In-memory Machine Learning using Adaptive Multivariate Decision Trees and Memristors

Akash Chavan, Pranav Sinha, Sunny Raj
Dept. of Computer Science and Engineering, Oakland University, Rochester, MI, USA
{akashchavan,pranavsinha,raj}@oakland.edu

***Abstract*—We introduce a framework to design in-memory decision tree machine-learning (ML) circuits using memristor crossbars. Decision trees (DTs) offer many advantages over neural networks, such as enhanced energy efficiency, interpretability, safety, privacy, and speed, along with reduced dependence on extensive training data. We propose an adaptive multivariate decision tree (AMDT) training algorithm, which constructs decision trees that incorporate both univariate and multivariate features, facilitating the creation of higher accuracy and energy-efficient crossbar designs compared to the state-of-the-art (SOTA). Our circuits are realized using pure memristor crossbars, requiring just one memristor per cell and no transistors while employing sneak-paths for flow-based in-memory computations. In comparison to the SOTA, our approach produces designs that are, on average, 4% more accurate and require 12.6% lower energy.**

***Index Terms*—In-memory computation, memristors, flow-based computing, decision trees, multivariate decision trees, energy-efficient hardware.**

I. Introduction

In-memory computing using memristor crossbars provides solutions for multiple problems with existing computing devices. The von Neumann computing architecture utilized in almost all mainstream computing devices suffers from a bottleneck between the memory and the compute units, where the bandwidth between the two units, instead of the CPU speed, determines the throughput of data-intensive tasks such as machine learning [1]. Moor's Law and Dennard Scaling, which enabled impressive year-over-year improvements in CMOS devices, has ended, leading to an uncertain future, necessitating the search for novel computing devices. In-memory computing using memristors is an attractive solution for both the memory bottleneck and the device problem.

However, most popular in-memory memristor devices utilize traditional CMOS circuit components, including transistors, alongside memristors to accelerate computations [2]. Using traditional CMOS circuit elements alongside memristors dilutes some critical advantages memristor devices enjoy. Specifically, adding a transistor to a crossbar cell increases its size and makes it less energy efficient, and the presence of traditional CMOS elements makes the circuits vulnerable to radiation degradation [3]. This hybrid approach is necessary to control the so-called 'sneak paths' that affect the final computed value [4].

Another approach termed 'flow-based computing' has been proposed that utilizes these pathogenic sneak-paths for computations and provides SOTA energy and space efficiency. Flow-based memristor crossbar designs have been shown to provide SOTA energy efficiency for circuits in multiple standard circuits benchmarks, including RevLib benchmark and for multiple machine learning tasks [5]–[7]. Flow-based circuits for machine learning have been created using decision trees as they are similar to the BDD (Binary Decision Diagram) based approach for designing general-purpose circuits and provide a straightforward conversion from ML model to in-memory flow-based designs.

DTs have many desirable properties that make them suitable for a variety of tasks. They have lower complexity compared to neural networks and thus require less energy during inference, making them ideal for use in energy-constrained environments such as edge computing and mobile devices [8], [9]. DTs are less vulnerable to adversarial attacks than neural networks and can be employed in scenarios prioritizing safety, robustness, privacy, and security [10], [11]. Furthermore, neural network outputs are not easy to interpret, while the output generated by DTs can be explained easily [12]. DTs also find usage in situations where the speed and lower energy cost have been used to select one of multiple powerful machine learning algorithms to run on the input data [8].

However, existing work on creating flow-based circuits for ML has utilized univariate decision trees and does not provide an algorithm to create designs of higher accuracy multivariate decision trees. A tradeoff for this higher accuracy is in circuit size and energy utilization, where multivariate nodes in the decision trees require complex designs, leading to larger circuits and, consequently, higher energy utilization. In this work, we propose a balancing of the two objectives, namely creating higher accuracy flow-based designs and, at the same time, ensuring lower energy utilization. To this end, we propose an AMDT creation algorithm that intelligently utilizes both multivariate and univariate decision nodes in the creation of the trained decision tree. Our AMDT algorithm is a generalization of univariate and multivariate decision trees and can potentially produce pure univariate or multivariate decision trees and thus choose the designs that improve both the accuracy and the energy efficiency. We make the following contributions in the paper:

- We propose a generalized adaptive decision tree generation algorithm that intelligently selects between univariate and multivariate decision tree nodes, improving the accuracy and efficiency of synthesized crossbar designs.
- We experimentally verify by testing the algorithm on 11 ML datasets that our AMDT circuits on average provide

more than 4% higher accuracy and utilize 12.6% less energy compared to the SOTA.

