and accuracy for two datasets from UCI. For a problem size of D=3, K=5, and N=40000, a single decision tree operating at 103MHz is capable of 8.3× faster inference than the 1.2 GHz ARM Cortex-A53 core. Compared to a reference implementation of the Hoeffding tree, we achieve comparable classification accuracy for the UCI datasets.**

***Index Terms*—Decision Tree, Hoeffding Tree, Machine Learning, Incremental Learning, FPGA, Hardware, High-Level Synthesis**

I. Introduction

With the rise of edge computing, FPGA vendors have been releasing and marketing CPU+FPGA SOCs as the ideal solution for this domain. As edge devices are often specialised for a single task in a constrained environment, it is advantageous to build dedicated hardware to improve performance and energy efficiency. FPGAs offer the advantage of targeted hardware without losing the ability to adapt the platform to changes (e.g., security updates), while being more efficient than a pure software solution.

As High Level Synthesis (HLS) matures [1], it becomes a more attractive approach to creating efficient high-preformance accelerators for FPGA devices.

Machine Learning (ML) algorithms are a prime candidate for acceleration at the edge, but their computational requirements exceed the capabilities of many embedded devices. Inference at the edge is a problem being addressed by many works, but training at the edge still faces hurdles to adoption despite its clear benefits. In the field of Decision Treess (DTs), many algorithms are incompatible with devices of this class due to memory constraints. ID3 [2], and derivatives such as C4.5 and C5.0 require the entire training dataset be present in memory for training. Incremental learning algorithms such as ID5 [3], ID5R [4] and ITI [5] do allow for ongoing learning from streaming data but store the dataset samples within the tree.

Hoeffding Trees [6] are incremental learning trees, which are more suitable for embedded scenarios because they have the following advantages: they asymptotically guarantee the same classification as traditional batch learners, and they store information about the distribution of samples statistically rather than the samples themselves, which drastically reduces memory requirements, especially for large datasets.

In this work, we present a flexible C/C++ HLS implementation of a Hoeffding Tree, designed for compatibility with HLS flows for FPGAs. The Hoeffding tree variant we build upon was proposed by Lin et al. [7] which is in turn based on an earlier variant where the storage of the statistical data of the sampling distribution of the original Hoeffding Tree was replaced by a Gaussian approximation [8]. Lin et al. replaced this approximation with quantile estimation using asymmetric signum functions [9]. The result is a larger memory footprint but a reduction in computational requirements, while achieving similar results. Since it is designed in Verilog, the applicability of the implementation is limited to circuit synthesis, e.g., for FPGA. By using HLS, an implementation can be created that is equally suitable for both CPU and FPGA.

The contributions of this work are as follows:

- a generic, template-based C/C++ implementation of the Hoeffding Tree classifier as per Lin et al; [7], but that is suited for HLS.
- functional validation of the implementation through software execution, and post-synthesis on a Xilinx ZCU102 development board;
- evaluation of memory requirements of the tree object for different template parameters;
- evaluation of FPGA resource requirements for different template parameters;
- evaluation of the design's training and inference time, for 4 synthetic data sets, and 2 datasets from UBI, versus an ARM Cortex-A53 core;

- accuracy comparison with the reference implementation for the 2 UBI datasets.

II. HLS Hoeffding Tree Implementation

A decision tree is a type of machine learning algorithm used either for classification or regression. A decision tree performs sequential binary decisions over an incoming vector of features, and a classification is computed when a leaf node is reached. During training, leaf nodes are added to the tree, based on a splitting criteria, that separate the data into two regions at every tree junction. A Hoeffding tree is a type of decision tree where the criteria is the Hoeffding bound, shown in Equation 1. The tree performs learning and inference by relying on a property of the Hoeffding bound that guarantees that best splitting point is chosen. If a gain function G, is to be maximised, then given $G(X)$ and $G(Y)$ (X and Y being the attributes that generate the highest and second highest values of G) if $G(X) - G(Y) > \varepsilon$ then the Hoeffding bound guarantees with probability $1 - \delta$ that X is the best attribute to split on. R represents the range of the attributes and N the number of samples on a node.

$$\varepsilon = \sqrt{\frac{R^2 ln(1/\delta)}{2N}} \tag{1}$$

Over other criteria, the Hoeffding bound has two characteristics: it allows for online incremental learning and growth of the tree which asymptotically tends towards the results provided by batch learners, and is independent of the probability distribution of the data sampling. The Hoeffding tree allows for continuous learning and node splitting for a potentially infinite number of samples (e.g. streaming applications) [6].

