REDRESS: Generating Compressed Models for Edge Inference Using Tsetlin Machines

Sidharth Maheshwari, Tousif Rahman, Rishad Shafik, *Senior Member, IEEE*, Alex Yakovlev, *Fellow, IEEE*, Ashur Rafiev, Lei Jiao, *Senior Member, IEEE*, and Ole-Christoffer Granmo

***Abstract*—Inference at-the-edge using embedded machine learning models is associated with challenging trade-offs between resource metrics, such as energy and memory footprint, and the performance metrics, such as computation time and accuracy. In this work, we go beyond the conventional Neural Network based approaches to explore Tsetlin Machine (TM), an emerging machine learning algorithm, that uses learning automata to create propositional logic for classification. We use algorithm-hardware co-design to propose a novel methodology for training and inference of TM. The methodology, called REDRESS, comprises independent TM training and inference techniques to reduce the memory footprint of the resulting automata to target low and ultra-low power applications. The array of Tsetlin Automata (TA) holds learned information in the binary form as bits: $\{0, 1\}$, called excludes and includes, respectively. REDRESS proposes a lossless TA compression method, called the include-encoding, that stores only the information associated with includes to achieve over 99% compression. This is enabled by a novel computationally minimal training procedure, called the Tsetlin Automata Re-profiling, to improve the accuracy and increase the sparsity of TA to reduce the number of includes, hence, the memory footprint. Finally, REDRESS includes an inherently bit-parallel inference algorithm that operates on the optimally trained TA in the compressed domain, that does not require decompression during runtime, to obtain high speedups when compared with the state-of-the-art Binary Neural Network (BNN) models. In this work, we demonstrate that using REDRESS approach, TM outperforms BNN models on all design metrics for five benchmark datasets viz. MNIST, CIFAR2, KWS6, Fashion-MNIST and Kuzushiji-MNIST. When implemented on an STM32F746G-DISCO microcontroller, REDRESS obtained speedups and energy savings ranging 5-5700× compared with different BNN models.**

***Index Terms*—Machine learning, tsetlin machine, learning automata, logic-based learning, binary neural networks, edge inference, automata re-profiling.**

Manuscript received 19 October 2022; revised 28 March 2023; accepted 12 April 2023. Date of publication 19 April 2023; date of current version 4 August 2023. This work was supported in part by the U.K. Northern Accelerator under Grant NACCF 220, in part by the Lloyds Registers Foundation under Grant 5th ICON-12, and in part by Norwegian Research Council under Grant AI Everywhere project. Recommended for acceptance by V. Morariu. *(Sidharth Maheshwari and Tousif Rahman contributed equally to this work.) (Corresponding author: Sidharth Maheshwari.)*

Sidharth Maheshwari, Tousif Rahman, Rishad Shafik, Alex Yakovlev, and Ashur Rafiev are with the Microsystems Research Group, School of Engineering, Newcastle University, NE1 7RU Newcastle upon Tyne, U.K. (e-mail: sidharth.maheshwari@newcastle.ac.uk; s.rahman@newcastle.ac.uk; rishad.shafik@newcastle.ac.uk; alex.yakovlev@newcastle.ac.uk; ashur.rafiev@newcastle.ac.uk).

Lei Jiao and Ole-Christoffer Granmo are with the Centre for Artificial Intelligence Research (CAIR), University of Agder, 4879 Grimstad, Norway (e-mail: lei.jiao@uia.no; ole.granmo@uia.no).

This article has supplementary downloadable material available at https://doi.org/10.1109/TPAMI.2023.3268415, provided by the authors.

Digital Object Identifier 10.1109/TPAMI.2023.3268415

I. Introduction

THE emergence of AI in sensor based systems has empowered greater model functionality thus allowing for substantial advances in intelligent wearables, personalized healthcare [1] and smarter and more sustainable cities [2]. To address the burden of long inference times and meeting embedded device resource constraints, offloading the data for cloud computation is often the method of choice [3], [4]. This comes with the added issues of privacy, increased latency and network connectivity.

To enable true wide-spread integration of AI sensors for low and ultra-low power edge inference there is a need for focused design effort towards delivering energy efficient and memory frugal implementations [5], [6], [7]. The prevailing approach to edge inference is based on Neural Network (NN) models [5], [8]. However, for a given inference application, designers are forced to choose intelligent trade-offs through hardware-software co-design considerations as the deep neural network (DNN) models are resource hungry in terms of storage, runtime memory (RAM) and computation. These trade-offs stem from finding the balance between two key considerations: the memory and compute limitations of the target platform and selecting a trained model that achieves a competitive accuracy within acceptable latency [9]. Recently, dedicated Application Specific Integrated Circuits (ASIC) have been designed and validated for ultra-low power edge inference using NN variants [10], [11]. They present custom hardware capable of operating at near-threshold voltages with power-gating to reduce active chip power during operations.

Several approaches have already been considered for easing NN based edge transition. This includes network pruning [12], [13] to reduce the number of parameters and reduce model complexity, weight quantization [9], [14] to alleviate the compute intensity of floating point arithmetic or layer decomposition [15] to allow for model compression. These methods all tackle one significant and unavoidable challenge: the arithmetic nature of Neural Networks. Moreover, there are practical limitations of black box ML models being used in critical applications to make high-stake decisions [16] which suggests exploration of interpretable models instead [17]. This work considers a fundamentally different machine learning (ML) architecture called the

Tsetlin Machine (TM) [18]. The TM addresses the aforementioned challenges by removing floating-point arithmetic from training and inference routines. The learning element of the TM is the Tsetlin Automata (TA), which decides whether to include or exclude a feature while creating logic propositions to the learning problem. The includes and excludes are represented by bits '1' and '0', respectively. The logic-based underpinning enables bit-wise operations, in the inference routine, between boolean input features and include-exclude decisions making it computationally less intensive and energy frugal.

While TM offers the advantage of logic based inference, the memory footprint of trained models is usually quite significant [19]. A naive implementation would use the model as it is, which makes it impractical for edge inference. Each TM contains user-specified number of clauses each of which hold the TA that forms logic proposition. As the classification problem complexity increases, so does the number of logic propositions (or clauses) required by the TM. All TM related work clearly demonstrate that more clauses lead to better model accuracy with a large associated impact on memory. However, there is no deterministic and definitive method of selecting the optimum number of propositions per class for a given problem to minimize the memory footprint. MILEAGE [20] is the first attempt at determining the number of clauses or logic propositions sufficient to classify a given dataset. It proposes an automated method to minimizes the number of clauses at runtime depending on their contribution towards correct classification. Although it demonstrates that high-accuracy can be achieved using relatively fewer clauses, the model size still remains large for edge-inference. Therefore, to reap the advantages of logic over arithmetic computation when comparing against NNs in inference, TM's require substantial model compression.

This work proposes **REDRESS**: a novel lea**R**ning **ED**ge infe**R**ence methodology for **E**mbedded t**S**etlin machine**S**. REDRESS's primary objective is to find the optimum balance between the TM model size, accuracy and latency. To achieve this, the methodology first employs an automated TM architecture search paradigm to find the optimal model size and hyperparameters to best represent the classification problem. This is accomplished by automated training of multiple candidate TM configurations. The user can examine the resulting models to decide which model should proceed to the next stage. The next stage involves a computationally minimal REDRESS training procedure involving Tsetlin Automata Re-profiling, which attempts to increase the sparsity of useful automata decisions (the number of include decisions) but provides an optimally trained model with high accuracy. REDRESS proposes a lossless compression approach called include-encoding to compress the TA by storing only the information associated with includes, achieving over 99% compression in terms of memory footprint for all benchmark datasets. Finally, REDRESS proposes a bit-parallel inference methodology that operates on TA in the compressed domain and performs fast multi-class classification. The key difference between MILEAGE and REDRESS is that MILEAGE aims to reduce the model size by reducing the number of clauses while REDRESS aims to minimize the model using sparsity of includes irrespective of the number of clauses.

The study of TM in this work mainly targets microsystems with resource constraints for, e.g., IoT or edge inference applications, which fall in the low and ultra-low power category with limited storage and computational capacity. The scope of the paper is therefore limited to smaller datasets with Boolean features such as images encoded with Booleans. The validation is performed on STM32F746G-DISCO micro-controller that offers 1 Mbytes of flash memory and 340 Kbytes of runtime memory (RAM) with an ARM Cortex-M7 core. Deploying large models trained for CIFAR 10/100 or ImageNet is not possible on such a platform and lies outside the scope of this paper. Other variants of TM have been evaluated with larger datasets such as CIFAR10 and CIFAR100 where they have demonstrated accuracy up to 75% and 45%, respectively [21]. TM is an actively evolving field of research where novel architectures and training methods are being developed.

Through the REDRESS workflow we propose the following contributions:

- A novel TM training approach using TA Re-profiling procedure to find the best trade-off between model accuracy and memory footprint.
- An Include-Encoding based lossless TA compression technique.
- A fast inference algorithm capable of using bit-parallel arrangement to perform many image classifications simultaneously, achieving order-of-magnitude speedups.
- Extensive validation of the REDRESS-based TM vis-à-vis the state-of-the-art Binary Neural Networks [9] using several ML benchmark datasets on STM32 micro-controller.

II. Binary Neural Network (BNN)

The BNN, first proposed by Courbariaux et al. [22], is the state-of-the-art NN variant used for edge inference and embedded applications [9]. It performs the most extreme quantization of features and weights to a single bit, where the weights are passed through a sign function that converts them to ± 1. The negative weights are clamped to -1 and positive weights are clamped to 1, and are represented with 0 and 1 respectively. BNN enables fast computation by replacing 32-bit weights that require multiply accumulate operations to a 1-bit weights, utilizing `xnor` and `popcount` [9], [22]. This is similar to the TM clause output and class sum computation. A comparative analysis of TM and NN based approaches can be found in [23].

The use of the sign function poses challenges in training during back-propagation given that gradient will be zero and thereby eliminating the possibility of gradient descent. To accommodate for this Courbariaux et al. retain the real value weights and binarize them each time for the forward pass and a straight-through-estimator is used to update the real weights during the backward pass. This is a key difference between DNN and BNN. For details interested readers should refer to [22], [24]. To further optimize the effectiveness of the feed forward process, McDanel et al. propose the combination of the BNN

TABLE I
TABLE OF KEY TERMS FOR EXPLAINING TM TRAINING AND INFERENCE AND THE REDRESS METHODOLOGY

Symbol(s)	Definition
M	Number of Classes
N	Number of Clauses per Class
f	Number of Boolean Features
l	Number of Boolean Literals ($2\times f$)
(s, T)	TM training hyperparameters
W	Word Length for storing compressed TAs
D	Dataset
τ	Number of datapoints

layer computation with batch normalization (if specified) and the activation function. The temporary results are stored as binary outputs which further reduce the memory footprint in embedded implementation [9]. It is enabled by re-ordering the operations such that instead of computing all intermediates of a layer, the layer is computed in chunks where the batch normalization and activation function can be applied to this chunk and produce on the final output, thus, trading extra computation effort for a reduced memory footprint. To date this is the best performing method of deploying BNN to a resource constrained device, hence, has been used in this paper as gold standard for comparison.

BNN and TM are two completely different approaches to learning data. BNN is very similar to multilayered or DNN as it uses floating-point operations, backpropagation and multiple layers to learn input data. TM is single layered and the training process uses learning automata devoid of floating-point operations with randomised feedback process. In BNN, all neurons receive feedback while in the TM the clauses and the automata that receive feedback are selected randomly. In this paper, we have trained BNN in different configurations including single layered FC-128 which closely resembles the TM and shows that it outperforms FC-128 in computation time and energy.

