PHILOSOPHICAL TRANSACTIONS A

rsta.royalsocietypublishing.org

Research

Article submitted to journal

Subject Areas:
artificial intelligence, electrical engineering, microsystems.

Keywords:
Tsetlin machines, neural networks, AI hardware design, energy efficiency.

Author for correspondence:
Rishad Shafik
e-mail:
Rishad.Shafik@newcastle.ac.uk

Learning Automata based Energy-efficient AI Hardware Design for IoT Applications

Adrian Wheeldon[†], Rishad Shafik[†], Tousif Rahman[†], Jie Lei[†], Alex Yakovlev[†] & Ole-Christoffer Granmo[‡]

[†]Microsystems Group, School of Engineering
Newcastle University, NE1 7RU, UK.
[‡]CAIR, University of Agder, Postboks 422
4604 Kristiansand, Norway.

Energy efficiency continues to be the core design challenge for artificial intelligence hardware designers. In this paper, we propose a new artificial intelligence hardware architecture targeting Internet of Things applications. The architecture is founded on the principle of learning automata, defined using propositional logic. The logic-based underpinning enables low energy footprints as well as high learning accuracy during training and inference, which are crucial requirements for efficient artificial intelligence with long operating life. We present the first insights into this new architecture in the form of a custom-designed integrated circuit for pervasive applications. Fundamental to this circuit is systematic encoding of binarized input data fed into maximally parallel logic blocks. The allocation of these blocks is optimized through a design exploration and automation flow using FPGA-based fast prototypes and software simulations. The design flow allows for expedited hyperparameter search for meeting the conflicting requirements of energy frugality and high accuracy. Extensive validations on the hardware implementation of the new architecture using single and multi-class machine learning datasets show potential for significantly lower energy compared with the existing artificial intelligence hardware architectures. In addition, we demonstrate test accuracy and robustness matching the software implementation, outperforming other state-of-the-art machine learning algorithms.

THE ROYAL SOCIETY
PUBLISHING

1. Introduction

Advances in sensing devices have enabled a shift towards the fourth industrial revolution [1]. The large volume of the data produced by these devices is pushing the technology front of a new generation of artificial intelligence (AI) for Internet of Things (IoT) applications. These applications are expected to infer important decisions in the real world instantaneously rather than offloading data to the cloud servers [2]. Such a step change in technology requires significant strides in *energy efficiency*, which continues to be a primary design challenge for IoT hardware designers [3–5].

Existing AI systems predominantly follow the principle of neural networks (NNs). Originally inspired by Rosenblatt's Neural Automaton in 1957 [6], modern NNs have evolved in complexity across different application domains. Typically NNs define a learning problem by finding the weighted sum of all inputs in the training phase, organized in multiple layers. The weight updates are defined by a normalized activation function and are performed through rigorous gradient descent exercises. When implemented in hardware, the modular electronic neurons require arithmetic-heavy circuits, such as multiply-accumulate (MAC) units. The number of these units can quickly grow with more inputs and added complexity of the learning problem [7]. Given such a scale of arithmetic complexity, achieving required energy efficiency and performance in NNs can be daunting, which is exacerbated further by the large volume of data generated by IoT devices [8].

Over the last two decades, significant progress has been made in energy-efficient NN hardware research. A vast majority of existing works have considered pruning arithmetic complexity to save energy by exploiting the natural resilience of AI applications to minor deviations or error. Examples include precision scaling [9,10], approximate logic designs [11–13], new analog or mixed-signal circuit designs [14] and hardware/software co-design for NNs [15]. Recently, there are overwhelming interests in moving away from arithmetic to using binary logic as the core building blocks. Binarized neural networks (BNNs) are an example of this development. The key goal is to condense advanced AI workloads with low energy footprints. However, this can make the learning process (i.e., accuracy and convergence) sensitive to how gradient descent is designed, which is still arithmetic based [16].

Learning automata, originally defined by Mikhail Tsetlin in 1960s, constitute another class of machine learning algorithm that reinforces current action using the past history. Each action follows the trajectory of a probability distribution which is updated based on the environmental response the automaton obtains by performing a particular action. As the number of actions, and their probability distribution trajectories can have a very large number of combinations, adopting learning automata to machine learning (ML) hardware has been challenging [17,18].

Recently the Tsetlin machine has been proposed as a promising ML algorithm based on learning automata. The Tsetlin machine simplifies the traditional learning automata by discrete-step action updates through Tsetlin automata, defined as the finite automata with linear tactics. For action updates, each Tsetlin automaton uses rewards for reinforcing an action and penalties for weakening the automaton confidence in performing the action. This discretization with linear step updates allows for formulating the learning problem using powerful propositional logic [19], and furthermore, simplifies the learning mechanism enabling efficient on-chip learning. The input data in a Tsetlin machine are encoded in the binarized form as a set of propositional logic variables, called literals. These literals are used to build the logic expressions corresponding to inference classes through ensembles of parallel Tsetlin automata, called *Clauses*, during training [20]. When training is completed, the inference outputs are described by binarized classifications.

The logic-based structure of Tsetlin machines provides opportunities for energy efficient AI hardware design. This will require addressing the major challenges of systematic architecture allocation of low-level resources as well as parametric tuning and data binarization, which cannot be achieved by using high-level synthesis or hardware-assisted acceleration tools. This paper provides the first insights into an AI hardware architecture design using learning automata, addressing the challenges above. Specifically, we make the following **contributions**:

- a new AI hardware architecture capable of on-chip learning, targeting primarily an application-specific integrated circuit (ASIC) implementation;
- a binarization method for encoding data for the proposed architecture;
- an exploration and automation design flow for faster hyperparameter search and hardware optimization using a runtime-reconfigurable FPGA prototype;
- extensive validation experiments using several ML datasets, showing comparative analysis of performance, energy and learning efficacy.

Our aim is to corroborate the principles of learning automata applied in energy-frugal AI hardware design. As such, we will validate the efficiency of the hardware architecture using IoT-scale datasets that are carefully chosen to investigate both single and multi-class applications as well as to study the impact of noisy inputs on the overall learning efficiency. The remainder of this paper is organized as follows. Section 2 introduces the core learning automaton algorithm, leading to the Tsetlin machine. Section 3 presents the design flow and resulting Tsetlin machine hardware architecture. Section 4 discusses the experimental results using the ASIC implementation, while Section 5 reports further results from ML experiments conducted on the FPGA platform. Finally, Section 6 concludes the paper highlighting our future work.

2. Machine Learning using Learning Automata

Figure 1 depicts a schematic of different structural blocks in the learning automaton algorithm. The algorithm adopts of discrete-step updates using linear tactics, proposed by [19]. This enables the algorithm to be constrained by a finite number of states, defined by an ensemble of Tsetlin automata. Input data are defined as a set of binarized features and their complements, called literals (A, Figure 1). The literals are fed into the learning automaton structure through two major parts: one responsible for inference (i.e. classification) and the other for reinforcement and feedback for learning (i.e. training). In the following these parts and their parameters are further detailed.

(a) Inference

The main inference component is the conjunctive clause (B, Figure 1), which uses propositional logic expressions for output classification. The composition of each clause is controlled by a team of Tsetlin automata, each of which has a pre-defined number of states, divided between actions (see Figure 2). The automata decide whether their associated literal should be included in the clause or not, following a number of reinforcement steps (see Section (b)).