FPGAs have been intensively studied for decision tree implementations, as a tree structure maps efficiently to specialised hardware. In conjunction with other optimisations, decision trees in FPGAs have been shown to outperform CPU and GPU solutions [10]. Lin et al. [7] demonstrate speedups of up to 1500x for an RTL implementation of the Hoeffding tree versus a 2.6GHz processor. Our aim is to explore a higher abstraction level via HLS, providing greater applicability features, while evaluating the attainable performance.

We implemented the tree as a collection of C++ templated classes. The template parameters include the maximum number of nodes in the tree, the feature size, and the floating-point precision. These classes implement the training and inference methods that are then synthesised to hardware. At runtime, the C++ tree object can be manipulated in software, and passed as an argument to the training and inference methods, as summarised in Figure 1.

This allows for the instantiation of several tree objects in memory (with different template parameters, if desired). Trees with the same template parameters can be processed by the same synthesised circuit. Since the functions can also be invoked in software, this means that training or inference can be dynamically partitioned based on which device performs better for either task, as a function of the tree parameters.

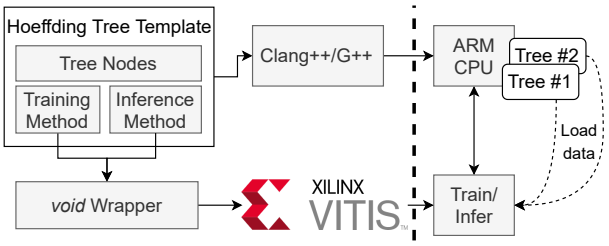

Fig. 1. Software and hardware architecture of the Hoeffding Tree implementation; the training and inference kernels are shared by multiple tree objects

This also means that if the FPGA is occupied processing a tree object, other trees can be evaluated via software without the need for a blocking wait.

Finally, evaluation of multiple trees is possible by either a combination of software and hardware invocations, by deploying multiple instances of the hardware kernel, or by time-multiplexing a single hardware kernel (as explained below). Either case allows for the possibility of arbitrary runtime tree ensembles. This evaluation is currently future work.

The Xilinx Vitis HLS flow enforces an OpenCL model for kernel invocation. The implemented kernel, `krnl_Tree`, receives 4 arguments: a `HoeffdingTree` object as mentioned, an array of samples, an array of output classifications, and the size of these arrays.

In this model, a large overhead penalty would occur for invocations with a single sample, due to the data transfer time. A practical application of the kernel design could be, e.g., in the sensor domain, where the tree could continuously sample fused data from multiple sensors (i.e., multiple attributes) without processor intervention, avoiding transfer overheads. Alternatively, streaming samples can be accumulated until a sufficiently large number is held such that it mitigates the data transfer overhead. We emphasise that this does not mean that the tree behaves as a batch learner, as one sample is processed per each *infer-then-train* step.

Inference on an incremental learning decision tree cannot be easily parallelised as the model changes and evolves with every training sample that arrives. This restricts the pipeline to dealing with one sample at a time, sequentially. The sample structure contains information about whether it should be used for training purposes or only for inference. Thus, as the kernel loops through the sample array, it executes either the `train` or `infer` method of the tree object accordingly. The results are placed in the output data structure.

The OpenCL API allows for fine-grained control of how these arguments are passed to the kernels, each argument being a separate buffer with persistent storage. Thus, trees can be transferred to FPGA memory once, and not retrieved between executions of the kernels. With this mechanism, a tree object can reside in memory while only new samples are transferred in, and the updated model can be retrieved in a final stage.