III. TSETLIN MACHINES

The Tsetlin Machine is a machine learning algorithm that relies on the principles of learning automata called *Tsetlin Automata* and game theory to create logic propositions for classification. Theoretical proof of TM's capability to solve complex pattern recognition problems and derivations of propositional formulas and its alignment with Nash equilibrium can be found in [18]. Interested readers can find the proof of convergence of TM in [25], [26] and further details on theoretical aspects of TM in [27], [28], [29], [30]. In this section, we visualize the details of the working principle and actual implementation of TM algorithm. A visual depiction is extremely helpful in understanding the REDRESS approach and facilitating TM adoption, exploration and knowledge sharing. In Section III-A, we examine the pre-processing methodology for input literals and the inference process to highlight the logic driven nature of the TM. In Section III-B, we present the training methodology used by TM to illustrate the reduced computational intensity in comparison with the gradient descent based NN approaches. Table I collates the key terms that will be used across these subsections and the subsequent sections.

A. Data Preparation and Inference

Fig. 1 demonstrates the inference routine for TM. It shows how raw data is prepared through a *Booleanization* method, this Boolean data is then linked to the learning elements, the TA, by computation of a *Clause* output. Multiple clause outputs are passed through a voting system, the class with the most votes is the inferred class.

Data Preparation: The TM requires *Boolean Literals* as inputs to the model. To transform raw data into *Boolean Literals* a Booleanization process is used, as demonstrated in the upper left corner of Fig. 1. Raw features that are of integer or floating point value are passed through a Booleanizer that compares it with a pre-determined threshold and generates a single bit boolean value of 0 or 1 called the *Boolean Features*. In the example presented in Fig. 1, the threshold is chosen arbitrarily for demonstration proposes, however, generally the threshold(s) can be created as per the designer's choice. For example, a threshold can be decided as a mid-point in the raw feature range or chosen through off-the-shelf adaptive thresholding techniques for images. It should be noted that the number of bits used to represent the Booleanized data is user and application dependent. It allows the designer control over the granularity of the input space seen by the TM. In Fig. 1, we have used a single bit to store the Booleanized data but the granularity required by datasets used in this work is discussed in detail in Section VII. The *Boolean Features* are then expanded into *Boolean Literals* by including their complements. By using both the features and their complements the *Boolean Literal* space can represent every possible value that each *Boolean Feature* can acquire. In Fig. 1, we have presented the Booleanization process using two Boolean features that produce four literals to be used as inputs to the TM.

Clause Computation: The main computation component in the inference is the clause output which interprets *Boolean literals* using the TA. Fig. 1 presents the clause computation process using the four literals, obtained from the previous Booleanization step, in conjunction with the TA that pass through the proposition logic consisting of `NOT`, `OR` and `AND` gates. Each literal is assigned a corresponding TA in each clause. Each automaton in our diagram has 6 states, the three on the left are the exclude states and the other three on the right are the include states. If the state of the automaton is exclude then its output will be a bit '0', else if the state is include then the output will be a bit '1'. The number of states that each TA can have is a design choice. Having a larger number of states will increase the granularity of the decision making. We will explore this later in Section VI. The positions of TA states shown in Fig. 1 assume that they have settled in near optimum positions after training. During inference the positions of TA can no longer transition and, therefore, have a fixed include or exclude decision. Through the logic circuitry for the clause, we create a logic proposition that relates the literals to the TA state decision and generate a 1-bit *Clause Output*.

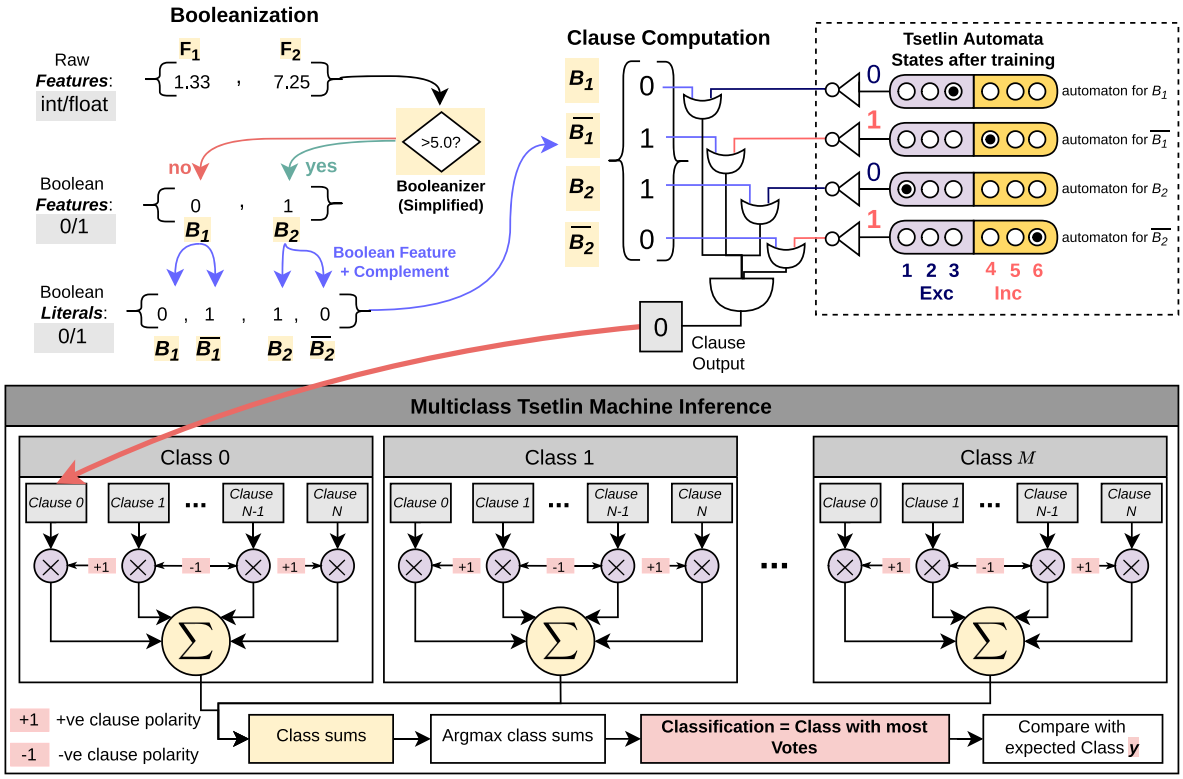


Fig. 1. Visualization of Multiclass Tsetlin Machine inference process. Booleanization stage prepares the input features from raw data. Clause computation presents the logic operations involved in generating the Clause Output from input features and trained automata. The inference routine illustrates the summation of the N Clause Outputs multiplied with their polarity for M classes to generate the respective Class sums. The one with the largest class sum is the classification output. Here, F_i is the i^{th} input features and B_i and $\overline{B_i}$ are the corresponding i^{th} Boolean features and its complement derived from the Booleanizer.

Structure and Inference: To classify a problem with multiple classes, we need the Multiclass TM model. Since most problems have multiple classes, we will use TM and Multiclass TM interchangeably in this work. In Fig. 1, we present a hypothetical example of a Multiclass TM of M classes with N clauses each containing 4 TA per clause. In actual implementation the number of TA per clause will mirror the number of input literals l $(= 2 \times f)$, where f is the number of Boolean input features.

An important design choice determining the size of TM and compute complexity is the number of clauses N specified by the user for any application. This is illustrated in the bottom half of Fig. 1. For every input datapoint,[1] which includes l *Boolean literals*, we compute a 1-bit *Clause Output* using the TA of the respective clauses. Each class must contain an even number of clauses, as each clause carries a polarity of $+1$ and -1 alternatively. The clause polarity determines whether the 1-bit *Clause Output* is multiplied with $+1$ or -1 before they are summed together to determine the *Class Sum*. The clause polarity indicates whether the clause will support or oppose the classification, i.e., a $+ve$ polarity clause will support the classification while the $-ve$ polarity clause will oppose the classification. With the (*Class Sums*) for all the classes, an

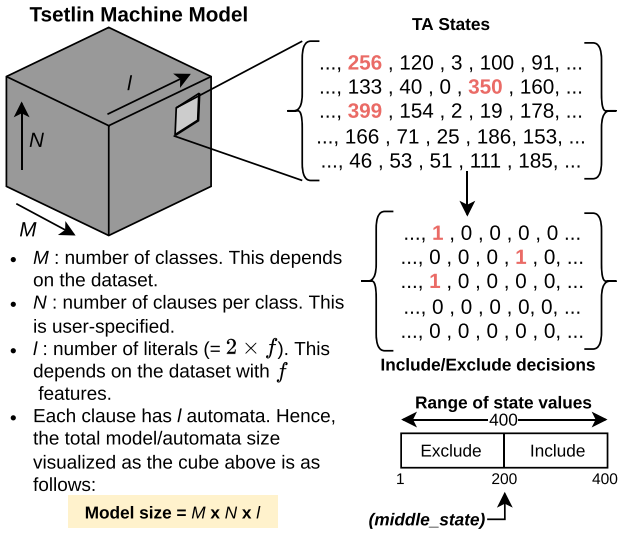


Fig. 2. Visualization of a trained TM model of size $M \times N \times l$. The range of values TA states can acquire is [1,400]. Any value >200 is regarded as include while the rest are excludes.

$argmax$ function is used to determine the largest value and determines the classification for the given datapoint.

Fig. 2 visualizes a trained TM model with model/automata size of $M \times N \times l$ (see Table I). Each class has the same

[1]A datapoint can be any instance of information that needs to be classified by a trained model, for example, an image or audio clip.

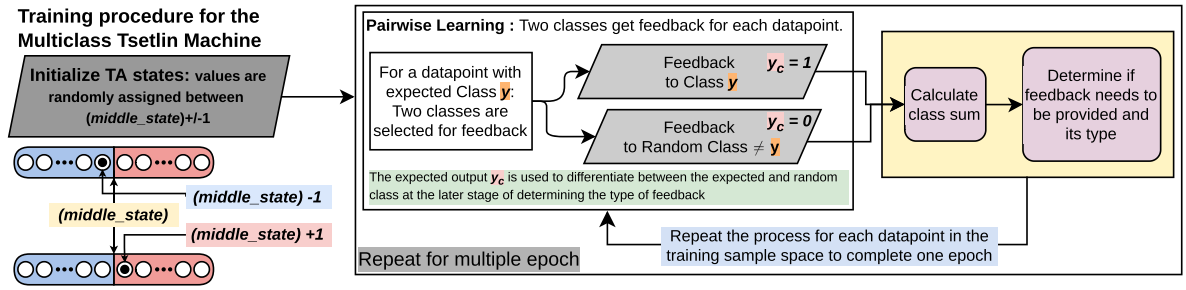


Fig. 3. Overview of the training procedure followed in Multiclass Tsetlin Machine classification. It begins with the initialization of the Tsetlin Automata with one of two randomly chosen values: $middle\_state \pm 1$. In an epoch, the training procedure iterates over all the datapoints in the training sample space. For each datapoint, two classes receive feedback, one of them is the expected class and the second is any randomly chosen class other than the expected class. They are indicated by expected output $y_c \in (0, 1)$, with $y_c = 1$ indicating the feedback is to the expected class while $y_c = 0$ indicating the feedback to random class.

number of user-specified clauses N and each clause consists of l automata, where l is the number of literals. For a dataset with f features $l = 2 \times f$. In Section VI-C, we discuss how an appropriate value of N is selected for a particular dataset. The range of values that TA states can acquire is $[1, 400]$, with any value > 200 is considered as include and the rest as exclude. The range of values TA can acquire is a user design choice. From our experimentation and the available literature, mostly the ranges of $[1, 200]$ or $[1, 400]$ are used and are sufficient for all encountered classification problems. The difference between the two is discussed in Section VI. In Fig. 2, the include-exclude decisions corresponding to TA state values can be seen. The includes are highlighted and purposefully shown relatively sparse compared with excludes as is, generally, the case. The sparsity of TA will be discussed further in Section VI.

B. Training Tsetlin Machines

The rationale behind the training process is to find an optimum combination of state positions for the TAs across the model to achieve high classification accuracy. TM provides *Feedback* to each automaton for its state transition. The TM is a supervised learning algorithm, therefore, TM feedback procedure is that it is independent of classification output obtained at the end of TM inference in Fig. 1. Instead it requires the class sum and the actual class the input datapoint belongs to.