Each inference class has a set of clauses associated with it. Each clause produces a vote or no vote for its class. Half of the clauses can vote positively, and half of the clauses can vote negatively. The inclusion of inhibition in the voting system is what enables non-linearity in the inference process. The votes are summed to produce a collective result which gives an indication of confidence. This confidence is used to influence future decisions of the automata (E, Figure 1)

In a single-class inference problem the output layer is a simple thresholding function. If the votes are positive (or zero), the input data is determined to belong to the class. For a negative sum the input data is determined to be *not* in the class. For multi-class problems we replace thresholding with argmax to determine the output class (D, Figure 1). In this case the class summation becomes indicative of confidence and argmax chooses the class with the highest confidence, thus avoiding any ambiguity in classification.

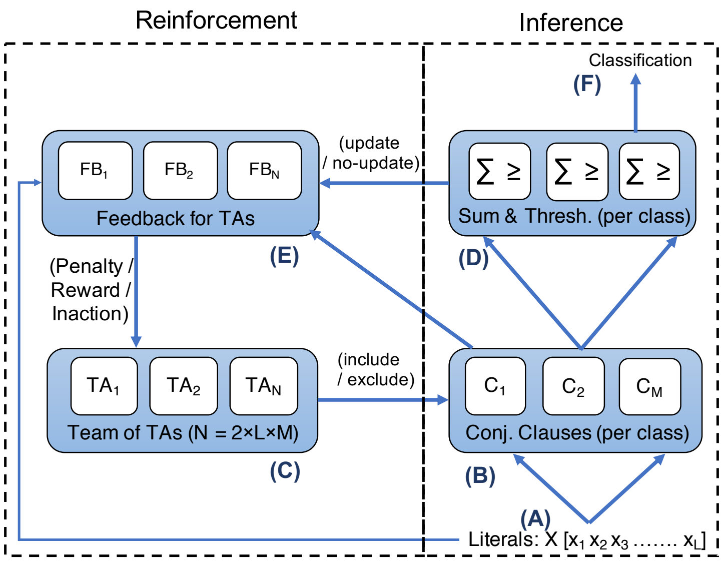


Figure 1. A schematic diagram of discretized implementation of learning automata proposed by [19].

(b) Reinforcement

Fundamental to reinforcement are the team of Tsetlin automata (C, Figure 1). Such automata are also known as automata with linear tactics, to emphasize the fact that they allowed gradual ascent, or reinforcement, in performing a particular action, and equally gradual descent from one action to performing another action. A variety of types of such learning automata have been studied in [21].

In the Tsetlin machine implementation, a two-action Tsetlin automaton is described by the state diagram in Figure 2. The automaton may be given reward, causing it to reinforce the current action decision (e.g., action 1) by moving away from the midstate (i.e., state n in Figure 2). Conversely it may be given a penalty which moves the state towards the decision boundary.

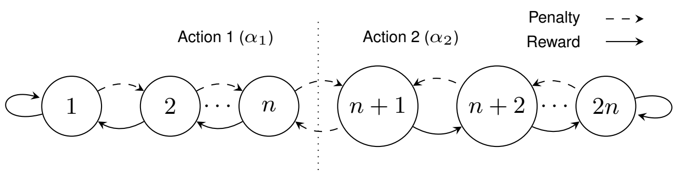


Figure 2. State diagram for the Tsetlin automaton.

In relation to processing the binarized literal through Tsetlin automata within clauses, the two actions are *include* and *exclude*. The update of the automata requires reinforcement through penalty, reward and inaction. The decision on whether to update individual automata is controlled by a number of conditions based on: 1) the values of literals, 2) votes from clauses described earlier, and 3) current actions of Tsetlin automata in terms of *include* and *exclude*. For further details of how algorithm reinforces these updates, please refer to [19].

(c) Parameters

The efficacy of machine learning using learning automata depends on a number of hyperparameters, which must be carefully tuned before training. The process of tuning for accuracy and convergence for a given problem is called *hyperparameter search*. Table 1 shows the Tsetlin machine hyperparameters with their associated symbols used throughout this paper. The numbers of binary inputs and classes are fixed by the problem at hand.

Table 1. Tsetlin machine parameters and their symbols.

Number of binary inputs	N_{Inputs}
Number of classes	N_{Classes}
Number of clauses per class	N_{Clauses}
Number of automaton states	$2n$
Automaton decision boundary	n
Automaton initialization state	ϕ_{Init}
Feedback threshold	T
Learning Sensitivity	s

For large ML problems, software-based hyperparameter search can be computationally expensive, requiring several hours to weeks of iterative computation times. Typically, software based hyperparameter search aims to achieve better accuracy and performance. However, hardware objectives are marginally different as hyperparameters need to be carefully exercised in low-level design configurations for energy-frugality, while achieving an acceptable accuracy. To achieve these objectives as well as for accelerated search, we will use an FPGA-based hardware prototype on ML problems with different dataset sizes. This prototype will also be used for automating the process of faster design exploration, while managing the tradeoffs between power, performance and efficacy.

3. Proposed Hardware Architecture

The hardware architecture inspired by the Tsetlin machine implementation of learning automata (presented in Section 2) is designed exploiting the principle of *maximal parallelism*. Update of all Tsetlin automata and computation of all clauses is executed in parallel. This allows for processing one datapoint (which is a set of the input features concurrently updating all automata in the Tsetlin machine) in a single clock cycle. Figure 3 shows the basic Tsetlin machine inference (i.e. classifier) architecture. This block is duplicated for the number of required classes.

The input to the proposed architecture is a set of binarized Literals. The Literals are organized in pairs of originally encoded binaries and their inverses (i.e. $\text{Literals} = \{\text{Inputs}, \overline{\text{Inputs}}\}$). The learning units (Figure 4) are self-contained and include the automata (TA), feedback generation (FB) and random generation associated with one clause. For inference, only the clause computation itself and include states previously calculated by the automata are required; the rest of the learning unit can be omitted or turned off. In the following different architectural components are described further. For ASIC synthesis, Tsetlin machines parameters are fixed at compile time enabling the lowest area and power possible. For FPGA prototyping, hyperparameters can be adjusted on-the-fly for rapid exploration and optimization.

(a) Binarization

The input Literals are encoded in a binarized form before they are compatible in the proposed architecture's learning and inference steps (Section 2). This binarization process affects the system accuracy, therefore to tune the accuracy to the required level, binarization is included in the

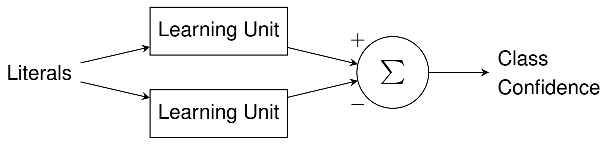


Figure 3. The basic architecture of a two-clause Tsetlin machine classifier.

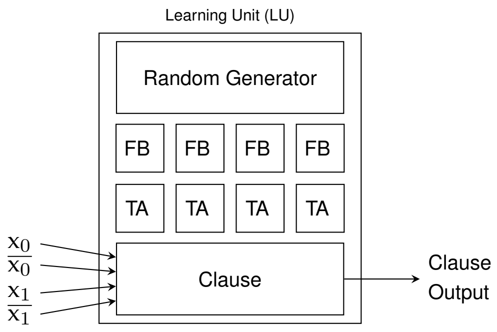


Figure 4. A two input Tsetlin machine learning unit comprising a clause, Tsetlin automaton (TA), feedback (FB) and random source.