II. Related Work

Numerous decision tree hardware designs in the literature focus on high throughput and energy efficiency. They achieve throughput in the range of 10^8 inferences per second with energy utilization in nJs per inference [7], [13]. However, most designs use traditional CMOS circuit elements, including 1T1R cells, 6T2R cells, or conventional CMOS peripheral circuitry for calculations [14], [15]. Furthermore, these lines of work do not provide a method to find designs for multivariate decision trees. In this paper, we present an algorithm for generating adaptive decision trees that contain both univariate and multivariate nodes and then create equivalent flow-based designs for energy-efficient in-memory computations.

A. Decision Tree

Traditionally, DTs are constructed using univariate nodes, where each internal node tests the value of a single feature to partition the data into two distinct subsets. Due to the simplicity of their decision nodes, existing work for energy-efficient ML hardware has focused on univariate models [14], [15]. For instance, Sinha and Raj [7] use the CART algorithm to produce univariate decision trees. Subsequently, the tree is converted into a Binary Classification Graph (BCG), which is then further processed to create the flow-based crossbar design. Due to univariate nodes in the decision tree, the resultant crossbar design leaves some accuracy gains on the table.

Multivariate decision trees are designed to capture complex interactions between multiple features, thereby potentially improving predictive performance over simple univariate DTs [16] and require complex circuitry for implementation. The flexibility of multivariate decision trees is a significant advantage, as they are not confined to the restriction that each splitting hyperplane must be orthogonal to an attribute's axis, unlike univariate decision trees, and can thus provide higher accuracy. The trade-off of the accuracy gain is in circuit complexity for realizing the decision node in hardware. Our approach improves both accuracy and energy efficiency and includes both univariate and multivariate nodes for making decisions. Several multivariate decision tree training methods have been proposed in the literature. In our implementation, we adopt a similar approach to [17] but use logistic regression [18] for the splitting process, as elaborated in Section III-C.

B. Flow-based computing

Our designs utilize the flow-based computing paradigm for performing in-memory computation using memristor crossbars [7]. In this paradigm, row and column wires of the crossbar are abstracted as nodes, and the memristors act as connections between the nodes. The synthesized crossbar design consists of labels assigned to the memristors in the crossbar. Each label corresponds to input bits and determines the configuration of the memristor during run-time. A label can be configured with the same value as the input, or it can be configured with the negation ($\neg$) of the input value. The input is loaded on the crossbars during run time according to the memristor labels. An input value of 1 is configured as a low resistance value, and 0 is configured with a high resistance value. Then, a current is applied to the bottommost row. The configuration of the memristors determines the current flow; a memristor configured with low resistance will connect the row and the column wires and allow current to flow through it; a memristor configured with high resistance will prevent this flow from happening. The presence of current in the top M rows, where M is the number of classes, determines the output of the classification. The memristors in the crossbar have to be configured for each input, and since the configuration energy required for the memristors is in the Femto Joules scale, it makes the computation highly energy efficient [19]–[21].

III. Method

The goal of the paper is to create flow-based circuits for ML using multivariate decision trees that are more accurate than simple univariate decision trees while at the same time being more energy efficient. In a typical multivariate decision tree, all of the features are used in all of the nodes to make a decision. This constant usage of all the features is unnecessary and can lead to inefficient circuit designs. In our proposed adaptive multivariate decision tree, we choose a fixed number of features that provide the best decision at a node. This choice of K features reduces the number of features that need to be mapped into the circuit by removing unnecessary and uninformative features. To further balance the issue of energy efficiency and accuracy, we take an adaptive approach: at each decision node, we consider the tradeoff determined by the hyperparameter λ of using univariate vs. K-variate nodes and make an informed choice to select either of the two.