Fig. 3 presents an overview of the training procedure. The process starts with initialization of TA states randomly to one of the two values: $middle\_state \pm 1$. Using uniform random number generator, initialization gives every automaton an equal chance of becoming an include or exclude. Let us assume a training sample space S_{train} consisting of τ datapoints: $(X, y) \in S_{train}$. $X \in (0, 1)^l$ is a binary literal vector of length l belonging to class y, referred to as the expected class. In each epoch, the feedback process iterates for all τ datapoints. In an M multiclass TM, each class is assigned one TM that consists of N clauses each with l automata. For each datapoint X_i, a pairwise learning approach is employed where two classes are selected for feedback. One of them belongs to the expected class y_i as in $(X_i, y_i) \in S_{train}$, while the other is any randomly chosen class $\neq y_i$. To differentiate between the two selected classes they are assigned an expected output value $y_c = 1$ or 0, as shown in Fig. 3. y_c is used to determine the type of feedback at a later stage, which will be discusses as follows.

Fig. 4 shows the feedback procedure within a selected class using decision tree. The feedback procedure requires two user defined hyperparameters viz. s and threshold T. For the selected class, $class\_sum$ is calculated using the inference procedure discussed in Section III-A. While training the $class\_sum$ needs to be clipped to lie within the threshold range of $[-T, T]$ such as: $Class\_sum = clip(class\_sum, [-T, T])$. The clipped $Class\_sum$ plays a crucial role in determining the probability of a clause getting feedback using T and a random number comparison as shown in Equation **C1** and **C2** in Fig. 4. This ensures that all clauses have equal chance of getting feedback and are randomly selected. A clause can get one of the two types of feedback, viz. the Type I and Type II, depending on the clause polarity and y_c. For Type I feedback, the hyperparameter s and the random number are used to determine the probability of an automaton transitioning state, be it increment or decrement in the state value, as shown in Equations *S1* and *S2* in Fig. 4. The clause output and the literal determine whether the state value will be incremented or decremented. The Type II feedback is straight forward and always increments the state value if clause output is 1 and the automaton is exclude. The hyperparameters will be discussed further in Section VI.

The Type I feedback combats false negatives and Type II feedback combats false positives. Ideally, for accurate classification, the class sum of the expected class must be higher than that of all other classes. By randomly selecting another class, all other classes have an equal chance to calibrate their TA for smaller class sums. Class sum increments if a $+ve$ polarity clause output is 1 and decrements if $-ve$ polarity clause output is 1. Therefore, to obtain a high class sum more $+ve$ clauses need to have an output 1 and $-ve$ clause outputs should be 0. Similarly, it is vice-versa if a smaller class sum is desired. As shown in Fig. 5, the Type I and Type II feedback attempts to change the clause outputs to achieve desired class sums. The figure presents a qualitative comparison of the motives and effects of both the types of feedback. To change the clause outputs, the feedback will alter the number of automata that identify as includes. The number of includes is extremely crucial to the TM model memory footprint as we will see in Sections IV and VI.

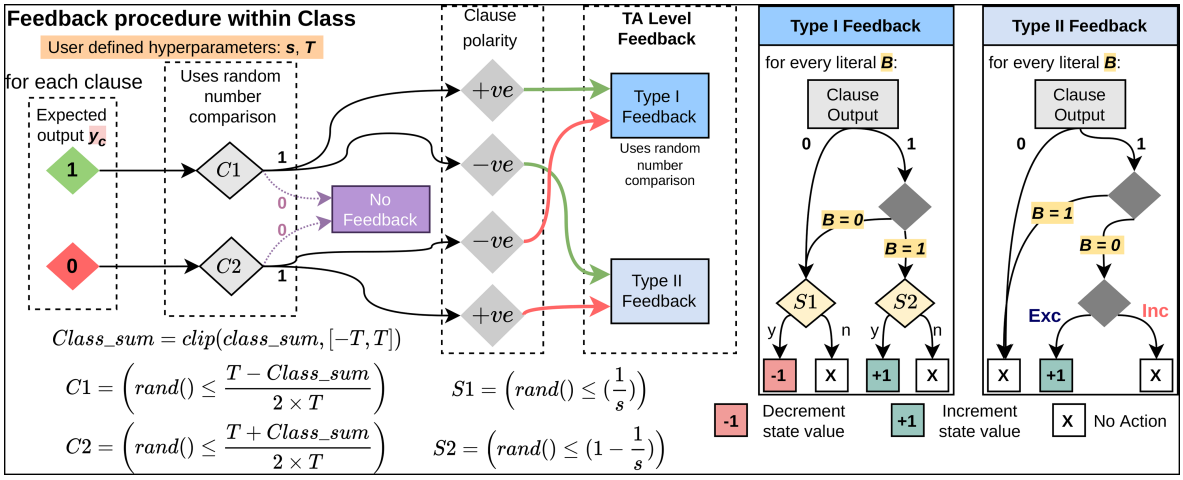


Fig. 4. Overview of the feedback procedure within the class. There are two types of feedback, viz. Type I and Type II. The type of feedback a clause gets depends on the hyperparameters, clause polarity, expected output (y_c), and probabilities *C1* and *C2*. *C1* and *C2* determine if a clause will get any feedback based on the comparison with random number, generated from uniform distribution. Similarly, in the Type I feedback, random number is used to determine the probability of an automaton getting increment or decrement in state value. y - Yes, n - No, Inc - Include, Exc - Exclude, T - Threshold hyperparameter and s - the second hyperparameter, as mentioned in Table I.

Expected class (y_c = 1)		Random class (y_c = 0)	
+ve polarity clause	*-ve clause polarity*	*+ve polarity clause*	*-ve clause polarity*
Type I	**Type II**	**Type II**	**Type I**
attempts to make *clause_output = 1*	attempts to make *clause_output = 0*	attempts to make *clause_output = 0*	attempts to make *clause_output = 1*
if *clause_output = 0* then reduces includes.	if *clause_output = 1* leads to more includes	if *clause_output = 1* leads to more includes	if *clause_output = 0* then reduces includes.
if *clause_output = 1* then may lead to more includes.	if *clause_output = 0* no change	if *clause_output = 0* no change	if *clause_output = 1* then may lead to more includes.

Fig. 5. A comparison of motives and effects of Type I and Type II feedback on clause output and number of includes.

IV. Include-Encoding

The proposition logic used for clause computation is shown again in Fig. 6(a). Here, we observe an interesting property of TM, where only the literals corresponding to includes contribute to the clause output. Assuming Exc means exclude, bitwise OR with 1 ($=$ NOT Exc) will invariably result in output of 1 and does not affect the output of the following AND gate. In the hypothetical example shown in Fig. 6(a), only the literals B_1 and $\overline{B_2}$ determine the clause output as their corresponding automata are includes. In almost all datasets trained using standard TM, the TA has been found to be extremely sparse consisting of $>99\%$ excludes. Considering the fact that $<1\%$ of TA are includes and only contribute in the clause computation, its reasonable to individually store only include-related information. Fig. 6(b) shows l TA corresponding to l literals and their respective offsets (or addresses). We encode these offsets along with some additional information in a 16-bit integer. It should be noted that only includes determine the clause outputs and we encode and store all includes, therefore, no loss of information occurs in the proposed compression scheme making it lossless.

Fig. 6(c) presents the detailed explanation of the encoded include. The additional information indicates the polarity of the clause and demarcate the change of clause while computing clause output. The encoded includes are stored serially, hence, it is required to know the polarity of the clause it belongs to, alongside clubbing all includes belonging to one clause together using 2nd most significant bit (MSB). Fig. 6(c) presents three cases to show how the first two MSB bits flip depending on the serial number of clause and its polarity. If a clause with all excludes is encountered then there is nothing to store, hence, the 2nd MSB flips only when the next include is encountered in the 3rd clause as shown in Fig. 6(c). Using Including-Encoding scheme, 2 Bytes is required to store each include. Hence, the memory footprint of the TM model becomes directly proportional to the number of includes. In Section VI, we present the REDRESS training procedure using TA re-profiling to address this challenge. We will present the details on compression achieved using the proposed encoding scheme in Section VIII. Although not specifically encoded, the least significant bit (LSB) in Fig. 6(c), by default, indicates the literal polarity. Literal polarity is used during inference to access the literals from the features of an input datapoint in the testing dataset and will be discussed along with the proposed inference algorithm in Section III-A. The include-encoding scheme stores only includes, hence, if more clauses with all excludes are encountered then they can be skipped entirely resulting in higher compression. The algorithm scans through the TAs once, and while, scanning it keeps track of clause changes, thus, no extra overhead is involved when more classes with only excludes are encountered.

V. REDRESS Inference

The data structure obtained following the Include-Encoding method comprises of two arrays: one holds the number of

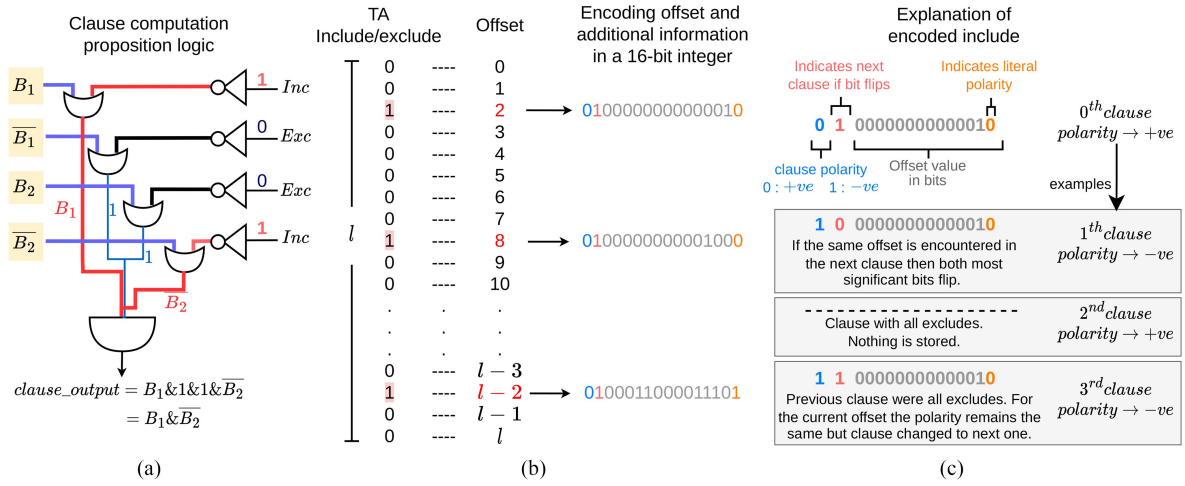


Fig. 6. Overview of the Include-Encoding methodology. (a) Clause output from the Tsetlin Machine proposition logic depends only on literals whose corresponding automata are includes. (b) The encoding of include offsets, in a clause, into 16-bit integers with two most significant bits (MSB) holding additional information. (c) The MSB holds the information on clause polarity and the bit next to it indicates the change of clause. Four consecutive clauses are shown to explain the flipping of two MSB bits. If an all exclude clause is encountered then the 2nd MSB bit does not flip, it flips only when an include is encountered in the next clause. Inc - Include, Exc - Exclude and B_i and $\overline{B_i}$ are i^{th} literal and its complement.