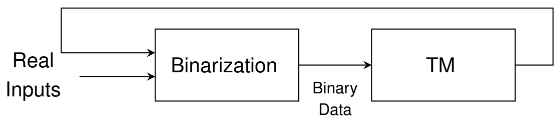


Figure 5. A closed-loop binarization method with accuracy feedback.

feedback loop during training as Figure 5 shows. Using a suitably chosen encoding method, the raw data are encoded with increased binary precision until the required accuracy is achieved.

Existing encoding method uses pre-defined thresholds and precision levels for encoding the raw dataset [22]. This method ignores the statistical significance of data, which defines how output inference classes are correlated with the dataset. For example, in a multi-class ML problem, it is possible that majority of the inference classes are statistically orthogonal and independent. A data significance-agnostic method will not exploit this orthogonality towards reducing the binarized encodings, which will lead to over-provisioning of resources (such as the number of parallel automata and the number of clauses) in the hardware architecture.

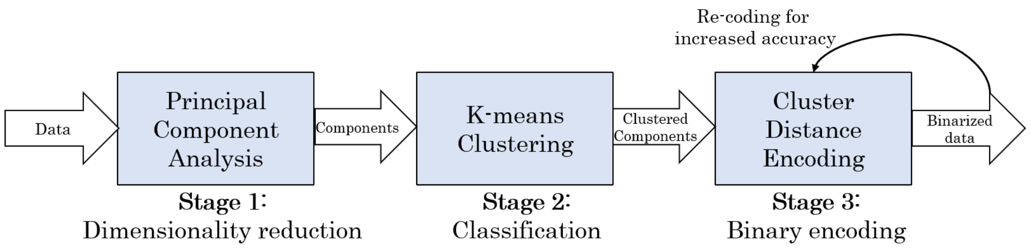

Re-coding for increased accuracy
Data
Principal Component Analysis
Components
K-means Clustering
Clustered Components
Cluster Distance Encoding
Binarized data
Stage 1: Dimensionality reduction
Stage 2: Classification
Stage 3: Binary encoding

Figure 6. A data flowchart of significance-driven binarization method, organised in three stages.

For resource-frugality considerations, we developed a significance-driven binarization method consisting of three stages, as shown in Figure 6. In the following, we briefly describe these three stages.

Stage 1 (Dimensionality Reduction): In the first stage the class labels are stripped off in a given ML dataset and then Principal Component Analysis (PCA) is carried out. PCA generates orthogonal transformation of the dataset to linearly uncorrelated components, defined by eigenvalues that represent their percentage of variance. The outliers in the component definition are discarded, leading to dimensionality reduction.

Stage 2 (Classification): After reducing dimensions through PCA, it is possible that the components will still have eigenvalues indicating high correlation between them. At this point, k-means clustering (kMC) is applied to quantitatively differentiate the data point positions. We chose kMC as it is a fast and scalable method, which can progressively adapt to cluster centroids starting with a random datapoint [23]. In our approach, we use hard clustering approach in kMC, which allocates each PCA point to only one cluster. Since we already know the target class each point belongs to from the original class labels, kMC allows for validating their true significance using orthogonality.

Stage 3 (Binary Encoding): With their true reflection of orthogonality and clusters in Stage 2, the distance between the clusters is determined in this stage, leading to our envisioned binarization method. We use a Clusters to left, Distance from true class, Clusters to the right (CDC) encoding scheme that can uniquely define a cluster, maximally maintaining the orthogonality and dimension reductions obtained from PCA. By identifying the cluster distances, threshold points can be estimated from the raw datasets for binary encoding. For overlapped clusters or higher accuracy needs, the thresholds are re-adjusted or re-coded considering the one or more cluster distances, particularly the overlaps to the left (C) and that to the right (C).

Figure 7 demonstrates the stage outcomes generated by the proposed binarization method, when applied to the Iris dataset[1]. Figure 7(a) shows the graphical representation of the first two principle components. We can see there is a clear classification of the Iris-setosa but an overlap with the other two classes. These classes are then clustered according to their variance, which can be visualised from the Figure 7(b). Note how the points seem to be distributed almost parallel to the y-axis for each class (particularly for the Iris-setosa class). Figure 7(c) depicts how orthogonal and overlapped classes are encoded in a binarized form for the Iris dataset.

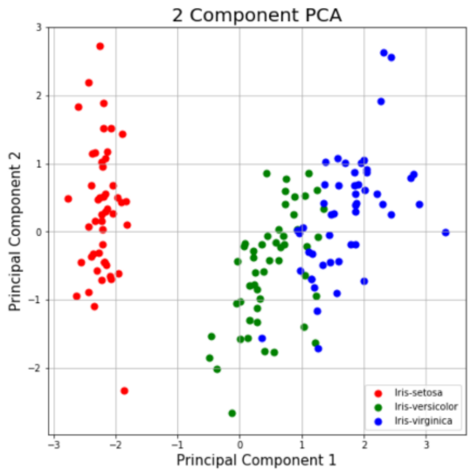
The impact of using a significance-driven binarization method on the machine size cannot be understated. This is because the encoding dictates the number of input features, and the number of automata in the machine scales with $2 \cdot N_{\text{Inputs}} \cdot N_{\text{Clauses}} \cdot N_{\text{Classes}}$. Following on, the number of automata in the machine is directly proportional to the number of inputs (N_{Inputs}). Taking the Iris dataset as an example, existing thresholding technique may produce a 16-bit encoding for data. Our proposed encoding with reduced dimensions and the size of dataset generate a 6-bit encoding with no reduction in inference accuracy, thereby achieving 2.67$\times$ reduction of the machine size.

A full analysis of this method with scalable application to larger ML datasets as well as accuracy-sensitive online optimization of binarization are considered for future work. In this paper we use pre-binarized datasets, proposed by [22].

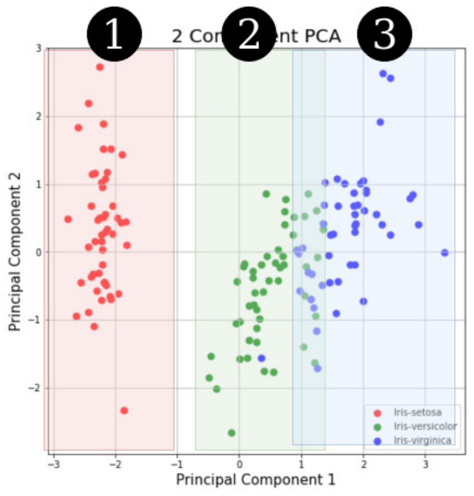
(b) Reinforcement: Tsetlin Automata and Feedback

We implement a specialized version of the original Tsetlin automaton described in Section 2(b). We use $\alpha_2 == 1$ to indicate *include*, and $\alpha_2 == 0$ to indicate exclude. We also introduce the notion of *inaction* for the Tsetlin automata. This means it is possible for neither penalty nor reward to be given to the automaton. In our hardware implementation, each Tsetlin automaton is modeled as a counter. The counter stores an internal state in a register which is tuned based on feedback from the current machine state. In the case of inaction, the state remains unchanged. For penalty

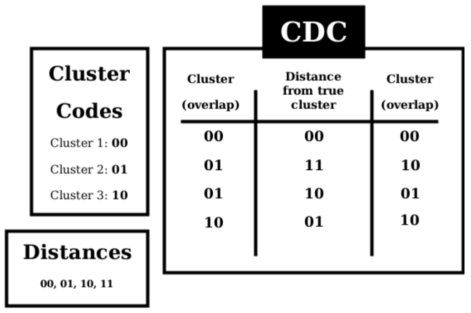
[1]https://archive.ics.uci.edu/ml/datasets/Iris



(a) The first two principle components after Stage 1.