Algorithm 1 takes as input a training sample matrix $\mathbf{X}$, containing the feature vectors, the corresponding class label matrix $\mathbf{y}$, the number of features to use for multivariate decisions K, and λ which controls the number of K-variate nodes in the decision tree. The algorithm outputs decision tree T, where each decision node is either univariate or K-variate, and the leaf node with the assigned class label. The algorithm maintains a list of nodes that are being worked on and needs further processing using the $queue$ data structure. It is initialized with the root node that contains the complete dataset. In each iteration of the while loop, the best split obtainable using a single feature is calculated; this step utilizes the Information Gain criteria used by the popular ID3 algorithm [22]. Next, the best split obtainable using K features is calculated (details in Section III-A), followed by selecting the best K features, and finally, training a logistic regression classifier to obtain a split of the data that best matches the separated classes. Once the best univariate and K-variate decisions are obtained, then depending on the information gain and the λ hyperparameter, either of the two is selected for inclusion in the AMDT. Finally, the data splits produced by the selected decision are inserted into the $queue$ for further processing if they do not meet the stopping criteria. If the selected splits meet

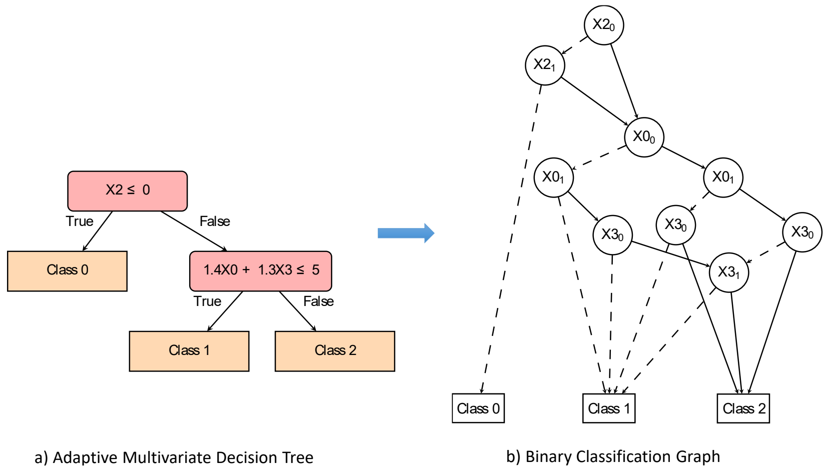

Fig. 1. (a) AMDT generated from our method for the Iris flower dataset, and (b) the corresponding binary classification graph (BCG) [7].

the stopping criteria, then they are assigned a label and not processed further. We utilize the majority class proportion-based stopping criteria proposed in [23] where after each split, the purity of the two sample subsets is computed. If the purity of a sample subset is less than a defined threshold δ, then the child node corresponding to the sample subset turns into a new parent node, with the sample subset requiring partition again; otherwise, the child node turns into a leaf node, and the sample subset will not be partitioned.

Once the AMDT is trained, we use the two mapping algorithms presented in [7] to create the intermediate BCG and then map it onto the crossbar to obtain the final crossbar design. The AMDT and BCG for the Iris dataset are shown in Figure 1. The crossbar design and the run time snapshot of the execution are also shown in Figure 2. A more detailed discussion of the steps involved in creating the AMDT is discussed in the following subsections.

Algorithm 1 Adaptive multivariate decision tree training

```
Require: Dataset $\mathbf{X}, \mathbf{y}$; hyperparameters $K$ and $\lambda$
Ensure: Decision tree $T$
1: Initialize tree $T$ with root node $N$
2: $queue.push(N)$
3: while $queue$ not empty do
4:     $N \leftarrow queue.pop()$
5:     Find best univariate split $S_u$ for $N$
6:     Find $K$-variate split $S_b$
7:     if InformationGain$(S_u)$ $>$ $\lambda \cdot$ InformationGain$(S_b)$ then
8:         $N.split \leftarrow S_u$ {Select univariate split}
9:     else
10:        $N.split \leftarrow S_b$ {Select $K$-variate split}
11:    end if
12:    Create child nodes $C_1, C_2$ partitioned by $N.split$
13:    if $C_1$ does not satisfy stopping criteria then
14:        $queue.push(C_1)$
15:    end if
16:    if $C_2$ does not satisfy stopping criteria then
17:        $queue.push(C_2)$
18:    end if
19: end while
20: return Decision tree $T$
```