Conversely, the samples themselves may remain in memory, and trees freely exchanged. This is one strategy for the construction of tree ensembles mentioned previously. Trees

TABLE I
N, D, K AND ND EFFECTS ON FPGA RESOURCE UTILISATION

Nodes	100	100	100	1000	100	100	100	1000
K	5	5	10	5	5	5	10	5
D	3	100	3	3	3	100	3	3
N	40k	40k	40k	40k	500k	500k	500k	500k
LUT	23304 (8.6%)	20567 (7.6%)	23776 (8.8%)	24351 (9.0%)	23304 (8.6%)	20567 (7.6%)	23776 (8.8%)	24351 (9.0%)
LUTRAM	1395 (1.0%)	1179 (0.8%)	1399 (1.0%)	1397 (1.0%)	1395 (1.0%)	1179 (0.8%)	1399 (1.0%)	1397 (1.0%)
FF	35682 (6.6%)	29775 (5.5%)	36374 (6.7%)	36336 (6.7%)	35682 (6.6%)	29775 (5.5%)	36374 (6.7%)	36336 (6.7%)
BRAM	12 (1.3%)	9.5 (1.0%)	12 (1.3%)	12 (1.3%)	12 (1.3%)	9.5 (1.0%)	12 (1.3%)	12 (1.3%)
DSP	23 (0.9%)	25 (1.0%)	25 (1.0%)	25 (1.0%)	23 (0.9%)	25 (1.0%)	25 (1.0%)	25 (1.0%)
BUFG	13 (3.2%)	13 (3.2%)	13 (3.2%)	13 (3.2%)	13 (3.2%)	13 (3.2%)	13 (3.2%)	13 (3.2%)
MMCM	1 (25.0%)	1 (25.0%)	1 (25.0%)	1 (25.0%)	1 (25.0%)	1 (25.0%)	1 (25.0%)	1 (25.0%)
Freq. (MHz)	103.6	103.6	103.6	103.6	103.6	103.6	103.6	103.6

can reuse the same kernel instance via time-multiplexing, or by concurrent instantiation of several copies of `krnl_Tree`. In either case, the same read-only sample buffer can be assigned to all trees, thus significantly reducing overhead and preventing data duplication. For brevity, the evaluation of ensembles is out of the scope of this paper.

III. HOEFFDING TREE SOFTWARE STRUCTURE

In this section we describe the code structure of our modular approach to the tree construction. It allows us to adapt the current code of the Hoeffding Tree to other ways of statistical sample storage or to create different Decision Tree types altogether. The following C++ templated classes are used:

A. NodeData

This class is responsible for storing all the data and methods regarding training in a node. In this case, it harbours all methods for quantile estimation, how to calculate the Gini impurity of the node and find the optimal split candidates. Template parameters of this class allow for compile-time customisation of the type used for storing the quantile values and make non-integer calculations. Defaults to `float` type. Changing this type affects the precision of all the tree's calculations. Other customisations include the number of tree attributes (D), output classes (K) and the types used to handle and store indexes for these properties.

B. Node

The `Node` class stores all information regarding a node in the tree: whether it is split, what are its children, the split value, split attribute and the corresponding Data object. The template parameters for this class allow for changes to the type used to store node indexes and to the Data object class (defaults to `NodeData`). If one decides that the Gaussian approximation method is preferable in their case, a re-implementation of `NodeData` is all that is necessary.

C. BinaryTree and HoeffdingTree

`BinaryTree` is a base class for binary tree operations. It stores an array of node objects (whose size and class is defined in the class template) and contains methods for managing those nodes (splitting a node and defining children) and sort a sample through the tree. The `HoeffdingTree` class extends `BinaryTree` to include methods on how to calculate the Hoeffding bound and the higher-level training algorithm agnostic to how the data is stored.

D. TypeChooserMath

This namespace encapsulates a set of macros to resolve data types. Based on the required precision, the class resolves the datatype to either standard native types or to Xilinx Arbitrary Precision (AP) types (up to 64 bits). Supporting AP types introduces compatibility issues with math functions in the `std` namespace, as Xilinx provides its own `hls` namespace for math functions for its AP types. The `tcm` (TypeChooserMath) namespace provides wrappers to resolve these conflicts based on chosen datatype.

IV. EXPERIMENTAL EVALUATION

We performed the following experiments: evaluated the resource utilisation of a single synthesised tree for a range of values for the feature size and number of classes; evaluated the training and inference time of a single tree in hardware, versus the ARM CPU, for several synthetic clustering datasets (varying number of samples, clusters, and feature size); evaluated the classification accuracy and execution time of a single tree for UCI's Bank and Covertype datasets.