(b) The PCA features in 2 dimensions after Stage 2.



Cluster Codes

Cluster 1: **00**
Cluster 2: **01**
Cluster 3: **10**

Distances

00, 01, 10, 11

CDC

Cluster (overlap)	Distance from true cluster	Cluster (overlap)
00	00	00
01	11	10
01	10	01
10	01	10

(c) Output binaries generated by the CDC encoding scheme after Stage 3.

Figure 7. A visual representation of the stage outputs of data significance-aware binarization method applied to the Binary Iris dataset.

Table 2. Tsetlin automaton internal state tuning.

	Include	Not Include
Penalty	-1	$+1$
Reward	$+1$	-1

or reward, the state (ϕ) is tuned according to Table 2 and saturates according to the bounds $1 \leq \phi \leq 2n, \phi \in \mathbb{Z}$.

The include output is asserted when the automaton internal state exceeds the decision boundary such that Include $= 0$ for $\phi \leq n$ and Include $= 1$ for $\phi > n$. This indicates that the associated literal will be included in the composition of the associated clause. For our hardware it is preferable for the number of possible automaton states to be a power of two so the include output becomes the most significant bit of the state. In other cases a magnitude comparator would be required at the expense of increased logic area.

Penalty and reward are issued to each automaton based on their associated literal, clause output, include state, summation output, expected class and feedback threshold (see [19,22] and Section 2 for further details).

An element of probability is introduced into the state tuning to facilitate diversity of learning among clauses. The probability element is provided by linear feedback shift registers (LFSRs). LFSRs produce a random number sequence in each cycle, which is then compared with a pre-determined sequence to define the probability in the circuit.

After learning is completed in the Tsetlin machine algorithm, the reinforcement logic becomes redundant and only the Include state is needed to perform inference. For our ASIC this means that the Tsetlin automaton registers can be *clock gated* since their contents will not change—that is, the clock signal will be disconnected in order to prevent wasted switching power in the register. Additionally, the feedback and LFSR blocks can be *power gated*, completely removing supply voltage and therefore reducing their quiescent power to zero [24]. These power saving techniques are essential for enabling maximum inference power and energy efficiency (see Section 4).

(c) Inference: Clauses, Voting and Confidence

Figure 8 shows the logic for a two-input clause, implemented following the algorithm (Section 2). The wide AND operation is implemented as a balanced tree of AND gates for minimum path delay. Include for each literal is used to mask the literal, forcing the corresponding AND gate input to one if the literal is to be excluded.

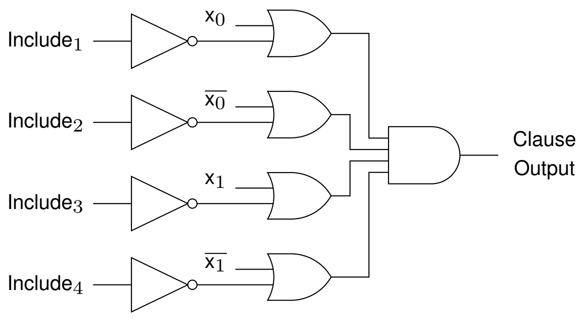


Figure 8. Logic implementation of a two-input clause.

The simplest Tsetlin machine consists of a single class and uses a threshold function to determine whether the input is in the class or not (see Figure 9a). As discussed in Section 2(a), a multi-class Tsetlin machine can be implemented by instantiating multiple Tsetlin classifiers and choosing the class with the greatest confidence using an argmax block (see Figure 9b). Argmax is built using a tree of comparators with accompanying multiplexers which pass through the corresponding argument number. Figure 10 shows the logic implementation of a two-input argmax which is used to build argmax of higher-numbered input. Max and Argmax outputs become x_i and a_i inputs for the next stage.

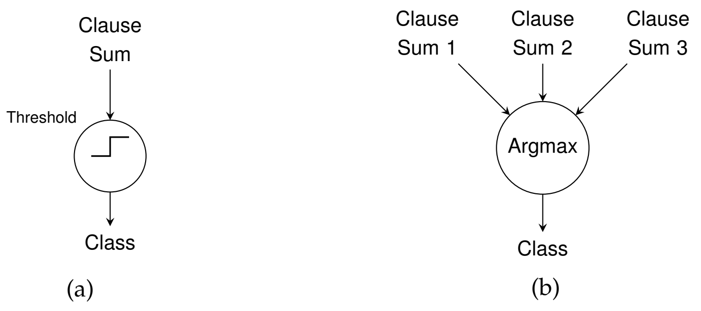


Figure 9. Output functions for (a) single class and, (b) multi-class Tsetlin machines.

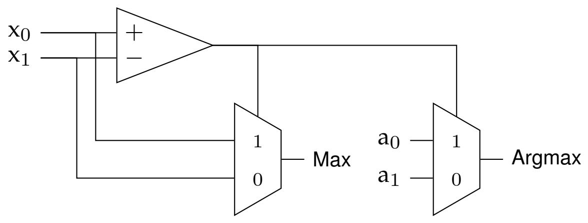


Figure 10. Logic implementation of a two-input argmax.

In Figure 11 we investigate the critical path of the inference hardware—using the hardware for generated for the Noisy XOR problem specifically. Include states for the clauses are already calculated by the automata and do not change once learning has ceased. The path is fully combinational and has a propagation delay of less than one clock cycle.

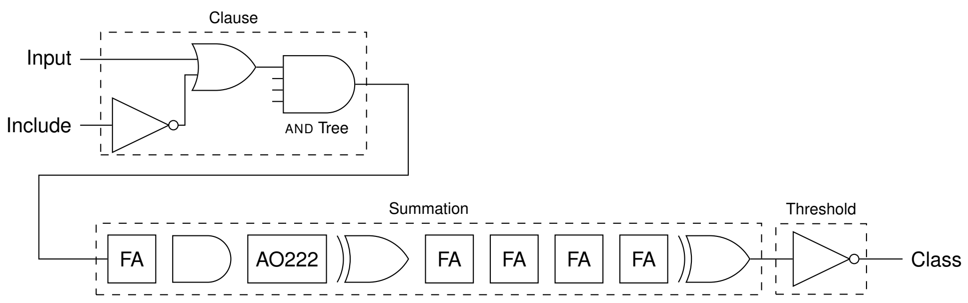


Figure 11. Critical path of the inference hardware.