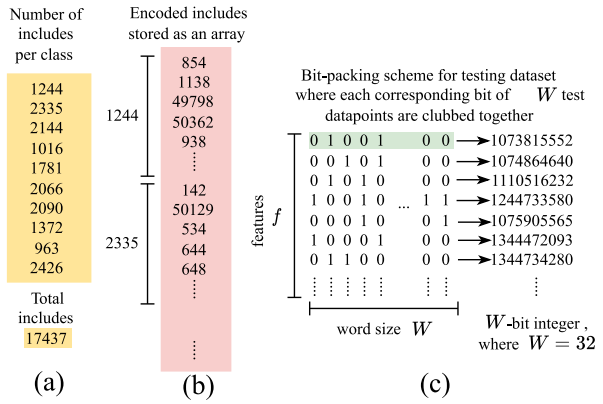


Fig. 7. Data structure and input required in proposed REDRESS inference algorithm. (a) The array with number of includes per class. (b) The array of encoded includes. (c) A bit-packed testing dataset which clubs the corresponding bits of W datapoints into a W-bit integer. f - the number of Boolean features.

includes per class and the second holds the encoded includes as shown in Fig. 7. The input datapoints from the testing dataset are bit-packed by forming W-bit integers from the corresponding bits of W datapoints. As we will see, this enables classification of W images simultaneously taking advantage of the maximum wordsize, which is assumed here to be 32 as we validate TM models on a 32-bit micro-controller. The input datapoints only contain the features and literals are obtained during runtime by complementing them such as: $\{l_{i0} = f_i, l_{i1} = \overline{f}_i\}$. The LSB of encoded include indicates if the corresponding literal is l_{i0} or l_{i1}, where i is the feature offset and 0 or 1 indicates the literal polarity, respectively.

Algorithm 1 presents the pseudocode of the proposed REDRESS inference algorithm. Inference of W datapoints is performed simultaneously in each iteration and hence, the total number of iterations performed is $\lceil \tau/W \rceil$, where τ is total number of datapoints in the testing dataset. The algorithm is based on the assumption that $W = 32$ and 16 bits or 2 bytes are used to encode each include which justifies the constants appearing in the pseudocode. However, theoretically different values of W can be used depending on the maximum word size of the underlying hardware.

The algorithm iterates through the array of encoded includes `IncEnc` once for every W datapoints. This enables parallel inference of W datapoints and significantly reduces the total computation time. While iterating through the `IncEnc` array, the algorithm, basically, scrolls through all the includes in the model. It uses the offset from the `IncEnc` to identify the literals corresponding to includes in each clause, and `AND`s them to calculate the clause outputs, simultaneously, for W datapoints. When the end of the clause is encountered, the clause outputs are then added/subtracted to the class sums depending on +ve or -ve clause polarity, respectively. The algorithm keeps track of the maximum class sum and the corresponding class. $literal\_polarity$ is used in line 21 to determine if the literal is the input feature or its complement as it is needed to compute the clause output. Comments in the Algorithm 1 provide further explanation to its implementation.

VI. REDRESS Training

A. Hyperparameter: T

The hyperparameter T (or threshold) plays a crucial role in classification accuracy and the number of includes. Section III-B and Fig. 4 show that the class sum is clipped to $[-T, T]$ and cannot exceed the threshold in magnitude. Equations $C1$ and $C2$, shown again in Fig. 8, determine the probability that a clause will get feedback, i.e., either Type I or Type II. Fig. 8 presents the average probability of a clause getting feedback for the possible range of class sums with respect to $T = 20$. It assumes that all classes obtain the same number of feedback hits $\tau \times \alpha$ for τ training samples and a certain constant α. We can

Algorithm 1: REDRESS Inference Algorithm Pseudocode.

Input: M, N, f, W, τ, num_of_includes, cl_polarity_bit, offset_bits, cl_change_bit
/* cl - clause. τ - test datapoints. M - classes. N - clauses. f - features. cl_polarity_bit= 15. cl_change_bit= 14. offset_bits = 16383 - bits holding the literal offset. */
Output: inferred_class
Data: Encoded Includes IncEnc, Includes per class Inc_per_class, Input test dataset inp_f

$W \leftarrow 32$ // assuming 32-bit word size
for $k = 0$ **to** $\lceil \tau/W \rceil$ **do** // W inferences simultaneously
  for $m = 0$ **to** W **do** // Initialization
    $class\_sum[m] = -2147483647$ // lowest 32-bit integer
    $cur\_class\_sum[m] = 0$; $classification[m] = 0$
  $inc\_count = 0$
  for $i = 0$ **to** M **do**
    $cl\_output = 4294967295$ // highest unsigned 32-bit integer
    $prev\_cl = 0$; $curr\_cl = 0$; $cl\_polarity = 0$
    $inc\_curr\_class = inc\_count +$ Inc_per_class$[i]$
    for $j = inc\_count$ **to** inc_curr_class **do** /* iterates through includes in class i to calculate the i^{th}class sum of W datapoints simultaneously */
      $curr\_cl =$ IncEnc$[j] >>$ cl_change_bit
      if $curr\_cl \neq prev\_cl$ **then**
        // if clause changed then add previous clause's vote to class sum
        for $m = 0$ **to** W **do**
          $cur\_class\_sum[m] = cur\_class\_sum[m] + ((1-(2 \times cl\_polarity)) \times ((cl\_output >> (31-m))\&1))$
        $cl\_output = 4294967295$ // reset as next clause output will be computed
      $cl\_polarity =$ IncEnc$[j] >>$ cl_polarity_bit
      $feature\_offset =$ IncEnc$[j]$ & offset_bits
      $literal\_polarity =$ IncEnc$[j]$ & 1
      $prev\_cl = curr\_cl$
      $cl\_output = (literal\_polarity == 0)$ **?** $(cl\_output$ & inp_f$[(k \times f) + feature_offset])$ **:** $(cl\_output$ & $(\sim$ inp_f$[(k \times f) + feature_offset]))$ // $\sim$ - logic NOT operation
    $inc\_count = inc\_count +$ Inc_per_class$[i]$
    for $m = 0$ **to** W **do** // find max class sum & note the class
      if $class\_sum[m] < cur\_class\_sum[m]$ **then**
        $class\_sum[m] = cur\_class\_sum[m]$
        $classification[m] = i$
      $cur\_class\_sum[m] = 0$
  for $m = 0$ **to** W **do** // store the classification results of W datapoints
    inferred_class$[(k \times W) + m] = classification[m]$

see that for the expected class the probability reduces to zero if $Class\_sum = T$, implying that no further change will happen in TA profile of the expected class. Similarly, random class with $Class\_sum = -T$ will stop getting feedback to prevent further deterioration of its TA profile vis-à-vis the expected class, as they already can be clearly distinguished with high accuracy due to the large difference in their class sums. At this point, a higher T would allow continued feedback even though not required.

Ideally, it is assumed that the training sample space S_{train} is unbiased such that all classes have equal number of samples for training. However, it is seldom the case in practical scenarios. With more samples the expected class will get more chances of obtaining higher class sums in comparison to other classes that can skew the training in favor of a particular class. Moreover, any classification problem has inherent biases where some class is relatively easier to distinguish among others depending on its properties, input raw data preprocessing and the underlying ML algorithm. Threshold combats these biases by limiting the number of feedback as shown in Fig. 8.

Fig. 9 presents the number of feedback actions per class for 4 epochs of training a TM: $(D = MNIST, M = 10, N = 100, l = 1568, s = 5, \tau = 50,000)$ (see Table I for key term definitions), using MNIST dataset for $T = \{5, 10, 20, 40, 60, 100\}$. We can draw the following observations from the figure:

- The higher the T the more the number of feedback.

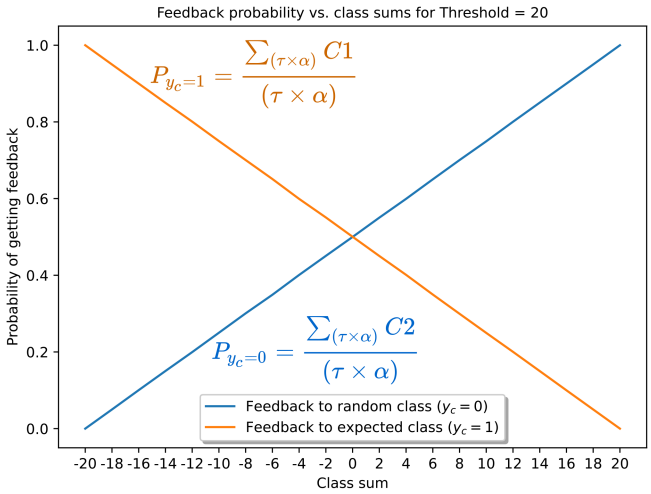


Fig. 8. This figure shows the mean probability of a clause getting feedback for a particular class sum. The mean is taken for $C1$ or $C2 = \texttt{TRUE}$ over $\tau \times \alpha$ (=1 million) total comparisons for each class sum indicated on the x-axis. τ is total number of training samples and we assume that each class gets same proportion (α) of feedback hits.

- Feedback per class varies considerably for small T while remains relatively uniform for high T.
- The training accuracy deteriorates with epoch for high $T = 40, 60, 100$.
- $T = 10$ outperforms $T = 5$ as can be seen from their accuracy in four epochs: epoch 1 - 91.23% versus 90.05%, epoch 2 - 92.81% versus 90.73%, epoch 3 - 92.97% versus 91.98% and epoch 4 - 93.25% versus 92.39%.

High T generates excess of feedback deteriorating the classification accuracy and more feedback increases the training time. It is unable to combat the aforementioned biases in the dataset. Similarly, a very small T fails to calibrate the TA profile to capture the important features. Fig. 9 shows that an optimum range of T exists for a specific datasets that will result in high accuracy.

B. Hyperparameter: s

The hyperparameter s is used to determine if the state transition (increment or decrement) will happen for an automaton in the Type I feedback, as shown in Fig. 4. Fig. 10 shows the Equations S1 and S2 again and plots the probability of S1 and S2 to be TRUE for $\beta = 1$ million checks for different values of s, each one being an independent trial. If S1 is TRUE then the state will decrement with the aim of making it exclude, if not already. If S2 is TRUE then the state will increment and more includes may appear for automata whose corresponding literal is 1. A high s will lead to more includes and vice-versa as shown by their probabilities in Fig. 10. Thus, the hyperparameter s determines the learning rate and capacity. The learning capacity relates to the maximum training accuracy at which the TM saturates and it will require an increase in s to achieve any better accuracy. The hyperparameter s plays an extremely crucial role in avoiding overfitting and achieving high accuracy while minimizing number of includes and forms the core of TA Re-profiling. We will continue to discuss hyperparameter s in detail in the upcoming sections. However, it should be noted that similar to T, there exists an optimum range of s for accurate classification of a dataset. The empirical evidence will be provided in the following sections.

TABLE II
AN EXCERPT FROM THE TABULATED OUTPUT OF TM ARCHITECTURE SEARCH PARADIGM FOR MNIST ARRANGED IN THE DESCENDING ORDER OF ACCURACY. THE DATA BELONGS TO FIG. 11

Clauses	Epochs	T	s	Includes	Accuracy↓
300	100	20	7.5	69808	96.26
300	100	20	10	124484	96.24
200	100	15	7.5	52356	96.09
200	100	10	10	41078	95.88
300	100	10	7.5	32034	95.74
200	50	10	7.5	18927	95.58
100	50	10	7.5	31247	94.98
100	50	15	5	30557	94.83
300	50	10	2.5	7752	89.81
200	50	15	2.5	4939	89.18

C. TM Architecture Search Paradigm

There does not exist a deterministic method to find TM architecture parameters suitable to classify any dataset with satisfactory accuracy and a certain number of includes. TM architecture parameters are determined empirically and mainly consist of the number of clauses, T and s and less often the number of epochs as it can be decided during runtime. As mentioned previously, there exists a range of TM architecture parameters that can produce an optimal result for a dataset. To identify the optimal solution, we propose a automated TM Architecture Search Paradigm (TMASP) approach where TM source code is integrated with data visualization tool from Weights & Biases [31]. It generates sweeps for each combination of architectural parameters specified, a subset of which can be visualized in Fig. 11 and tabulated in Table II. TMASP is an automated process that stores the raw state values of TA for each sweep, generates visualization, tabulate the results and can work in both online and offline modes. Multiple instances of TMASP can run simultaneously by running one architecture combination per core to speedup the exploration process. The source code can found at: `https://github.com/nclaes/tmasp`.