A. Class Separation

The nonlinear multivariate decision tree with multilayer perceptrons at the internal nodes was proposed by Guo et al. [24]. They also proposed a heuristic to group $M > 2$ classes into two, which is necessary as the nodes in the tree are binary. Thus they use a nested optimization problem where in the inner optimization, gradient-descent is used to find the weights that minimize the mean-square error as usual in training neural networks and so find a good split for the given two distinct groups of classes. In the outer optimization problem, the exchange heuristic is used to find the best split of M classes into two groups through a local search with backtracking, with time complexity $O(M^2)$. Loh and Shih [25] use an unsupervised 2-means clustering algorithm to do a preliminary grouping of the classes into two superclasses. We follow a similar strategy to [24], [25] but use a simple and deterministic method for class grouping. At each decision node, using Euclidean distances we create two distinct subsets of similar size, the first one includes the samples from the most frequent class along with the samples from the classes exhibiting close proximity to it, and the second set consists of samples from the remaining classes. The algorithm dynamically determines a positive class by identifying the class with the highest occurrence at the decision node. Subsequently, it calculates the mean of the positive class and measures the distances to the means of other classes. Through a sorting and iterative grouping process, the algorithm effectively divides the classes into a positive class group and a negative class group.

B. Feature Selection

The core concept is that specific dimensions within the instance subspace when reaching a particular decision node,

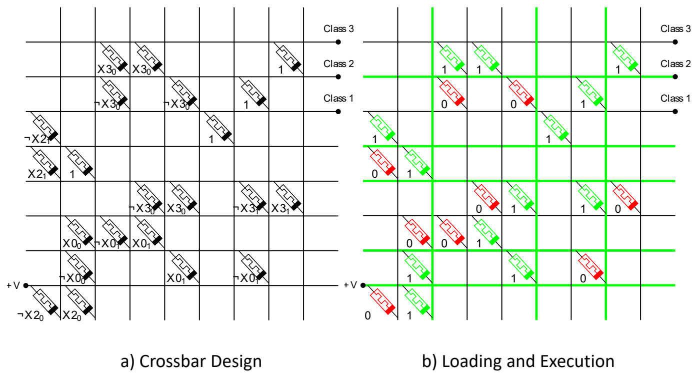

Fig. 2. (a) Crossbar design for classifying the Iris dataset with labeled memristors, obtained from the BCG from Figure 1 and (b) loading and execution of the crossbar. Wires with a current in them have been shown in green. A green class 2 wire indicates the classification output as 2.

TABLE I
COMPARISON OF OUR METHOD TO THE SOTA.

Method	Process (nm)	Depth	Energy (nJ)	Area (mm^2)	Accuracy
Intel X5560 [29]	45	6	2.04×10^7	-	-
Nvidia Tesla M2050 [29]	40	6	1.10×10^7	-	-
Xilinx Virtex-6 [29]	40	6	3.51×10^5	-	-
ASIC [30]	65	-	1.87×10^5	6.50	-
ASIC [31]	65	-	4.60×10^5	2.30	-
ASIC IMC [13]	65	6	19.4	0.56	-
ACAM [14]	65	10	1.28	1.80	-
DT [7] (worst-case)	70	10	2.05×10^{-2}	5.23×10^{-3}	-
DT (MNIST) [7]	70	10	1.74×10^{-2}	9.40×10^{-4}	0.82
AMDT (MNIST)	70	10	1.18×10^{-1}	5.5×10^{-2}	**0.88**

may remain constant and therefore be considered redundant. Avoiding these features may increase the generalization ability and reduce node complexity [26]. We perform the chi-squared statistical test [27] similar to Kim et al.'s CRUISE [28] to evaluate the relevance and significance of each feature concerning the target variable at each decision node and select K best features based on their scores. The indices of these selected features are extracted for subsequent use in the logistic regression model training.