(d) Design Exploration and Automation

Typically, software based hyperparameter search aims to achieve better accuracy and performance [25]. Hardware hyperparameter search is marginally different as it exercises these parameters in low-level design configurations for energy-frugality, while achieving an acceptable accuracy. As such, we developed an FPGA-based automation platform that we can use to flexibly program the hardware to enable accelerated design validations as well as energy frugality and matching stochasticity using small and readily-available datasets. This will ensure the parameters can be transferred exactly over to the ASIC design and as such will be especially important once we develop hardware-centric Tsetlin machine algorithms which further depart from the software implementation. The FPGA hardware expedites the hyperparameter search process due to high parallelism of the implementation and is capable of running one training cycle in the order of seconds, compared to several minutes for the software implementation on a desktop computer.

Figure 12 shows the design flow. Initially there is some heuristic to choose the hyperparameters based on the number of binary inputs. From there we perform a hyperparameter search which includes a feedback loop to suitably minimize the hardware resources for energy frugality, while also maintaining a high level of accuracy. The final hyperparameters are then used for ASIC synthesis.

An Altera Cyclone V FPGA development board is connected to a host PC through USB connections. A script on the host PC controls the FPGA hardware via Intel Quartus software. The script manipulates the FPGA IO to load the dataset and control the runtime-reconfigurable hyperparameters. After each training cycle the test accuracy is measured and recorded. The hyperparameter search is exhaustive based on a list of possible hyperparameters provided by the user. After the hyperparameter search is complete, the user can choose the preferred hyperparameter configuration based on the accuracy achieved and the resource required for the specific implementation. Changes to the binarization method can be made at this stage if the users requirements are not met. After the final hyperparameters are chosen, these can be input into the final ASIC synthesis by means of Verilog parameters.

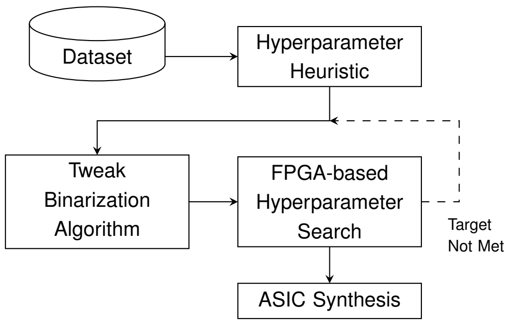

Dataset
Hyperparameter Heuristic
Tweak Binarization Algorithm
FPGA-based Hyperparameter Search
Target Not Met
ASIC Synthesis

Figure 12. Design exploration and automation flow using an FPGA.

For hyperparameter optimization we instantiate a parameterizable Tsetlin machine design on an FPGA and use runtime reconfiguration to disable functional units on-the-fly. This methodology allows us to test many Tsetlin machine configurations in a short time and without resynthesizing or reprogramming the FPGA hardware. Clauses can be disabled by forcing their output to zero, meaning they have no effect on Clause Sum. Their associated automata can be disabled to reduce power by giving constant *inaction*. Our future work includes designing the argmax block that can ignore certain inputs to disable the corresponding class, enabling problems with varying number of class outputs to be optimized on the same FPGA hardware.

Clauses are the main building block of the architecture and also determine the number of automata required in the system. Therefore our main optimization goal for area and power is to minimize the number of clauses. Overall accuracy of the machine depends heavily on the interaction between N_{Clauses} and T hyperparameters. The T parameter makes almost no difference to the hardware size or power, and we therefore optimize T in order to retain as much accuracy as possible with minimum N_{Clauses}. By using an iterative heuristics algorithm, we have been able to reduce the total number of clauses (given by $N_{\text{Classes}} \cdot N_{\text{Clauses}}$) in the Tsetlin machine from 300 to just 60, whilst retaining a test accuracy greater than $92\,\%$. This $80\,\%$ saving in clauses translates almost directly to the same saving in resources for the ASIC implementation. Figure 13 shows the highest test accuracies achieved for different combination of N_{Clauses} and T. It is seen that the larger the T value, the greater the potential to reach a higher accuracy. However as T increases, so does the number of clauses needed to achieve the highest accuracy.

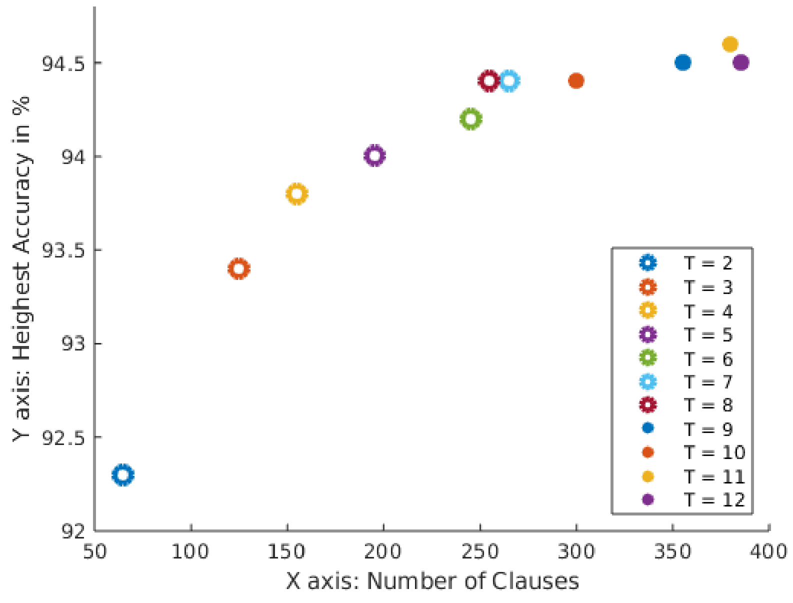


Figure 13. Effect of N_{Clauses} and T hyperparameters on test accuracy after training for 100 epochs.

4. Performance and Energy Efficiency

We test the hardware Tsetlin machine using both $65\,\text{nm}$ ASIC technology with $1\,\text{V}$ nominal supply voltage; and FPGA synthesis for Altera Stratix IV. For our validation experiments, we use widely available ML datasets (also used in [19,22]) to train our hardware (more details follow in Section 5). The Noisy XOR dataset comprises twelve inputs and one class output. This dataset illustrates the robustness of the Tsetlin machine learning algorithm and can be used as a first test to ensure functionality with a modest hardware size. Table 3 presents the Noisy XOR results *post-synthesis* (without performing any layout) for ASIC and FPGA constrained to $100\,\text{MHz}$ clock frequency. The Binary Iris dataset is a multi-class flower detection task and is representative of IoT applications. For this test we complete ASIC layout using CADENCE INNOVUS for a $65\,\text{nm}$ low power technology, giving high-effort power and area figures including scan-chain, IO and clock-tree consumption which we will later compare with other low-power hardware alternatives. We have observed that the energy consumption figures scale linearly with the number of datapoints in the dataset as well as the number of clauses in the Tsetlin machine architecture. More datapoints require extra compute cycles, while additional clauses increase the spatial data processing needs.

Noisy XOR has a much larger training set (with 5000 datapoints) than Binary Iris (with 120 datapoints). However, as only 2 out of 12 binarized features contribute to the learning formulation it requires less training epochs to obtain a good accuracy, when compared with Binary Iris. As the Noisy XOR implementation features $\approx 4\times$ higher clock frequency than Binary Iris, both datasets exhibit similar training times between them. Binary Iris consumes more area, power and energy due to its larger Tsetlin machine structure—requiring $N_{\text{Clauses}} = 20$ compared to Noisy XOR's $N_{\text{Clauses}} = 10$. The FPGA implementation for Noisy XOR performs similarly to its ASIC counterpart, however it suffers from high power mainly due to interconnect overhead.