A. Resource Utilisation

Table I presents various configurations of the kernel, tailored for datasets of different dimensions (D), with different number of classes (K), number of samples (N) and maximum number of nodes (Nd). The purpose is to determine the effect of these parameters on FPGA resource utilisation. The resource usage of the tree does not scale significantly as a function of problem size. Specifically, N has no particular effect since

the tree processes each sample sequentially. A design capable of processing samples in parallel could be devised where the resource scaling with N would be more noticeable as a function of the degree of parallelism. However, this would imply the creation of two different circuit implementations (i.e., kernel functions), since learning must be sequential, while later inference could benefit from parallel processing of incoming data. We leave this as future work, and out of the scope of this paper. The other problem size parameters result in a small increase of resource usage. The maximum number of nodes (Nd) only influences the maximum depth of the tree traversal while the feature size (D) and the number of classes (K) affect, mainly, some of the innermost loops of the tree circuit.

The highly sequential nature of the generated kernel, also explains why the performance on training tasks is poor when compared to the CPU. This overall advantage is less surprising when considered in the context of an 11-fold CPU advantage in terms of clock speed. Current HLS tools cannot automatically parallelize sequential code. Without hardware design expertise to optimise the design, the implementation will be far from optimal. In our implementation, we still believe that further parallelization can be achieved even within a single tree, through inner loop unrolling or memory partitioning. One interesting result is that of the kernel's operating frequency. It remains unchanged for all configurations. Looking deeper into the cause of this phenomenon, we found that the bottleneck is the sorting of a sample down from the root node to the appropriate leaf node. This sequential operation also prevents the kernel from being properly pipelined.

Figure 2 illustrates the C++ object size growth given different Nd, D and K parameter values. The number of samples processed to date by the tree does not influence the size of the model due to the Hoeffding Tree's statistical storage of sample data.

B. Performance

These results were obtained by feeding the tree with datasets of K clusters in a D dimensional spaces, constituted of N points. For these experimental runs, we will have the entire dataset transferred in a single operation to the FPGA's memory.

Looking at the first four rows of Table II (D=3) it can be observed that for a 3-dimensional dataset, regardless of the bundle size, the ARM CPU in the ZCU102 SoC significantly outperforms the FPGA implementation in both the training and inference tasks. Also, the performance gap between both implementations grows with the number of samples processed. This indicates that the kernel is slower, per iteration, than the pure software solution. Regarding the last four rows of Table II (D=100), the ARM CPU still outperforms the FPGA kernel in training. However, it does it with a lower margin and one that does not appear to grow with the added number of samples. On the inference task with this larger dataset, the FPGA outperforms the ARM processor by 8.3×.

Table III presents benchmarks of two of the UCI datasets used by Lin et al. [7]. The same tree parameters were used

TABLE II
TRAINING AND INFERENCE TIMES FOR FOUR SYNTHETIC CLUSTERING DATASETS, FOR THE ARM CPU (1.2GHZ) AND THE FPGA (103MHZ)

K	D	N	Task	ARM CPU	FPGA	Speedup
5	3	40k	Training	207 ms	1,990 ms	0.10×
			Inference	151 ms	462 ms	0.33×
		500k	Training	2,983 ms	30,933 ms	0.10×
			Inference	2,260 ms	11,442 ms	0.20×
	100	40k	Training	6,028 ms	51,648 ms	0.12×
			Inference	3,924 ms	469 ms	8.37×
		500k	Training	75,763 ms	651,775 ms	0.12×
			Inference	49,495 ms	11,494 ms	4.31×

TABLE III
EXECUTION TIME (INFER-THEN-TRAIN) AND ACCURACY (ACC.) FOR COVERTYPE AND BANK DATASETS, FOR THE ARM CPU (1.2GHZ) AND THE FPGA (103MHZ)

	ARM CPU		FPGA		
	Acc.	Time	Acc.	Time	Speedup
Bank	88.3%	202 ms	88.3%	8,525 ms	0.02×
Covertype	72.2%	9,712 ms	63.7%	374,600 ms	0.03×

($\delta = 0.001$, $\lambda = 0.01$, $\tau = 0.05$, $n_{min} = 200$, $n_{pt} = 10$, $n_{quantiles} = 16$, $Nd = 2047$), with one being of special relevance: Nd (maximum number of nodes). A significant slowdown occurred. With the increased number of nodes, the sequential tree traversal algorithm increases in length. While the previous evaluation separated the train and inference components, so that we could evaluate the impact of each on the performance, this evaluation processes the training data as intended by the runtime learning algorithm, i.e., a sequence of infer-and-train steps per each consumed data point.