From Table II, we notice that $s = 7.5$ has consistently produced high accuracy compared to others while $s = 2.5$ is the worst performer. The number of includes is quite high for high $T = 15, 20$ for $s = 7.5$, $N = 300$ and $epochs = 100$. This reinforces the understanding developed in Section VI-A that high threshold increases the number of feedbacks, specifically the Type I feedback, resulting in more includes. Often more clauses may lead to more includes. The storage and computational requirements are directly proportional to the number of includes. Fewer includes will require smaller space to store and will consume less computation time during inference. It should be noted that using the REDRESS training procedure the number of includes can be reduced and the accuracy can be improved as will be discussed in Section VI-D, hence, the selection here determines the configuration at the start of training.

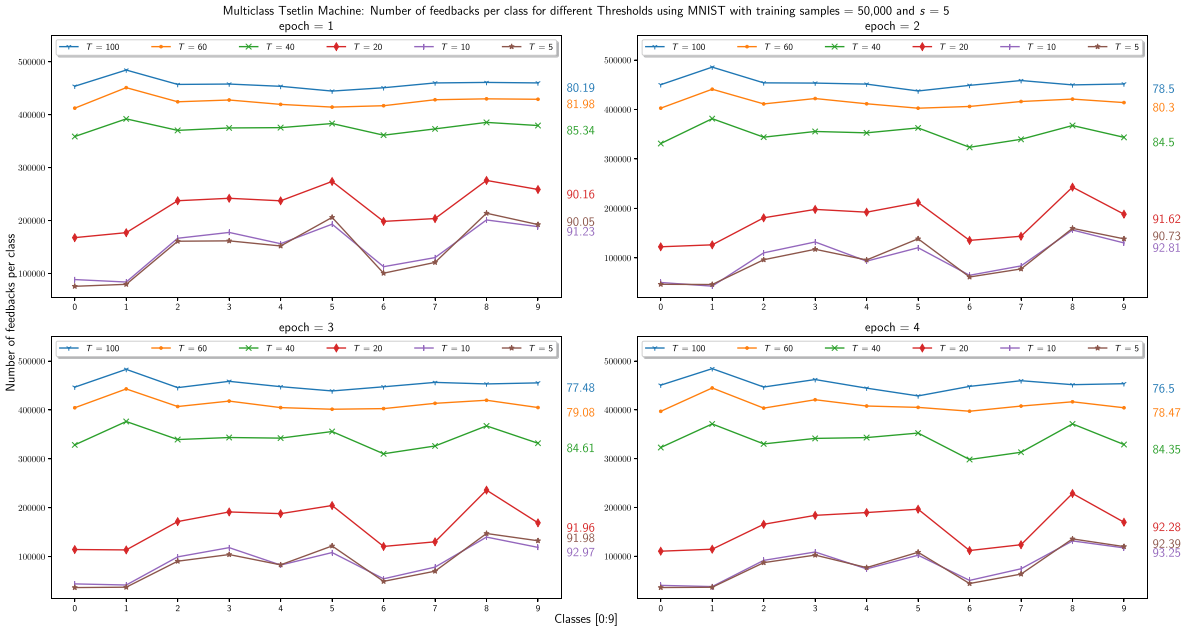


Fig. 9. These figures show the variations in number of feedback received by classes with varying $T = \{5, 10, 20, 40, 60, 100\}$ while training TM: ($D = MNIST$, $M = 10$, $N = 100$, $l = 1568$, $s = 5$, $\tau = 50,000$). For very small or large T, the accuracy deteriorates all through the 4 epochs, indicates an existence of optimum range of T values for a dataset. D- name of the dataset, M - the number of classes, N - the number of clauses per class, l - the number of literals(≡automata) per clause, s - hyperparameter, T - hyperparameter and τ - the number of samples).

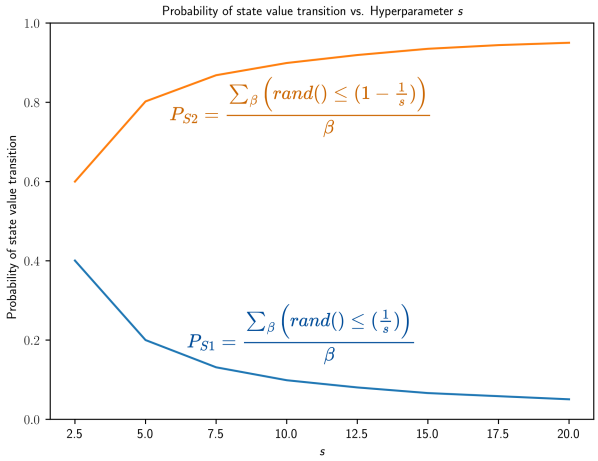


Fig. 10. This figure shows probability of Tsetlin automaton state transition with respect to different s values. The comparison operations with random numbers were performed $\beta = 1$ million times and its ratio with sum of TRUE outputs is plotted.

Using Table II, we can select the TM architecture parameters for MNIST that satisfy our requirements of includes and accuracy. Among other choices, we select the following parameter values: $N = 200$, $s = 7.5$, $T = 10$. Section VII elaborates on the parameters selected for other datasets in this paper while discussing the complete REDRESS flow.

D. Tsetlin Automata Re-Profiling

TA re-profiling is the procedure to disturb the Nash equilibrium of the automata [18] while retaining the include-exclude decisions, hence, the training accuracy from the last training epoch. It, basically, involves forced resetting of state to minimum possible value such that it retains the include-exclude characteristic. Fig. 12 presents examples of two automata where they retain include and exclude decision, however, their values are reset to $middle\_state + 1$ and 1, respectively. The standard training procedure initializes the TA with random values around the middle of the state value range as shown in Fig. 3. TA re-profiling enables us to initialize the TM with a previously trained model and gives TM an opportunity to re-calibrate the automata with user specified hyperparameters. Re-profiling TA saves us from starting the training process from scratch and enables TM to search for global optimal solution. It enables re-using a well trained model for fine tuning to increase the accuracy and decrease the model size.

E. Methodology

The proposed REDRESS training methodology is based on the hypothesis that by re-profiling the TA and adjusting the s hyperparameter, a similar or better accuracy may be achieved with fewer includes. We discuss the methodology here and empirically validate this hypothesis in Section VIII. Fig. 13. (a) presents the REDRESS training methodology using the TA re-profiling procedure. First, the TMASP is performed to determine optimal values for clauses and hyperparameters. In TMASP mentioned in Section VI-C, the TA is initialized randomly to includes and excludes following the standard training procedure presented in Section III-B. However, in our experiments we have found that when all TA are initialized as excludes and assigned the state value of 1 then the TM tends to achieve same accuracy with fewer includes albeit with more epochs of training. Similarly, the range of state values from $[1, 200]$ to $[1, 400]$ increases the number of

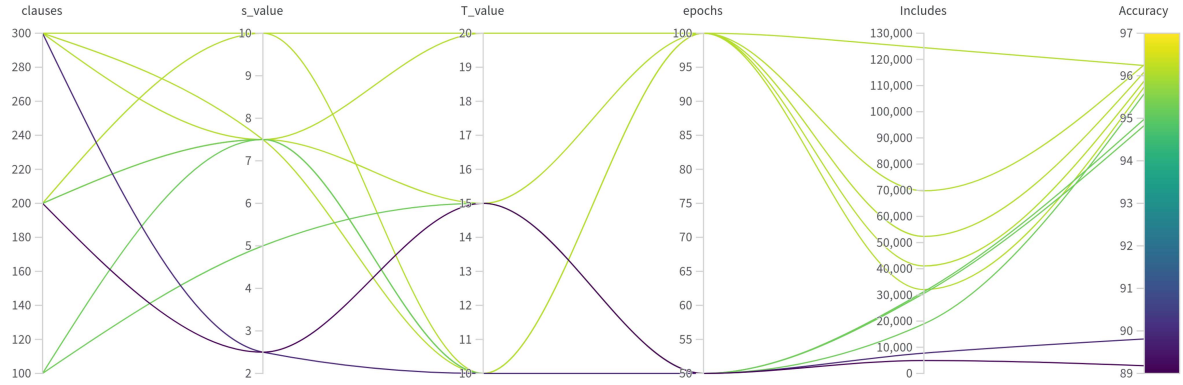


Fig. 11. Visualization of the sweeps of TM Architecture Search Paradigm (TMASP) for MNIST generated using ML visualization tools from Weights & Biases [31]. Only a subset of all the sweeps are shown for understanding. The TMASP was performed for all combinations of the following architecture parameter ranges: $N = [100, 200, 300]$, $T = [10, 15, 20]$, $s = [2.5, 5, 7.5, 10]$, $epochs = [50, 100]$. N - the number of clauses per class.

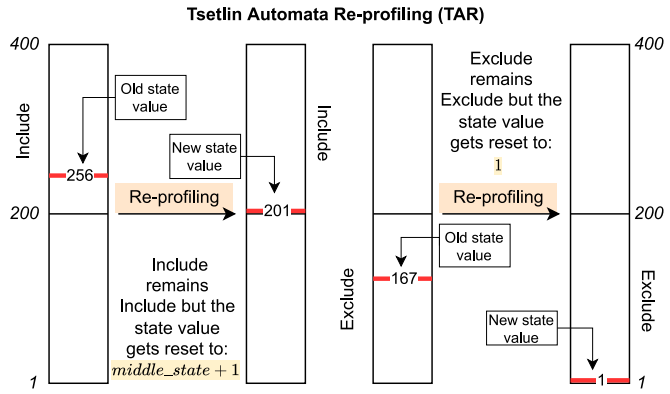


Fig. 12. Tsetlin Automata Re-profiling: The state value of each automaton is reset to minimum possible value such that it retains the include-exclude decisions.

automaton-level increments in the Type I feedback required to become an include. Therefore, any include requires double the feedback and may demonstrate high confidence and importance in accurate classification. Larger ranges of state value may be explored by interested readers. However, this exploration is beyond the scope of this paper.

TMASP, also, provides the raw TA file for the chosen architectural parameters to start the training process. The raw TA file can be used to initialize the TA after going through the TA re-profiling procedure or, at user's discretion, the first training cycle can start from scratch with all TA initialized as excludes with state value of 1. A training cycle includes the TA re-profiling based initialization of TA file obtained from the previous training cycle followed by the desired number of training epochs. In all training cycles the number of clauses and T are kept fixed, however, s can vary as per user-specification. The training procedure saves the raw TA file every time the training accuracy improves. If the accuracy and include profile, i.e., the number of includes, after the current epoch are found to be satisfactory then the training process can be terminated. Otherwise, the user has a choice of continuing training or start next training cycle that will require TA re-profiling based re-initialization

TABLE III
ENLISTING THE OPTIMUM TM MODEL PARAMETERS OBTAINED FROM TMASP RUNS FOR DIFFERENT DATASETS

Dataset	TM parameters
MNIST	$M = 10,\ N = 200,\ l = 1568,\ s = 7.5,\ T = 10$
FMNIST	$M = 10,\ N = 500,\ l = 1568,\ s = 7.5,\ T = 20$
KMNIST	$M = 10,\ N = 500,\ l = 1568,\ s = 10,\ T = 20$
CIFAR2	$M = 2,\ N = 1000,\ l = 2048,\ s = 7.5,\ T = 30$
KWS6	$M = 6,\ N = 300,\ l = 754,\ s = 7.5,\ T = 25$

of the TA. In Section VIII, we will see that the choice of s is crucial before starting the training cycle to achieve the desired model. Fig. 13(b) presents the overview of the entire REDRESS approach. All the datasets used in this paper for validation have gone through the same flow to obtain the optimized TM models.