Table 3. Results of training for ASIC synthesis in 65 nm technology and runtime-reconfigurable FPGA hardware for two different datasets: a single class Noisy XOR and a multi-class Binary Iris. As expected, the FPGA prototype implementations return significantly higher energy compared to that of ASIC.

	Noisy XOR		Binary Iris
	ASIC Synthesis	FPGA	ASIC Layout
N_{Inputs}, N_{Clauses}, N_{Classes}	12, 10, 1		16, 20, 3
ASIC Area	0.246 mm^2	—	0.386 mm^2
Frequency	118 MHz	110 MHz	33.3 MHz
Training Time	4.24 ms	4.55 ms	1.80 ms
Avg. Power (Training)	16.8 µW	1.10 W	1.85 mW
Energy/Datapoint (Training)	142 fJ	10.0 nJ	55.6 pJ
Energy/Datapoint (Inference)	—	—	30.6 pJ

The logic based structure in Tsetlin machine allows for low-complexity, energy-efficient learning and inference. This is a major differentiator when compared with the state-of-the-art neural network based AI. Table 4 compares the Tsetlin machine energy efficiency with 3 recently reported neural networks approaches: a mixed-signal neuromorphic approach using time-domain arithmetic organized in a spatially unrolled neuron architecture [26], a low-power FPGA-based convolutional BNN (CBNN) approach that uses XNOR adder based integer weight biases to reduce the arithmetic-heavy batch normalization for synchronization between the deeper layers [27] and finally an in-memory BNN approach using parallel content-addressable memories (CAMs) to reduce the frequent data movement costs [28]. Our comparative analysis considered disparities between these approaches in terms of: a) their internal structures both in combinational and sequential parts, and b) the size of datasets used to validate the efficiencies. To maximally avoid any bias in the presence of these disparities, we normalize the energy

Table 4. Energy efficiency of the proposed Tsetlin machine architecture compared with NN implementations.

	Neuromorphic [26]	CBNN [27]	BNN [28]	Proposed
Technology	65 nm	65 nm	65 nm	65 nm
Voltage	1.0 V	1.1 V	1.1 V	1.0 V
Features	Time-domain	No normalization	Parallel CAMs	Logic based
Architecture	Spatially unrolled	XNOR based	Pipelined	Clause selection
Infer. Energy	48.2 Top/J	25.2 Top/J	88.5 Top/J	**62.7 Top/J**
Train. Energy	—	—	—	**34.6 Top/J**

efficiency figures in terms of the number of atomic data operations (a set of multiply-additions in the case of NNs and AND logic followed by argmax in the case of Tsetlin machine) per unit energy, expressed as Tera operations per Joule, $\mathrm{Top/J}$. The Tsetlin machine (TM) energy efficiency is estimated by dividing the post-synthesis energy per datapoint (Table 3) by the product $(2 \cdot N_{\text{Inputs}} \cdot N_{\text{Clauses}} \cdot N_{\text{Classes}})$ and then normalizing that to $\mathrm{Top/J}$. As can be seen, the inference Tsetlin machine (which is fully digital) outperforms the highly specialized BNN approaches by up to $2.5\times$ ($62.7\,\mathrm{Top/J}$). This efficiency is enabled by the lean propositional logic within Tsetlin automaton followed by majority voting between clauses as well as power gated reinforcement blocks, such as random generation and Tsetlin automaton update circuits. Power gating the key reinforcement circuits causes the slack times to increase significantly, which makes it possible to either improve the inference performance by scaling the operating frequencies up or increase the energy efficiency further by aggressive voltage scaling. The NN approaches depend on parallel binary operations in multiple layers with a set of pre-trained weights and their biases and as such their arithmetic and data movement operations contribute to higher complexities during inference.

The training energy efficiency of Tsetlin machine is lower ($34.6\,\mathrm{Top/J}$) than its inference energy efficiency (Table 4). This is because the reinforcement building blocks, such as random generation and Tsetlin automaton update circuits, are now powered on and crucial. Worth noting here, the Tsetlin machine training energy efficiency is still considerably high (although no training energy efficiency figures were reported for the NN implementations for comparisons). The complexity of Tsetlin machine reinforcement building blocks is significantly lower than the NN approaches which depend on parallel multiply-add operations in multiple layers followed by gradient-descent based weight updates. This efficiency during training in Tsetlin machine can be exploited for emerging IoT applications where continuous on-chip learning is crucial for adapting to environmental changes at the microedge. Our future research includes architectural support for on-chip continuous learning.

Among other comparative examples, Hirtzlin et. al. [29] achieves $524\,\mathrm{fJ}$ per clock cycle in $28\,\mathrm{nm}$ technology for the *basic cell* which makes up their BNN architecture including MRAM and registers. This compares with our *clause* building block which achieves $0.661\,\mathrm{fJ}$ per clock cycle in $65\,\mathrm{nm}$ technology, which however does not contain any memory elements. Another example of work in this area is [8] which claims $0.4\,\mathrm{pJ}$ per operation for their NN MAC implemented in $28\,\mathrm{nm}$ FD-SOI technology. It should be noted that this specialized low-power technology gives a significant advantage over the $65\,\mathrm{nm}$ node used in our work. In the area of hyperdimensional computing, Karunaratne et. al. [30] demonstrate a system in $65\,\mathrm{nm}$ silicon capable of $430\,\mathrm{nJ}$ *per query*. Here, a *query* is a unit of inference datapoint comprising a natural language sentence. The data structure of each *query* is organized in the form of hypervectors for parallel in-memory interfaces.

In Figure 14 we illustrate the immense power and time advantages of the ASIC Tsetlin machine implementation, compared to more off-the-shelf embedded platforms. We run the same Iris dataset benchmarks across the three platforms: software Tsetlin machine running on a Raspberry Pi 3 (featuring ARM Cortex-A53 cores with 1GB LPDDR2 memory), hardware implemented on a low-power FPGA development board (Digilent Cmod A7-35T), and finally our custom ASIC hardware. For accuracy measurements with a high degree of confidence, we run the training over 100 epochs for all Tsetlin machine implementations. For each experiment, the training times are calculated as the latency per epoch. We measure the power consumption of the software and FPGA-based Tsetlin machine implementations using a precision DC power analyzer (Keysight Tech. model: N6705C).