Our HLS implementation achieves comparable accuracy for *Bank*, although the performance for *Covertype* is inferior. Lin et al. [7] reports 89.30% and 72.51%, respectively. We believe a difference in calculation precision between the CPU and FPGA caused the degradation, despite the use of 32-bit floating point data types for both devices.

Figure 3 illustrates a tree model obtained from training with the *Covertype* dataset that was only allowed to grow to a maximum of 5 nodes (Nd=5).

V. RELATED WORK

Kulaga et al. [11] present an HLS decision tree ensemble solution for inference tasks. The results achieved are competitive regarding performance when compared to the ARM core present in the tested SoC. However, the design is highly dependent on the number of trees and corresponding depths, as a change in ensemble parameters requires re-tuning multiple pragmas.As we have also seen, an unavoidable sequential portion of the algorithm is the sample sorting through the tree structure. Unlike our approach, the number of trees in an

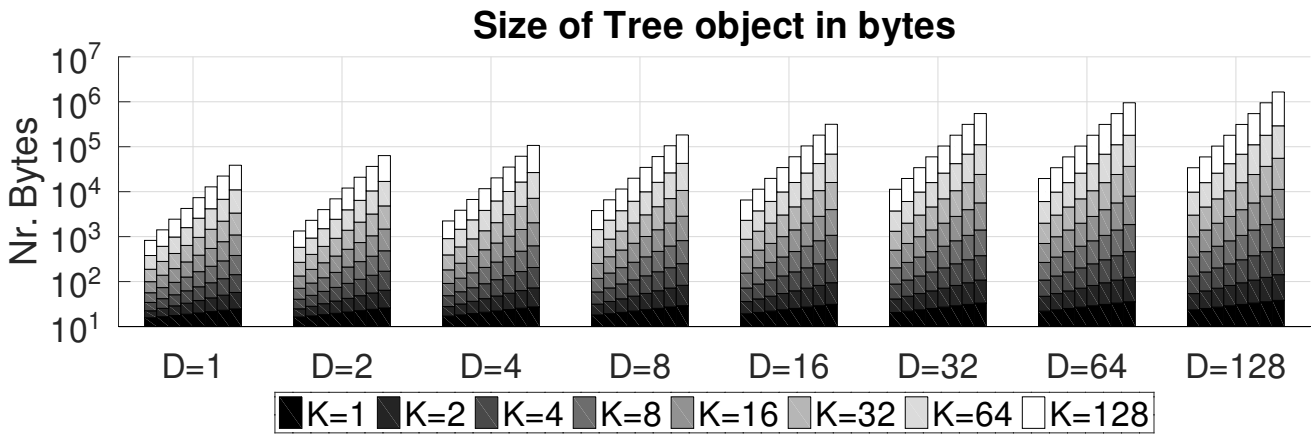

Fig. 2. Size of Tree objects in bytes for Nd, D and K. Each bar in every grouping, depicts a tree with a maximum number of nodes (Nd) from 2^0 to 2^7.

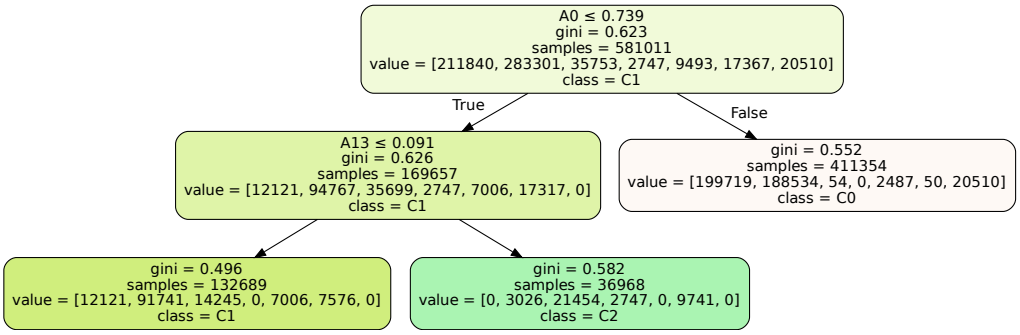

Fig. 3. Illustrative visualisation of tree model derived from UCI Covertype dataset. The tree was only allowed to grow to 5 nodes (Nd=5) for the purposes of this illustration.

ensemble is hardcoded into the synthesised kernel. In contrast, by having one or more synthesised training/inference methods (for different hyper-parameters), we can deploy *N* instances of such circuits and process a runtime allocated number of trees.