C. Data Partitioning

A logistic regression model is trained on the reduced sample matrix which contains only selected features K. The use of logistic regression at each node within the algorithm for data partitioning provides a transparent and interpretable decision boundary formulation, as the estimated coefficients and thresholds can be directly interpreted in terms of the influence of each feature on the classification outcome [32], and can be used in a straightforward manner to create flow-based memristor crossbar circuit designs.

IV. RESULTS

We train the AMDT algorithm on multiple datasets from the UCI machine learning repository and also the popular MNIST dataset [33], [34]. The experiments are performed on an AMD Ryzen 9 7950X 5.7 GHz CPU having 128 GB of RAM.

For hyperparameter tuning, a grid search is performed with a range of $[0, 1)$ for λ, $[0.85, 0.995]$ for δ, and $[1, 10]$ for the maximum tree depth. The value of the K parameter is set to 2 during our experiments. While training, each feature of the input is scaled to be between 0 and $2^L - 1$, where L is the bit length of the feature. We use a train-test split ratio of 80:20. We use the memristors synthesized by Goux et al. which have energy utilization and cell width of 10 fJ and 70 nm respectively [35]. Energy utilization of the crossbar design is calculated by multiplying the number of programmable memristors present on the crossbar by 10 fJ. The area is calculated by multiplying the number of rows and the number of columns by 4900 nm^2.

TABLE II
ACCURACY AND ENERGY UTILIZATION COMPARISON BETWEEN UNIVARIATE AND ADAPTIVE MULTIVARIATE DECISION TREES.

	Accuracy			Energy (pJ)		
Dataset	DT [7]	AMDT	Delta	DT [7]	AMDT	Delta
Iris	0.9	**0.93**	0.03	0.58	**0.3**	- 0.28
Wine	0.92	**1.00**	0.08	1.34	**0.56**	- 0.78
Banknote	**1.00**	**1.00**	0	**1.6**	2.64	1.04
Car-evaluation	**0.86**	**0.86**	0	0.18	**0.1**	- 0.08
Ionosphere	0.96	**0.99**	0.03	1.52	**1.08**	- 0.44
Balance-scale	0.79	**0.91**	0.12	8.84	**8.4**	- 0.44
Indian-Diabetes	0.77	**0.8**	0.03	14.92	**0.72**	- 14.2
Tic-tac-toe	**0.96**	0.95	- 0.01	**1.2**	1.3	0.1
Monk1	0.8	**0.92**	0.12	0.84	**0.28**	- 0.56
Statlog-shuttle	**1.00**	**1.00**	0	5.8	**1.38**	- 4.42
MNIST	0.86	**0.88**	0.02	**118.12**	118.13	0.01
Average	0.89	0.93	0.04	14.08	12.3	-1.82

As shown in Table I, on the popular MNIST dataset, SOTA [7] achieves a test accuracy of 0.82 with a crossbar area of 9.40×10^{-4}mm^2 consuming 1.74×10^{-2}nJ of energy. On the other hand, using AMDT the test accuracy increases to 0.88 with a crossbar area of 5.5×10^{-2}mm^2 consuming 1.18×10^{-1}nJ of energy.

Table II shows the comparison between SOTA [7] and AMDT on multiple datasets based on their highest accuracy among the tested 1 to 8 bit length, energy consumption and space utilization. As expected, in a multivariate environment, the average test accuracy of AMDT outperforms the SOTA [7] by 4%. We also manage to use 12.6% less energy, thanks to the intelligent selection performed between univariate and multivariate features in the AMDT algorithm.

V. CONCLUSION

The AMDT training algorithm presented in our work enables the integration of both univariate and multivariate decision nodes which leads to the creation of decision trees that exhibit higher accuracy and superior energy efficiency in crossbar designs when compared to the current SOTA methodologies. These circuits utilize sneak paths and 0T1R memristor crossbars, making them robust against resistance drift and radiation degradation from which other decision tree accelerator hardware suffer. We also experimentally verify that our AMDTs produce flow-based designs that are more accurate and energy-efficient compared to univariate DTs for multiple machine learning datasets.

VI. ACKNOWLEDGMENT

We acknowledge support from NSF Award #2245756 to Sunny Raj.