As can be seen from Figure 14(a), the test accuracy of the software is slightly higher than the hardware. This is attributed to the differences in random number generation between the platforms. Software random number generation uses significantly higher precision than that in the hardware implementation which manifests a more well-defined stochasticity [31]. Both hardware platforms utilize the same psuedo-random number generation technique (LFSR) and

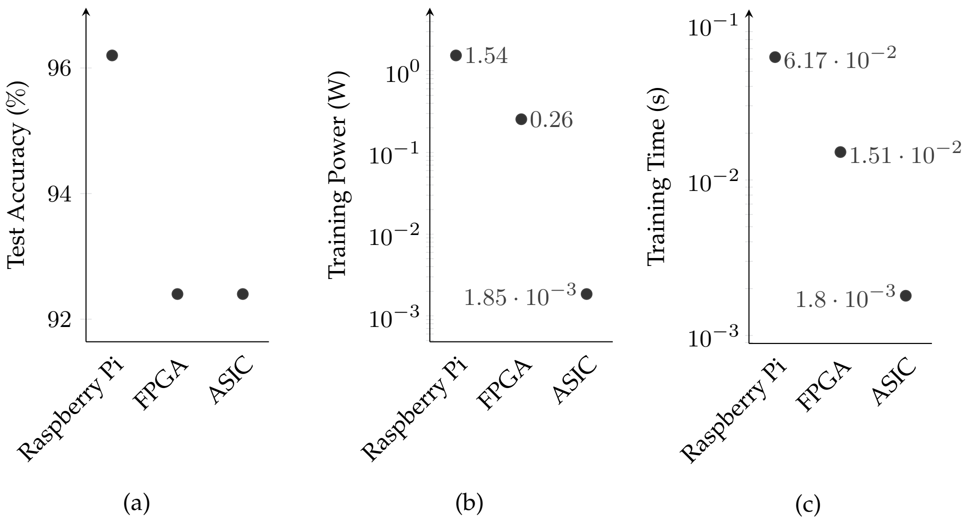


Figure 14. Comparison of software (Raspberry Pi) and hardware (FPGA, ASIC) platforms for (a) test accuracy, (b) training power and, (c) training time.

therefore exhibit equal accuracy. According to Figure 14(b), the FPGA platform shows improved power and time over the Raspberry Pi since it is free from operating system and extraneous peripheral overheads. The ASIC however, shows several orders-of-magnitude lower power consumption again. The ASIC has no reconfiguration overhead or unused logic to leak power. Additionally, logic gates are free to be placed close to each other in order to optimize critical logic path lengths, allowing higher speed computation and faster training/inference times (see Figure 14(c)). In contrast, the FPGA must configure internal wires to connect the already-placed logic gates.

5. Machine Learning Experiments

To observe learning and inference behavior closely, we ran a number of experiments on the proposed hardware architecture using ML datasets: XOR, Noisy XOR and Binary Iris.

The Noisy XOR dataset contains 12 binary inputs—two of which are related by XOR with the remaining 10 inputs randomized. The training set provides 5000 examples and has 40 % of the outputs inverted for added noise, and for this reason, training accuracy is limited to 60 %. More details of the noise immunity of the TM are available in [32]. The test set provides a further 5000 examples, this time without output inversions, meaning 100 % test accuracy is theoretically possible. Figure 15a shows the Tsetlin machine achieving 58.8 % and 95.8 % accuracy in training and test sets respectively after only 10 epochs.

In Figure 15b we explore how the number of states affects inference accuracy. There is a lower bound on the number of action states ($2n$) for each dataset below which the Tsetlin automata do not provide enough state space to be able to distinguish features of the dataset.

Figure 15c shows how the number of clauses per class influences the inference accuracy. For successful learning, the number of clauses must be sufficient to capture the features of the input data and enable an ensemble learning effect. The turning point for learning of the Noisy XOR dataset is at 12 clauses. As we increase the number of clauses further the accuracy increase tends toward 100 % with some variation which is attributed to the stochastic nature of Tsetlin automata feedback.

Table 5 shows that the ASIC Tsetlin machine can replicate the training and test accuracy of the software Tsetlin machine when using the same configuration (16, 100, 3). This validates the hardware, allowing it to act as a basis for future hardware-specific Tsetlin machine architectures.

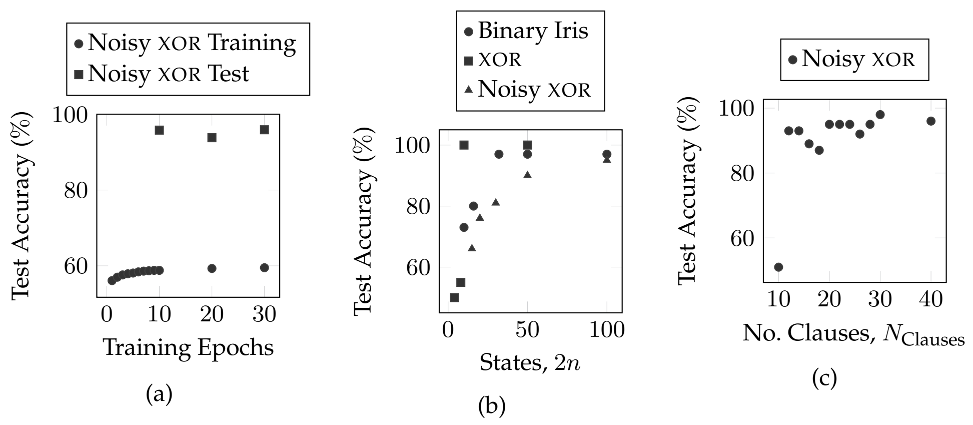


Figure 15. (a) Learning convergence of the Tsetlin machine, (b) Tsetlin automaton action states vs. inference accuracy after 10 epochs, (c) Number of clauses per class vs. inference accuracy after 10 epochs.

Table 5. Comparison of hardware- and software-based Tsetlin machines and XGBOOST in the Iris dataset. Here, both hardware and software Tsetlin machines use the same configuration (16, 100, 3).

Tsetlin machine	Accuracy (%)	
Implementation	Train	Test
Software (Desktop PC)	97.3	95.7
Software (RPi)		96.2
ASIC (This Work)	96.3	97.0
XGBOOST	98.3	96.7

All Tsetlin machine implementations are theoretically capable of the same accuracy, however variances in the stochasticity of each implementation result in variations in system accuracy. All Tsetlin machine implementations exceed the test accuracy of XGBOOST [33], a renowned lean and high-accuracy tree boosting ML system, proving that the Tsetlin machine algorithm is competitive with other state-of-the-art ML algorithms.

6. Conclusions

The paper presents the first ever AI hardware design method using the principles of learning automaton. The method leverages the natural ability of an ensemble of Tsetlin automata to learn from a set of training data. The overall framework of a collective of Tsetlin automata lends itself to energy frugality (cf. the principle of a least action!) for inferences based on boolean *fabric* used for solving classification problems. This was the initial hypothesis for our research and this paper corroborates that through our proposed AI hardware architecture for IoT-scale applications. We also demonstrated the advantages of our hardware design method by comparing power, accuracy and performance figures with software Tsetlin machine implementations on a number of embedded platforms as well as recently reported BNN implementations.

Our hardware implementations in the form of an ASIC benefited from a fast design flow using an FPGA prototype. The design flow facilitated hyperparameter search to achieve energy efficiency, while also retaining a high level performance and learning efficacy. We believe that the proposed AI hardware architecture is a crucial step towards packing complex AI systems with on-chip learning capability, particularly suitable for applications that require continuous learning. Future work includes the development of a scalable hardware architecture to enable larger ML problems using advanced architectural allocations and VLSI design techniques.

Acknowledgement

The authors would like to acknowledge the funding and support from the EPSRC Platform Grant STRATA (EP/N023641/1) and EPSRC Impact Acceleration projects: "Event-Driven Survivable AI Hardware Design" and "Whisperable". The authors gratefully appreciate the interactions and discussions with colleagues: Thanasin Bunnam of Microsystems Group and Jonathan Edwards of Temporal Computing.