As previously stated, the work on this paper builds on Lin et al. [7] work. However, their implementation is closed-source and done in Verilog, which excludes native execution on CPUs. Also, as the work was developed for a datacenter-class FPGA device, the implementation is very resource intensive and thus not suitable for small devices such as the ones used on embedded systems.

VI. Conclusions

We presented a flexible and scalable implementation of a Hoeffding Tree compatible with HLS tools[1] We performed a functional validation of the tree design, against software execution, by implementation on chip on a Xilinx ZCU102. We provide a evaluation of the design's resource usage for multiple template parameter values (i.e., maximum tree size, number of sample attributes, number of clusters, and number of dataset samples), as well as execution time versus an ARM Cortex-A53 processor. The resource requirements of the tree do not scale significantly with problem size, although further HLS optimisations such as unrolling remain unexplored. Even so, we outperform the ARM by 8.3x times for largest dataset for the inference task, while being 8.6x slower during training. As future work, we envision the use of tree ensembles, and the partitioning of training and inference task between software and hardware based on problem size.

[1]https://github.com/lm-sousa/Hoeffding-Tree

Acknowledgements

This work was supported by the PEPCC project (PTDC/EEI-HAC/30848/2017), financed by Fundação para a Ciência e Tecnologia (FCT).

References

[1] Xilinx Inc., "Vivado High-Level Synthesis," Online, Tech. Rep., 2020. [Online]. Available: https://www.xilinx.com/products/design-tools/vivado/integration/esl-design.html

[2] J. R. Quinlan, "Learning Efficient Classification Procedures and Their Application to Chess End Games," *Machine Learning*, pp. 463–482, 1983. [Online]. Available: https://link.springer.com/chapter/10.1007/978-3-662-12405-5%5F15

[3] P. E. Utgoff, "ID5: An Incremental ID3," in *Machine Learning Proceedings 1988*. Elsevier, 1 1988, pp. 107–120. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/B9780934613644500177

[4] ——, "Improved Training Via Incremental Learning," in *Proceedings of the Sixth International Workshop on Machine Learning*. Elsevier, 1 1989, pp. 362–365. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/B9781558600362500928

[5] P. E. Utgoff, N. C. Berkman, and J. A. Clouse, "Decision Tree Induction Based on Efficient Tree Restructuring," *Machine Learning 1997 29:1*, vol. 29, no. 1, pp. 5–44, 1997. [Online]. Available: https://link.springer.com/article/10.1023/A:1007413323501

[6] P. Domingos and G. Hulten, "Mining high-speed data streams," in *Proceeding of the Sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. New York, New York, USA: Association for Computing Machinery (ACM), 2000, pp. 71–80. [Online]. Available: http://portal.acm.org/citation.cfm?doid=347090.347107

[7] Z. Lin, S. Sinha, and W. Zhang, "Towards Efficient and Scalable Acceleration of Online Decision Tree Learning on FPGA," in *2019 IEEE 27th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*. IEEE, 4 2019, pp. 172–180. [Online]. Available: https://ieeexplore.ieee.org/document/8735508

[8] B. Pfahringer, G. Holmes, and R. Kirkby, "Handling Numeric Attributes in Hoeffding Trees," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 5012 LNAI, pp. 296–307, 2008. [Online]. Available: https://link.springer.com/chapter/10.1007/978-3-540-68125-0%5F27

[9] A. Althoff and R. Kastner, "An Architecture for Learning Stream Distributions with Application to RNG Testing," *Proceedings - Design Automation Conference*, vol. Part 128280, 6 2017. [Online]. Available: http://dx.doi.org/10.1145/3061639.3062199

[10] M. Barbareschi, S. Barone, and N. Mazzocca, "Advancing synthesis of decision tree-based multiple classifier systems: an approximate computing case study," *Knowledge and Information Systems 2021 63:6*, vol. 63, no. 6, pp. 1577–1596, 4 2021. [Online]. Available: https://link.springer.com/article/10.1007/s10115-021-01565-5

[11] R. Kułaga and M. Gorgoń, "FPGA Implementation of Decision Trees and Tree Ensembles for Character Recognition in Vivado Hls," *Image Processing & Communications*, vol. 19, no. 2-3, pp. 71–82, 9 2014. [Online]. Available: https://www.sciendo.com/article/10.1515/ipc-2015-0012