VII. EXPERIMENTAL SETUP

We use five datasets for validation viz. MNIST [32], Fashion-MNIST [33] (FMNIST), Kuzushiji-MNIST [34] (KMNIST), CIFAR2 [35] and 6-keyword spotting [36] (KWS6). MNIST, FMNIST and KMNIST are well known 10-class benchmark containing 60 k training and 10 k testing datapoints composed of 28×28 pixel images ($\equiv 784$) features. MNIST, FMNIST and KMNIST are Booleanized using adaptive Gaussian thresholding procedure with window size 11 and threshold value 2. KWS6 is Booleanized form of six speech commands which include: {yes, no, up, down, left, right} [19]. CIFAR2 is a variant of CIFAR-10 with 2 classes of images viz. animals and vehicle, containing 50 k training and 10 k testing 32×32 pixel grayscale images with 1,024 features.

The same REDRESS training and inference methodology, shown in Fig. 13(b), is used with all the five datasets (or benchmarks). Before training, however, we initialize the TA as excludes with state value of 1 in all our experiments. The state value range used in our experiments is $[1, 400]$. Table III presents the optimum TM architecture configuration obtained from TMASP runs for different datasets. We compare REDRESS models with seven models of BNN with the configurations shown in Table IV. We have used the default settings to train the BNN as provided

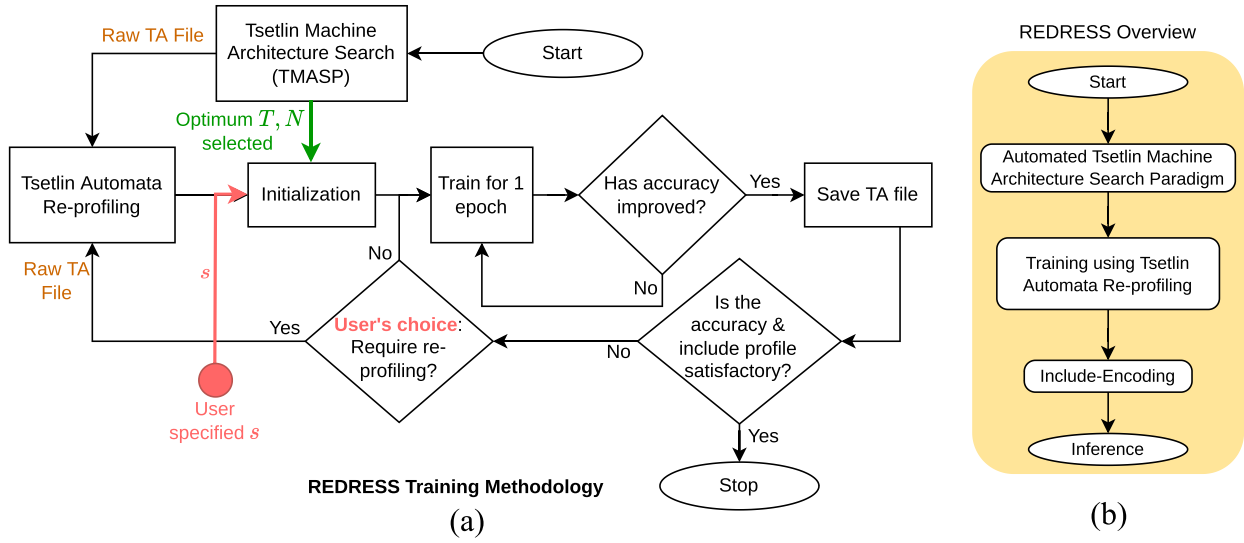


Fig. 13. (a) This figure presents the REDRESS training methodology combining various components together. It starts with TM architecture search followed with TA re-profiling based initialization and standard training routine for one epoch. Depending on the resulting accuracy the training continues to iterate. It saves raw TA file once the accuracy improves. The user can terminate and restart the training using the TA re-profiling based initialization with a change of hyperparameters in search of a better model. (b) The complete overview of the REDRESS training and inference methodology.

TABLE IV
ENLISTING THE BNN MODELS. ALL THE STATED MODELS WERE EVALUATED FOR EACH DATASET. L_i DENOTES THE NUMBER OF NEURONS IN THE i^{th} LAYER

Model	Description
FC-128	1 layer: L_1-128, fully-connected
FC-256	1 layer: L_1-256, fully-connected
FC-256-128	2 layers: L_1-256, L_2-128, fully-connected
FC-512-256	2 layers: L_1-512, L_2-256, fully-connected
FC-512-256-128	3 layers: L_1-512, L_2-256, L_3-128, fully-connected
Conv-1L	1-layer convolution
Conv-2L	2-layer convolution

by the authors of cited works. Typically BNNs already provide a frugal means of encoding weights using signed binary representations of weights in neurons. Modifying those settings and exploring DNNs with different precision has already been studied in [9]. We have validated the inference time, accuracy and energy of using the proposed REDRESS TM on STM32F746G-DISCO, running at 216 kHz, equipped with ARM Cortex-M7 core, 1 Mbytes of flash memory and 340 Kbytes of RAM. The power used by the micro-controller during inference is measured at source using Keysight N6705 C DC Power Analyzer [37]. The power and energy measurements presented in this paper include the entire micro-controller SoC board power. The source code of REDRESS approach can be found at: `https://github.com/nclaes/tm-redress`

VIII. RESULTS AND DISCUSSION

Fig. 14 presents the effect of T on the number of includes. TM parameters for each dataset are taken from Table III and for each $T = \{15, 20, 25, 30\}$ they are kept constant. For each dataset the training is performed for the same number of epochs. We can see that for all datasets the includes increase with T. For MNIST,

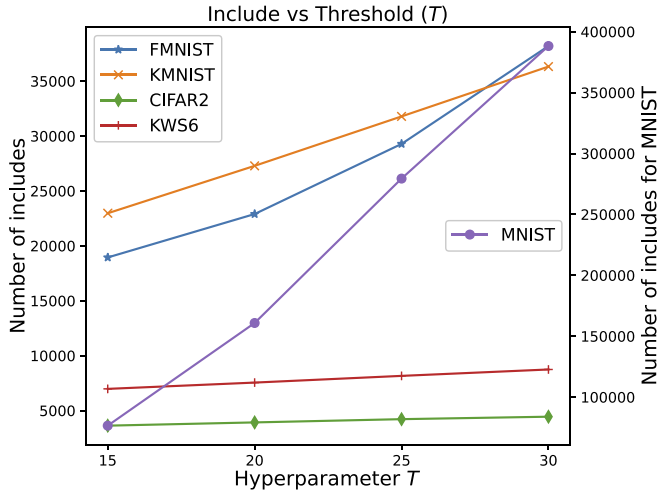


Fig. 14. The effect of threshold (T) on includes when all other parameters are kept constant. Other TM parameters are taken from Table III.

the number of includes are particularly larger compared to other dataset and hence, a different y-axis is used for representational purpose. This reinforces the choice of $T = 10$ for MNIST with the aim to reduce includes.

As mentioned in Section VI-B, the probability of state increment tends to 1 with increase in s, implying that includes will increase with s. This is, also, demonstrated in Fig. 15, which presents the effect of s on the number of includes. For all datasets the includes increase with s.

Fig. 16 presents the changes in the TA profile and accuracy when a TM model goes through cycles of TA re-profiling based REDRESS training procedure. The experiments have been performed for all five datasets where all parameters, taken from Table III, are kept constant except s. Two sets of experiments

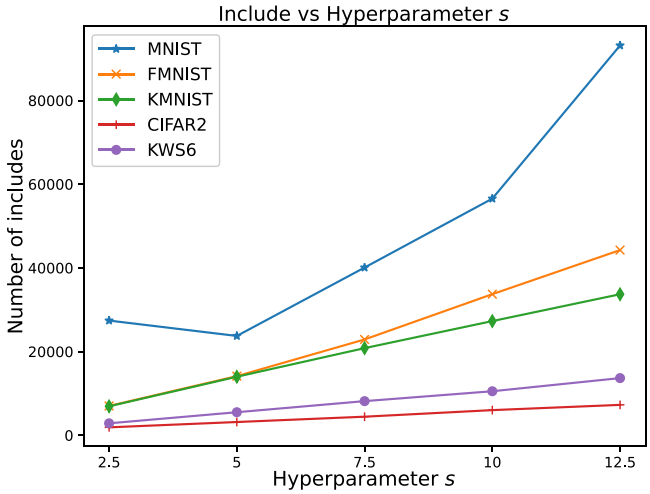


Fig. 15. The effect of hyperparameter s on includes when other parameters are kept constant. Other TM parameters are taken from Table III.

are performed for each dataset, one with decreasing and the other with increasing s, shown in the left and right columns in Fig. 16, respectively. In each REDRESS training cycle, the TM is trained for variable number of epochs, generally, less that 10 and the training starts from scratch by initializing all TA as excludes with state value $= 1$. The hyperparameter s is kept the same for three consecutive REDRESS training cycles as shown in Fig. 16. We exit the current training cycle once the training results in improved accuracy. The accuracy shown in the figure is test accuracy in the latest epoch. At the start of the next REDRESS training cycle the TA re-profiling is induced to initialize the TA. Thus, the TA file obtained from the previous training cycle is used in the next after re-profiling.

For MNIST the chosen $s = \{2.5, 5, 7.5, 10\}$. Within the three consecutive training cycles with the same s, we notice that better accuracy is achieved with similar or fewer number of includes. The number of includes dip when s is reduced from 10 to 7.5 and rises when it increases from 2.5 to 5. This corroborates our observation in Fig. 15. An important observation in the decreasing s case can be seen when s is reduced from 5 to 2.5, which leads to significant decrease in the number of includes and accuracy, both. This suggests that $s = 2.5$ is too small for this dataset. For other experiments, we avoid $s = 2.5$ as they all demonstrate the same trend.

For FMNIST, KMNIST, CIFAR2 and KWS6, $s \in [5, 15]$. For higher s values such as 12.5 and 15, we observe erratic behaviour in the case of decreasing s in FMNIST and KMNIST. This happens when TM is unable to improve the accuracy while at the current optimal solution and hence, the number of includes jump to find a better solution. However, once the accuracy improves this jump in includes can be mitigated at a later stage by decreasing the s as shown in Fig. 16. This demonstrates that higher s can help approach a global optima with satisfactory accuracy and then the TA re-profiling based REDRESS may be used with smaller s to minimize the number of includes. Fig. 16 shows the effectiveness of our approach in finding an optimum model with high accuracy and a minimum number of includes.

TABLE V
COMPARISON BETWEEN REDRESS TM AND BNN MODELS BASED ON MEMORY FOOTPRINT (MEM), ACCURACY (ACC), ENERGY (E) AND INFERENCE TIME OF TESTING DATASET. REDRESS OUTPERFORMS BNN MODELS EXCEPT FOR THE VALUES IN BLUE COLORED CELLS

	Models	Mem(kB)	Acc(%)	E(J)	Time(s)
	MNIST				
BNN models	FC-128	54.4	94.83	64.1	47.5
	FC-256	69.4	96.3	122.5	90.8
	FC-256-128	75.8	96.5	144.9	107.3
	FC-512-256	120	96.96	321.3	238
	FC-512-256-128	136.9	97.1	355.4	263.3
	Conv-1L	51.1	95.6	2476.7	1905.2
	Conv-2L	50.4	96.7	18411.7	14162.8
	REDRESS	51.7	96.48	6.56	4.86
	FMNIST				
BNN models	FC-128	55.5	85.4	64.1	47.5
	FC-256	70.4	85.8	127.7	94.6
	FC-256-128	76.9	86.2	150.3	111.4
	FC-512-256	121	87.2	332.8	246.6
	FC-512-256-128	127.5	87.5	355.4	263.3
	Conv-1L	52.1	85.9	2477	1905.4
	Conv-2L	51.4	86.7	18281.5	14062.7
	REDRESS	65.76	87.67	13.01	9.64
	KMNIST				
BNN models	FC-128	56.83	78.3	56.1	41.6
	FC-256	71.7	81.5	111.7	82.7
	FC-256-128	78.2	83.2	132.1	97.9
	FC-512-256	122.4	86	300	222.3
	FC-512-256-128	128.8	86.4	322.8	239.1
	Conv-1L	47	75.2	4455.3	3427.2
	Conv-2L	46.2	76.3	14053	10809.9
	REDRESS	75.5	88.6	14.1	10.4
	CIFAR2				
BNN models	FC-128	59	79.2	80.4	59.6
	FC-256	77.5	79.9	160.6	119
	FC-256-128	84.1	81	183.8	136.2
	FC-512-256	135.6	80.8	400.2	296.4
	FC-512-256-128	142.2	80.9	423.3	313.6
	Conv-1L	45.9	85.4	3264.6	2511.3
	Conv-2L	46.2	86	24777.7	19059.8
	REDRESS	28.4	84.8	4.5	3.3
	KWS6				
BNN models	FC-128	49.1	81.7	11.5	8.54
	FC-256	57.7	82.6	23.3	17.2
	FC-256-128	64.2	85.4	30.2	22.4
	FC-512-256	95.8	84	70.7	52.3
	FC-512-256-128	102.3	81.7	77.4	57.4
	Conv-1L	46	67.4	364.6	280.5
	Conv-2L	46	71.2	2525.2	1942.5
	REDRESS	33	87.1	1.39	1.03