References

[1] J. Backus, "Can programming be liberated from the von neumann style? a functional style and its algebra of programs," *Communications of the ACM*, vol. 21, no. 8, pp. 613–641, 1978.

[2] S. Rafiq, J. Hazra, M. Liehr, K. Beckmann, M. Abedin, J. S. Pannu, S. K. Jha, and N. C. Cady, "Investigation of reram variability on flow-based edge detection computing using hfo 2-based reram arrays," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 68, no. 7, pp. 2900–2910, 2021.

[3] C. Yakopcic, T. M. Taha, and R. Hasan, "Hybrid crossbar architecture for a memristor based memory," in *NAECON 2014-IEEE National Aerospace and Electronics Conference*. IEEE, 2014, pp. 237–242.

[4] M. A. Zidan, H. A. H. Fahmy, M. M. Hussain, and K. N. Salama, "Memristor-based memory: The sneak paths problem and solutions," *Microelectronics journal*, vol. 44, no. 2, pp. 176–183, 2013.

[5] R. Wille, D. Große, L. Teuber, G. W. Dueck, and R. Drechsler, "Revlib: An online resource for reversible functions and reversible circuits," in *38th International Symposium on Multiple Valued Logic (ismvl 2008)*. IEEE, 2008, pp. 220–225.

[6] S. Thijssen, S. K. Jha, and R. Ewetz, "Compact: Flow-based computing on nanoscale crossbars with minimal semiperimeter," in *2021 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2021, pp. 232–237.

[7] P. Sinha and S. Raj, "Designing energy-efficient decision tree memristor crossbar circuits using binary classification graphs," in *2022 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*, 2022, pp. 1–9.

[8] M. Al Moteri, S. B. Khan, and M. Alojail, "Machine learning-driven ubiquitous mobile edge computing as a solution to network challenges in next-generation iot," *Systems*, vol. 11, no. 6, p. 308, 2023.

[9] ——, "Machine learning-driven ubiquitous mobile edge computing as a solution to network challenges in next-generation iot," *Systems*, vol. 11, no. 6, p. 308, 2023.

[10] D. Puthal, E. Damiani, and S. P. Mohanty, "Secure and scalable collaborative edge computing using decision tree," in *2022 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*. IEEE, 2022, pp. 247–252.

[11] A. Akavia, M. Leibovich, Y. S. Resheff, R. Ron, M. Shahar, and M. Vald, "Privacy-preserving decision trees training and prediction," *ACM Transactions on Privacy and Security*, vol. 25, no. 3, pp. 1–30, 2022.

[12] Y. Izza, A. Ignatiev, and J. Marques-Silva, "On tackling explanation redundancy in decision trees," *Journal of Artificial Intelligence Research*, vol. 75, pp. 261–321, 2022.

[13] M. Kang, S. K. Gonugondla, S. Lim, and N. R. Shanbhag, "A 19.4-nj/decision, 364-k decisions/s, in-memory random forest multi-class inference accelerator," *IEEE Journal of Solid-State Circuits*, vol. 53, no. 7, pp. 2126–2135, 2018.

[14] G. Pedretti, C. E. Graves, S. Serebryakov, R. Mao, X. Sheng, M. Foltin, C. Li, and J. P. Strachan, "Tree-based machine learning performed in-memory with memristive analog cam," *Nature communications*, vol. 12, no. 1, pp. 1–10, 2021.

[15] M. Rakka, M. E. Fouda, R. Kanj, and F. Kurdahi, "Dt2cam: A decision tree to content addressable memory framework," *arXiv preprint arXiv:2204.06114*, 2022.

[16] C. Brodley and P. Utgoff, "Multivariate decision trees," *Machine Learning*, vol. 19, pp. 45–77, 04 1995.

[17] F. Wang, Q. Wang, F. Nie, Z. Li, W. Yu, and F. Ren, "A linear multivariate binary decision tree classifier based on K-means splitting," *Pattern Recognition*, vol. 107, p. 107521, Nov. 2020.