References

1. K. Schwab, *The fourth industrial revolution*. London : Portfolio Penguin, 2017.
2. N. Javaid, A. Sher, H. Nasir, and N. Guizani, "Intelligence in IoT-Based 5G Networks: Opportunities and Challenges," *IEEE Communications Magazine*, vol. 56, no. 10, pp. 94–100, October 2018.
3. A. Biswas and A. P. Chandrakasan, "Conv-RAM: An energy-efficient SRAM with embedded convolution computation for low-power CNN-based machine learning applications," in *2018 IEEE International Solid - State Circuits Conference - (ISSCC)*, Feb. 2018, pp. 488–490.
4. M. Chen, Y. Miao, X. Jian, X. Wang, and I. Humar, "Cognitive-LPWAN: Towards Intelligent Wireless Services in Hybrid Low Power Wide Area Networks," *arXiv e-prints*, Sep. 2018.
5. E. O. Neftci, "Data and power efficient intelligence with neuromorphic learning machines," *iScience*, vol. 5, pp. 52 – 68, 2018.
6. F. Rosenblatt, *The perceptron, a perceiving and recognizing automaton Project Para*. Cornell Aeronautical Laboratory, 1957.
7. S. Draghici, "Neural networks in analog hardware—design and implementation issues," *International journal of neural systems*, vol. 10, no. 01, pp. 19–42, 2000.
8. L. Benini, "Plenty of room at the bottom: Micropower deep learning for cognitive cyberphysical systems," *RTNS Keynote*, 2017.
9. E. Wang, J. J. Davis, R. Zhao, H.-C. Ng, X. Niu, W. Luk, P. Y. Cheung, and G. A. Constantinides, "Deep neural network approximation for custom hardware: Where we've been, where we're going," *ACM Computing Surveys (CSUR)*, vol. 52, no. 2, p. 40, 2019.
10. R. Andri, L. Cavigelli, D. Rossi, and L. Benini, "YodaNN: An ultra-low power convolutional neural network accelerator based on binary weights," in *2016 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, 2016, pp. 236–241.
11. R. Shafik, A. Yakovlev, and S. Das, "Real-power computing," *IEEE Transactions on Computers*, vol. 67, no. 10, pp. 1445–1461, 2018.
12. I. Qiqieh, R. Shafik, G. Tarawneh, D. Sokolov, S. Das, and A. Yakovlev, "Significance-driven logic compression for energy-efficient multiplier design," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 8, no. 3, pp. 417–430, 2018.
13. V. Mrazek, S. S. Sarwar, L. Sekanina, Z. Vasicek, and K. Roy, "Design of power-efficient approximate multipliers for approximate artificial neural networks," in *2016 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 2016, pp. 1–7.
14. S. Cawley, F. Morgan, B. McGinley, S. Pande, L. McDaid, S. Carrillo, and J. Harkin, "Hardware spiking neural network prototyping and application," *Genetic Programming and Evolvable Machines*, vol. 12, no. 3, pp. 257–280, 2011.
15. G. Zhong, A. Dubey, C. Tan, and T. Mitra, "Synergy: A HW/SW framework for high throughput cnns on embedded heterogeneous SoC," *CoRR*, vol. abs/1804.00706, 2018.
16. Y. Takefuji and K. Lee, "An artificial hysteresis binary neuron: A model suppressing the oscillatory behaviors of neural dynamics," *Biological Cybernetics*, vol. 64, no. 5, pp. 353–356, 1991.
17. B. J. Oommen and J. P. R. Christensen, "ϵ-optimal discretized linear reward-penalty learning automata," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 18, no. 3, pp. 451–458, 1988.
18. K. S. Narendra and M. A. L. Thathachar, "Learning automata - a survey," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. SMC-4, no. 4, pp. 323–334, July 1974.

19. O.-C. Granmo, "The Tsetlin machine - a game theoretic bandit driven approach to optimal pattern recognition with propositional logic," *arXiv e-prints*, Apr. 2018. [Online]. Available: http://arxiv.org/abs/1804.01508
20. R. Shafik, A. Wheeldon, and A. Yakovlev, "Explainability and dependability analysis of learning automata based AI hardware," in *Proceedings of the IEEE International On-line Test Symposium (IOLTS)*. IEEE, 2020.
21. K. S. Narendra and M. A. L. Thathachar, *Learning Automata: An Introduction*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1989.
22. O.-C. Granmo, "Introduction to the Tsetlin machine," University of Agder, Norway, Tech. Rep., 2019.
23. T. Kanungo, D. M. Mount, N. S. Netanyahu, C. D. Piatko, R. Silverman, and A. Y. Wu, "An efficient k-means clustering algorithm: Analysis and implementation," *IEEE transactions on pattern analysis and machine intelligence*, vol. 24, no. 7, pp. 881–892, 2002.
24. J. M. Rabaey and M. Pedram, *Low power design methodologies*. Springer Science & Business Media, 2012, vol. 336.
25. S. Rahimi Gorji, O.-C. Granmo, A. Phoulady, and M. Goodwin, "A Tsetlin machine with multigranular clauses," *arXiv preprint arXiv:1909.07310*, 2019.
26. D. Miyashita, S. Kousai, T. Suzuki, and J. Deguchi, "A neuromorphic chip optimized for deep learning and CMOS technology with time-domain analog and digital mixed-signal processing," *IEEE Journal of Solid-State Circuits*, vol. 52, no. 10, pp. 2679–2689, 2017.
27. H. Yonekawa and H. Nakahara, "On-chip memory based binarized convolutional deep neural network applying batch normalization free technique on an FPGA," in *2017 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*. IEEE, 2017, pp. 98–105.
28. W. Choi, K. Jeong, K. Choi, K. Lee, and J. Park, "Content addressable memory based binarized neural network accelerator using time-domain signal processing," in *Proceedings of the 55th Annual Design Automation Conference*, 2018, pp. 1–6.
29. T. Hirtzlin, B. Penkovsky, M. Bocquet, J. Klein, J. Portal, and D. Querlioz, "Stochastic computing for hardware implementation of binarized neural networks," *IEEE Access*, vol. 7, pp. 76 394–76 403, 2019.
30. G. Karunaratne, M. Le Gallo, G. Cherubini, L. Benini, A. Rahimi, and A. Sebastian, "In-memory hyperdimensional computing," *Nature Electronics*, 2020.
31. K. S. Narendra and M. A. L. Thathachar, "Learning Automata - A Survey," *IEEE Trans. Syst. Man. Cybern.*, vol. SMC-4, no. 4, pp. 323–334, Jul. 1974. [Online]. Available: http://ieeexplore.ieee.org/document/5408453/
32. K. D. Abeyrathna, O.-C. Granmo, L. Jiao, and M. Goodwin, "The regression tsetlin machine: A tsetlin machine for continuous output problems," in *Progress in Artificial Intelligence*, P. Moura Oliveira, P. Novais, and L. P. Reis, Eds. Cham: Springer International Publishing, 2019, pp. 268–280.
33. T. Chen and C. Guestrin, "XGBoost: A scalable tree boosting system," *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining - KDD '16*, 2016. [Online]. Available: http://dx.doi.org/10.1145/2939672.2939785