The three colums of subfigures, in Fig. 17, each representing a training cycle, show the trends of feedback and includes before and after the junction at which TA re-profiling is applied twice. It shows the number of includes in the y-axis (left), feedback on the secondary y-axis (right), and accuracy (in the plot) at the TA re-profiling junctions. All TM parameters are kept constant in all the three training cycles with each cycle training for 10 epochs. Only MNIST test case is provided here, the subfigures for CIFAR2, KWS6, FMNIST and KMNIST can be found in Fig. 1 of the supplementary material, (available online). We can observe that the number of includes dip along with accuracy immediately after the TA re-profiling and gradually rises producing better training accuracy with fewer includes, in the

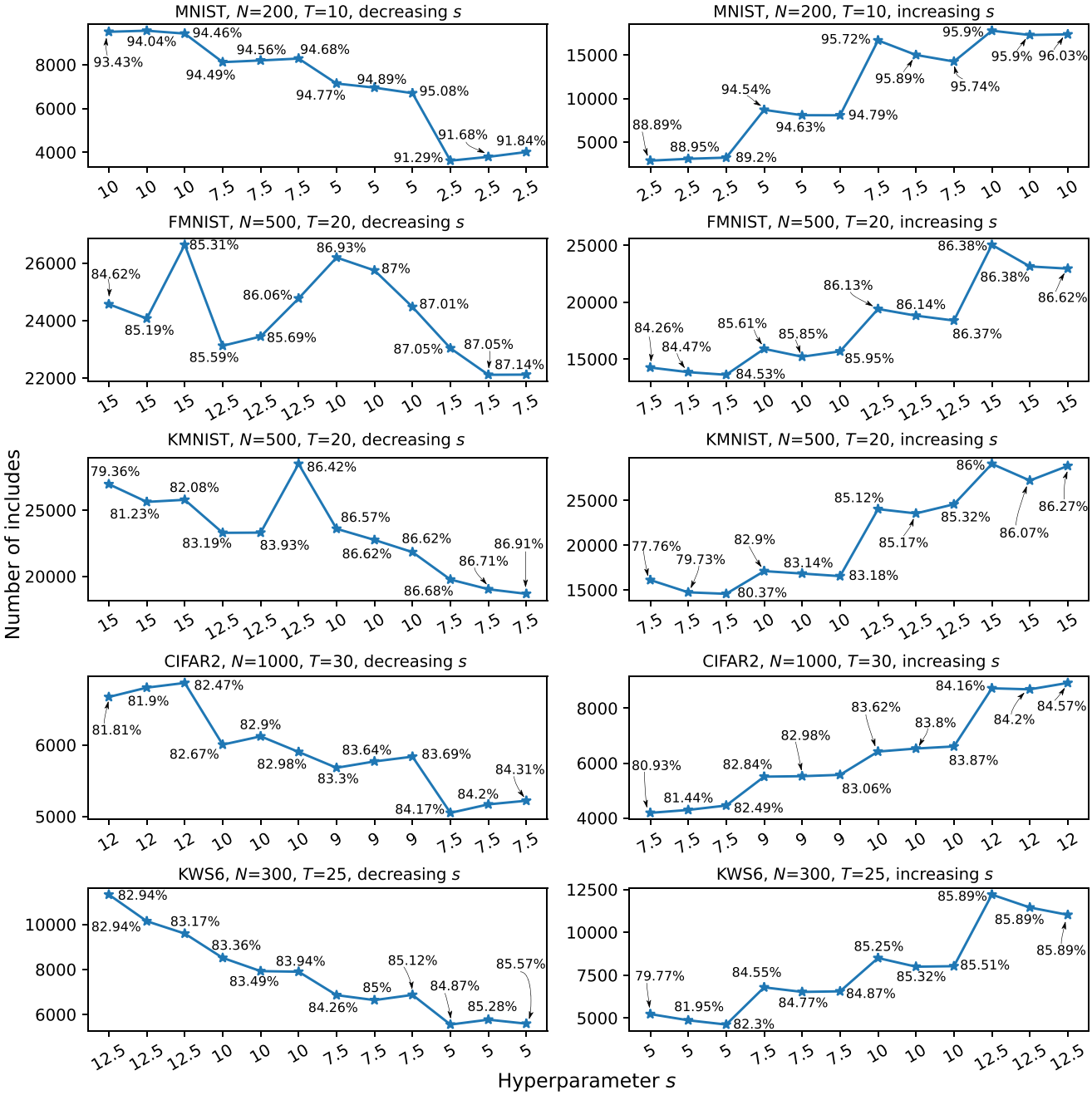


Fig. 16. These figures demonstrates the advantage of using TA re-profiling through variations in the number of includes and accuracy vis-à-vis s. TM parameters, taken from Table III, are kept constant through the runs except s that is changed after every three consecutive training cycles. The TA file obtained from the previous training cycle is used to initialize the TA in the next training cycle after re-profiling. A total of 12 training cycles, in the decreasing and increasing order of s are shown for each dataset. It demonstrates how optimal model can be obtained through REDRESS training approach with changes in s. High s can help identify global optimal solution and decreasing s in the following training cycles can reduce the number of includes and fine-tune the model.

case of CIFAR2 and KWS6, or similar number of includes as in the case of MNIST, FMNIST and KMNIST. This demonstrates that TA re-profiling is a computationally minimal procedure to fine-tune the model to improve accuracy and reduce memory footprint.

Table V presents the inference results obtained from the micro-controller for the testing dataset of MNIST, FMNIST, KMNIST, CIFAR2 and KWS6. It compares seven different BNN models against TM models obtained from REDRESS. All BNN models were trained for 100 epochs as per the parameters mentioned in Table IV. The parameters used in TM models are mentioned in Table III. The memory footprint displayed in Table V consists of the model size together with the size of the source code. The model size in kilo bytes (kB) includes coefficients in the case of BNN and encoded includes in the case TM. The accuracy is measured over the testing dataset. The stabilized power measured during Conv-1 L and Conv-2 L inference was found to be 1.3 W while in the case of all other BNN and TM

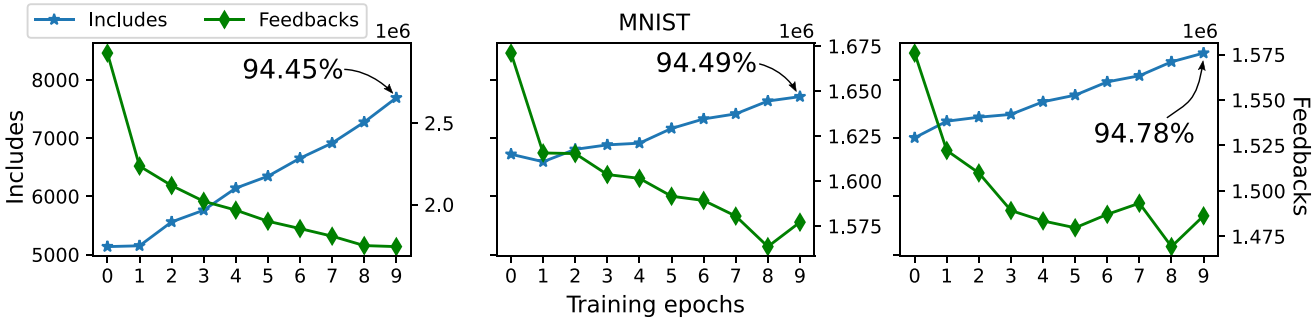


Fig. 17. The effect of TA re-profiling while training MNIST if all parameters are kept constant and trained for the same number of epochs. The training accuracy improves at the end of every training cycle with fewer or similar number of includes. It demonstrates how REDRESS Training procedure can help fine-tuning a TM model.

TABLE VI
MEMORY FOOTPRINT COMPARISON OF COMPRESSED MODEL OBTAINED FROM RLE [38] AND REDRESS VERSUS UNCOMPRESSED TA IN KILO BYTES (KB)

Dataset	TA model	RLE	REDRESS
MNIST	3062.5	2853.8	34.1
FMNIST	7656.3	7156.8	48.2
KMNIST	7656.3	6919.7	58
CIFAR2	4000	3865	10.9
KWS6	1325.4	1248.8	15.4

models, it was measured to be 1.35 W. The difference can be attributed to larger model sizes leading to more data movement. The model size, excluding the source code size, for Conv-1 L and Conv-2 L range between 4-10 kB while the minimum model size for other models is >16 kB. Energy is the product of inference time with power shown in the unit of Joule. The blue colored cells in Table V demarcate those instances where the BNN models outperform TM. As we can see, in most cases TM outperforms BNN, especially, in the amount of energy and computation time by orders of magnitude. The difference in accuracy for MNIST and CIFAR2 between best performing BNN model compared to TM is $<1.2\%$. TM results in notably better test accuracies in the case of FMNIST, KMNIST and KWS6. Overall, TM models demonstrates significantly better overall performance compared to the state-of-the-art BNN models. Table VI presents a comparison of the memory footprint of TM model upon compression using the proposed Include-Encoding method. It shows that the REDRESS model occupies 86-367$\times$ less memory on the micro-controller compared with the entire TM model, if 1 B is used to store an automaton.

Run length encoding (RLE) based compression approach is presented in [38] to use TM in edge inference scenario. Table VI presents a comparison of memory footprints of compressed TA obtained using RLE and REDRESS relative to the original uncompressed TA in kilo bytes. We can see that REDRESS considerably outperforms RLE producing 81-354$\times$ more compression. It is possible that the TM models have many alternating include-exclude bits that may be responsible for relatively poor performance of RLE. The compression is performed after training is complete and the raw TA file with include-exclude decisions is available. Here, the same TA file goes through RLE and REDRESS compression methods and both techniques are lossless, hence, both have the same accuracy.

IX. CONCLUSION

This article proposes how to significantly improve the sparseness of Tsetlin Machines (TMs), a recent and increasingly popular machine learning algorithm. After introducing the vanilla TM training and inference methodology to a wider audience, we present an off- and online TM Architecture Search Paradigm (TMASP) named REDRESS for exploring model architectures and hyper-parameters on application-specific datasets. The proposed training methodology drastically increases the sparseness of TM models. We achieve this by re-profiling the underlying Tsetlin Automata that drives the learning, helping the TM search for a better solution with fewer includes (more concise pattern representation). The sparse models then go through lossless compression using the so-called include-encoding, which distills the bare minimum information required to perform inference without loss in accuracy. Using the compressed-domain inference algorithm, the REDRESS TM outperforms seven BNN models on inference time, energy, and memory footprint across five benchmarks viz. MNIST, FMNIST, KMNIST, CIFAR2 and KWS6. REDRESS further simplifies the vanilla TM clause compute by skipping unnecessary excludes, thereby classifying with fewer bitwise operations. In conclusion, the REDRESS approach demonstrates that highly sparse TMs yield improved accuracy while boosting computation speed and energy efficiency.