[18] D. R. Cox, "The regression analysis of binary sequences," *Journal of the Royal Statistical Society. Series B (Methodological)*, vol. 20, no. 2, pp. 215–242, 1958. [Online]. Available: http://www.jstor.org/stable/2983890

[19] S. Goswami, A. J. Matula, S. P. Rath, S. Hedström, S. Saha, M. Annamalai, D. Sengupta, A. Patra, S. Ghosh, H. Jani *et al.*, "Robust resistive memory devices using solution-processable metal-coordinated azo aromatics," *Nature materials*, vol. 16, no. 12, pp. 1216–1224, 2017.

[20] S. Goswami, S. Goswami, and T. Venkatesan, "An organic approach to low energy memory and brain inspired electronics," *Applied Physics Reviews*, vol. 7, no. 2, p. 021303, 2020.

[21] S. Pi, C. Li, H. Jiang, W. Xia, H. Xin, J. J. Yang, and Q. Xia, "Memristor crossbar arrays with 6-nm half-pitch and 2-nm critical dimension," *Nature nanotechnology*, vol. 14, no. 1, pp. 35–39, 2019.

[22] J. R. Quinlan, "Induction of decision trees," *Machine Learning*, vol. 1, pp. 81–106, 1986. [Online]. Available: https://api.semanticscholar.org/CorpusID:13252401

[23] F. Wang, Q. Wang, F. Nie, W. Yu, and R. Wang, "Efficient tree classifiers for large scale datasets," *Neurocomputing*, vol. 284, pp. 70–79, 2018. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0925231218300328

[24] H. Guo and S. B. Gelfand, "Classification trees with neural network feature extraction," in *Proceedings 1992 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*. IEEE Computer Society, 1992, pp. 183–184.

[25] W.-Y. Loh and Y.-S. Shih, "Split selection methods for classification trees," *Statistica sinica*, pp. 815–840, 1997.

[26] S. B. Kotsiantis, "Decision trees: a recent overview," *Artificial Intelligence Review*, vol. 39, pp. 261–283, 2013.

[27] P. E. Greenwood and M. S. Nikulin, *A guide to chi-squared testing*. John Wiley & Sons, 1996, vol. 280.

[28] H. Kim and W.-Y. Loh, "Classification trees with unbiased multiway splits," *Journal of the American Statistical Association*, vol. 96, no. 454, pp. 589–604, 2001. [Online]. Available: https://doi.org/10.1198/016214501753168271

[29] B. Van Essen, C. Macaraeg, M. Gokhale, and R. Prenger, "Accelerating a random forest classifier: Multi-core, gp-gpu, or fpga?" in *2012 IEEE 20th International Symposium on Field-Programmable Custom Computing Machines*. IEEE, 2012, pp. 232–239.

[30] T.-W. Chen, Y.-C. Su, K.-Y. Huang, Y.-M. Tsai, S.-Y. Chien, and L.-G. Chen, "Visual vocabulary processor based on binary tree architecture for real-time object recognition in full-hd resolution," *IEEE transactions on very large scale integration (VLSI) systems*, vol. 20, no. 12, pp. 2329–2332, 2011.

[31] K. J. Lee, G. Kim, J. Park, and H.-J. Yoo, "A vocabulary forest object matching processor with 2.07 m-vector/s throughput and 13.3 nj/vector per-vector energy for full-hd 60 fps video object recognition," *IEEE Journal of Solid-State Circuits*, vol. 50, no. 4, pp. 1059–1069, 2015.

[32] R. Thomas, P. Zhu, B. Zumbo, and S. Dutta, "On measuring the relative importance of explanatory variables in a logistic regression," *Journal of modern applied statistical methods: JMASM*, vol. 7, pp. 21–38, 05 2008.

[33] D. Dua and C. Graff, "Uci machine learning repository. university of california, school of information and computer science, irvine, ca (2019)," 2019.

[34] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[35] L. Goux, A. Fantini, G. Kar, Y.-Y. Chen, N. Jossart, R. Degraeve, S. Clima, B. Govoreanu, G. Lorenzo, G. Pourtois *et al.*, "Ultralow sub-500na operating current high-performance TiN\Al2O3\HfO2\Hf\TiN bipolar RRAM achieved through understanding-based stack-engineering," in *2012 Symposium on VLSI Technology (VLSIT)*. IEEE, 2012, pp. 159–160.