REFERENCES

[1] G. Litjens et al., "A survey on deep learning in medical image analysis," 2017, *arXiv:1702.05747*.
[2] W. Y. Yi, K. M. Lo, T. Mak, K. S. Leung, Y. Leung, and M. L. Meng, "A survey of wireless sensor network based air pollution monitoring systems," *Sensors*, vol. 15, no. 12, pp. 31392–31427, 2015.
[3] V. S. Marco, B. Taylor, Z. Wang, and Y. Elkhatib, "Optimizing deep learning inference on embedded systems through adaptive model selection," 2019, *arXiv:1911.04946*.
[4] Y. Kang et al., "Neurosurgeon: Collaborative intelligence between the cloud and mobile edge," *SIGARCH Comput. Archit. News*, vol. 45, no. 1, pp. 615–629, Apr. 2017.

[5] G. Gobieski, N. Beckmann, and B. Lucia, "Intelligence beyond the edge: Inference on intermittent embedded systems," in *Proc. 24th Int. Conf. Architectural Support Program. Lang. Operating Syst.*, 2019, pp. 199–213.
[6] M. Nardello, H. Desai, D. Brunelli, and B. Lucia, "Camaroptera: A batteryless long-range remote visual sensing system," in *Proc. 7th Int. Workshop Energy Harvesting Energy-Neutral Sens. Syst.*, 2019, pp. 8–14.
[7] A. Canziani, A. Paszke, and E. Culurciello, "An analysis of deep neural network models for practical applications," 2016, *arXiv:1605.07678*.
[8] B. Islam and S. Nirjon, "Zygarde: Time-sensitive on-device deep inference and adaptation on intermittently-powered systems," *Proc. ACM Interact. Mobile Wearable Ubiquitous Technol.*, vol. 4, no. 3, Sep. 2020, Art. no. 82.
[9] B. McDanel, S. Teerapittayanon, and H. Kung, "Embedded binarized neural networks," in *Proc. Int. Conf. Embedded Wireless Syst. Netw.*, USA: Junction Publishing, 2017, pp. 168–173.
[10] P. C. Knag et al., "A 617 TOPS/W all digital binary neural network accelerator in 10 nm FinFET CMOS," in *Proc. IEEE Symp. VLSI Circuits*, 2020, pp. 1–2.
[11] M. Scherer, A. Di Mauro, G. Rutishauser, T. Fischer, and L. Benini, "A 1036 TPp/s/W, 12.2 mW, 2.72 μJ/inference all digital TNN accelerator in 22 nm FDX technology for TinyML applications," in *Proc. IEEE Symp. Low-Power High-Speed Chips*, 2022, pp. 1–3.
[12] Z. Liu, M. Sun, T. Zhou, G. Huang, and T. Darrell, "Rethinking the value of network pruning," 2018, *arXiv:1810.05270*.
[13] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding," 2015, *arXiv:1510.00149*.
[14] B. Jacob et al., "Quantization and training of neural networks for efficient integer-arithmetic-only inference," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2018, pp. 2704–2713.
[15] Y.-D. Kim, E. Park, S. Yoo, T. Choi, L. Yang, and D. Shin, "Compression of deep convolutional neural networks for fast and low power mobile applications," 2015, *arXiv:1511.06530*.
[16] C. Rudin, "Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead," *Nature Mach. Intell.*, vol. 1, no. 5, pp. 206–215, 2019.
[17] A. Rafiev et al., "Visualization of machine learning dynamics in tsetlin machines," in *Proc. 1st Int. Symp. Tsetlin Mach.*, 2022, pp. 81–88.
[18] O.-C. Granmo, "The Tsetlin Machine–A game theoretic bandit driven approach to optimal pattern recognition with propositional logic," 2018, *arXiv:1804.01508*.
[19] J. Lei et al., "Low-power audio keyword spotting using tsetlin machines," *J. Low Power Electron. Appl.*, vol. 11, no. 2, 2021, Art. no. 18.
[20] T. Rahman, S. Maheshwari, R. Shafik, A. Yakovlev, and S. Das, "MILEAGE: An automated optimal clause search paradigm for tsetlin machines," in *Proc. 1st Int. Symp. Tsetlin Mach.*, 2022, pp. 49–52.
[21] J. Sharma, R. K. Yadav, O. Granmo, and L. Jiao, "Drop clause: Enhancing performance, interpretability and robustness of the tsetlin machine," in *Proc. AAAI Conf. Artif. Intell.*, 2023. [Online]. Available: https://aaai.org/aaai-publications/aaai-conference-proceedings/
[22] M. Courbariaux and Y. Bengio, "BinaryNet: Training deep neural networks with weights and activations constrained to 1 or -1," 2016, *arXiv:1602.02830*.
[23] J. Lei, A. Wheeldon, R. Shafik, A. Yakovlev, and O.-C. Granmo, "From arithmetic to logic based AI: A comparative analysis of neural networks and tsetlin machine," in *Proc. IEEE 27th Int. Conf. Electron. Circuits Syst.*, 2020, pp. 1–4.
[24] Y. Bengio, N. Léonard, and A. Courville, "Estimating or propagating gradients through stochastic neurons for conditional computation," 2013, *arXiv:1308.3432*.
[25] X. Zhang, L. Jiao, O. Granmo, and M. Goodwin, "On the convergence of tsetlin machines for the IDENTITY- and NOT operators," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 44, no. 10, pp. 6345–6359, Oct. 2022.
[26] L. Jiao, X. Zhang, O. Granmo, and K. Abeyrathna, "On the convergence of tsetlin machines for the XOR operator," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 45, no. 5, pp. 6072–6085, May 2023.
[27] R. K. Yadav, L. Jiao, O. C. Granmo, and M. Goodwin, "Human-level interpretable learning for aspect-based sentiment analysis," in *Proc. 35th AAAI Conf. Artif. Intell.*, 2021, pp. 14203–14212.
[28] R. K. Yadav, L. Jiao, O. C. Granmo, and M. Goodwin, "Robust interpretable text classification against spurious correlations using AND-rules with negation," in *Proc. Int. Joint Conf. Artif. Intell.*, 2022, pp. 4439–4446.
[29] K. D. Abeyrathna et al., "Massively parallel and asynchronous tsetlin machine architecture supporting almost constant-time scaling," in *Proc. 38th Int. Conf. Mach. Learn.*, PMLR, 2021, pp. 10–20.
[30] A. Abouzeid, O.-C. Granmo, C. Webersik, and M. Goodwin, "Socially fair mitigation of misinformation on social networks via constraint stochastic optimization," in *Proc. AAAI Conf. Artif. Intell.*, 2022, pp. 11801–11809.
[31] L. Biewald, "Experiment tracking with weights and biases," 2020. [Online]. Available: https://www.wandb.com/
[32] L. Deng, "The MNIST database of handwritten digit images for machine learning research," *IEEE Signal Process. Mag.*, vol. 29, no. 6, pp. 141–142, Nov. 2012.
[33] H. Xiao, K. Rasul, and R. Vollgraf, "Fashion-MNIST: A novel image dataset for benchmarking machine learning algorithms," 2017, *arXiv:1708.07747*.
[34] T. Clanuwat, M. Bober-Irizar, A. Kitamoto, A. Lamb, K. Yamamoto, and D. Ha, "Deep learning for classical japanese literature," 2018, *arXiv:1812.01718*.
[35] A. Krizhevsky et al., "Learning multiple layers of features from tiny images," Toronto, ON, Canada, 2009.
[36] P. Warden, "Speech commands: A dataset for limited-vocabulary speech recognition," 2018, *arXiv:1804.03209*.
[37] N6705c DC power analyzer, 2022. [Online]. Available: https://www.keysight.com/gb/en/product/N6705C/dc-power-analyzer-modular-600-w-4-slots.html
[38] A. Bakar, T. Rahman, A. Montanari, J. Lei, R. Shafik, and F. Kawsar, "Logic-based intelligence for batteryless sensors," in *Proc. 23rd Annu. Int. Workshop Mobile Comput. Syst. Appl.*, New York, NY, USA, 2022, pp. 22–28.

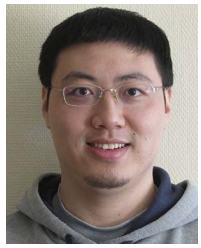
Sidharth Maheshwari received the BTech degree from IIT Guwahati, India, in 2013, and the PhD degree from Newcastle University, in 2021. He is a research associate with the Microsystems Group, Newcastle University, U.K. He is working on embedded Tsetlin Machine (TM) implementations. He is developing new TM implementations to reduce memory footprint, computation time, and improve accuracy. He is, also, developing an affordable and portable embedded genome analysis device for monitoring and surveillance of anti-microbial resistance.

Tousif Rahman received the MEng (1st class Hons.) degree from Newcastle University, in 2020. He is currently working toward the 2nd year PhD degree with the Microsystems Group, Newcastle University supervised by Dr Rishad Shafik and professor Alex Yakovlev. His research focuses on the transition of machine learning algorithms to energy efficient hardware.

Rishad Shafik (Senior Member, IEEE) received the BSc (with distinction) degree from IUT, Bangladesh, in 2001, and the MSc (with distinction) and PhD degrees from Southampton, in 2005, and 2010. He is a reader in electronic systems within the School of Engineering, Newcastle University, U.K. He is one of the editors of the Springer USA book "Energy-efficient Fault-tolerant Systems". He is also author/co-author of more than 120 IEEE/ACM peer reviewed articles, with several best paper nominations/awards. He recently chaired multiple international conferences/symposiums, including DFT2017, UKCAS2020, ISTM2022; guest edited a special theme issue in Royal Society Philosophical Transactions A. His research interests include hardware/software co-design for energy-efficiency and autonomy.

Alex Yakovlev (Fellow, IEEE) received the PhD degree from St. Petersburg Electrical Engineering Institute, in 1982. He is a professor of computer systems design, who founded and leads the Microsystems Research Group, and co-founded the Asynchronous Systems Laboratory, Newcastle University. He was awarded an EPSRC Dream Fellowship, in 2011–2013. He has published 8 edited and co-authored monographs and more than 400 papers in IEEE/ACM journals and conferences, in the area of concurrent and asynchronous circuits and systems, with several best paper awards and nominations. He has chaired organizational committees of major international conferences. He has been Principal investigator on more than 30 research grants and supervised more than 60 PhD students. He is a fellow of Royal Academy of Engineering in the United Kingdom.

Ashur Rafiev received the MEng degree in computing science from Kyrgyzstan, in 2005, and the PhD degree in EE from Newcastle University, in 2011. Previously worked in the Human-Computer Interaction and the Security and Software Reliability research groups with the School of Computing, Newcastle University. At the moment, he works as a research associate with the School of Engineering, Newcastle University, with the research interest in many-core systems including power modelling, simulation, visualisation, and applications for distributed computing. His recent work is focused on visualisation and optimisation of Tsetlin Machines and machine learning.



Lei Jiao (Senior Member, IEEE) received the BE degree in telecommunications engineering from Hunan University, Changsha, China, in 2005, the ME degree in communication and information system from Shandong University, Jinan, China, in 2008, and the PhD degree in information and communication technology from the University of Agder (UiA), Norway, in 2012. He is currently working as an Associate Professor with the Department of Information and Communication Technology, UiA. His research interests include reinforcement learning, Tsetlin machine, resource allocation and performance evaluation for communication and energy systems.

Ole-Christoffer Granmo received the master's and PhD degrees both from the University of Oslo, Norway, in 1999 and 2004, respectively, and created the Tsetlin machine, in 2018. He is the founding director of the Centre for Artificial Intelligence Research (CAIR), University of Agder, Norway. He has authored more than 150 refereed papers with six best paper awards within machine learning, encompassing learning automata, bandit algorithms, Tsetlin machines, Bayesian reasoning, reinforcement learning, and computational linguistics. He has further coordinated more than 7 research projects and graduated more than 55 master- and eight PhD students. He is also a co-founder of the Norwegian Artificial Intelligence Consortium (NORA). Apart from his academic endeavours, he co-founded the company Anzyz Technologies AS.