Logic-based Intelligence for Batteryless Sensors

Abu Bakar[†*°] Tousif Rahman[§*] Alessandro Montanari[‡] Jie Lei[◇¶] Rishad Shafik[§] Fahim Kawsar[‡]
[†]Northwestern University [§]Newcastle University [‡]Nokia Bell Labs [◇]Polytechnical University of Valencia
[*]Indicates equal contribution. [°]Work done while author was interning at Nokia Bell Labs.
[¶]Work done while author was at Newcastle University.

ABSTRACT

The emergence of embedded machine learning has enabled the migration of intelligence from the cloud to the edge and to the sensors. To explore the practicalities of wide-spread deployments of these intelligent sensors, we look beyond traditional arithmetic-based neural networks (NNs) to the logic-based learning algorithm called the Tsetlin Machine (TM). TMs have not yet been implemented and explored on general purpose microcontrollers especially that are intermittently powered. In this paper, we argue that their simple architecture makes them a promising candidate for batteryless ML systems. However, in their current form, they are not suitable to be deployed on resource-constrained sensors because of the substantial memory footprint of trained models. To tackle this issue, we propose a lossless compression scheme based on run-length encoding and evaluate against standard TMs for vision and acoustic workloads. We show that our encoding can compress the model by up to 99% without accuracy loss. This translates into lower memory footprint and better energy efficiency (up to 4.9×) compared to the original Tsetlin Machine algorithm, and provides promising trade offs when compared against binary neural networks.

CCS CONCEPTS

• Software and its engineering → Embedded software; **• Computing methodologies → Neural networks**; **• Hardware → Analysis and design of emerging devices and systems**.

KEYWORDS

Tsetlin Machines, Neural Networks, Energy Efficiency, Intermittent Computing, Battery-free.

ACM Reference Format:
Abu Bakar, Tousif Rahman, Alessandro Montanari, Jie Lei, Rishad Shafik, Fahim Kawsar. 2022. Logic-based Intelligence for Batteryless Sensors. In *The 23rd International Workshop on Mobile Computing Systemsand Applications (HotMobile '22), March 9–10, 2022, Tempe, AZ, USA.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3508396.3512870

1 INTRODUCTION

Sensing-based computation is increasingly becoming pervasive in our lives. With the vision that trillions of devices will be deployed by 2035 [21], there is a growing need for making them battery-free [9]. Batteryless sensors enable autonomous operation by harvesting ambient energy from solar, kinetic, radio frequency or other sources, with the ambition to make edge applications maintenance-free and long-lived for a sustainable future. However, these devices are impractical without the pairing of an ML model which can compute meaningful results directly on the sensor, eliminating the need for high-energy data transmission to the cloud [7].

Traditional machine learning applications rely on deep neural networks (DNNs) [7]. DNNs can provide good accuracy but at a high inference latency and model complexity. The critical challenge is that energy is scarce and intermittent, as such the device can suffer one or more power failures during a single inference. To retain program state in these conditions, the volatile program state must be stored into non-volatile memory before a power failure and restored when the sensor comes back online. This has a significant energy cost and delays the execution of inference tasks [16, 24].

Apart from inadequate energy and compute resources, memory management of DNNs is the main factor contributing to higher inference latency [7]. This cannot be easily avoided as memory containing intermediate results between two layers may be large and must be handled correctly to avoid memory inconsistencies [7, 24]. In this paper, we address the questions: *are there any alternatives to DNNs offering comparable accuracy but with lower latency and resource utilisation, and are they suitable for batteryless systems?*

The Tsetlin Machine (TM) is an emerging machine learning algorithm utilizing the principles of learning automata and game theory [8]. The TM's inference routine uses propositional logic as opposed to arithmetic operations, which makes it a less computationally intensive and energy frugal alternative to traditional artificial neural networks (§ 2). The computational simplicity of TMs results in two main benefits for intermittent systems. First, it makes writing task-based applications, usually a daunting exercise for intermittent systems developers [16, 24], significantly simpler, speeding up the development and in turn deployment of intelligent applications. Second, since the computation of each class is independent from one another (contrary to the case with neural networks), no data has to be transferred in-between different computational units back-and-forth. This means that if the execution is interrupted by a power failure, there is only a small portion of the memory that needs to be written to the non-volatile memory and restored after power up. Whereas in DNNs, each layer is typically computationally heavy and the memory required to store the intermediate result and pass it to the next layer is significant. This increases memory requirements adding latency overheads to intermittently powered applications. Figure 1 highlights the benefit of TMs over DNNs by showing the memory that would need to be stored and restored at each power failure.

The memory footprint of TMs is not significant for small problems [8], but does not scale easily on resource-constrained microcontrollers (MCUs). For example an MNIST classifier achieving

HotMobile '22, March9–10, 2022, Tempe, AZ, USA

ACM ISBN 978-1-4503-9218-1/22/03…$15.00
https://doi.org/10.1145/3508396.3512870

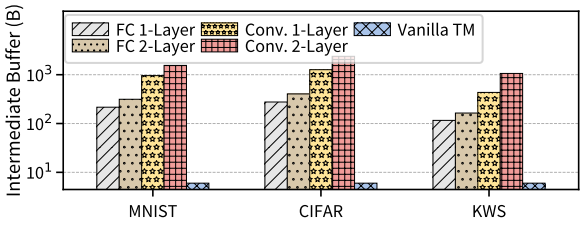

Fig. 1: Memory overhead of fully-connected and convolutional BNNs and TMs for different datasets. Vanilla TM refers to the standard Tsetlin Machine algorithm as presented in [8].

90% accuracy requires 306KB of memory, exceeding the capacity for a MSP430FR5xxx[1], a standard series of MCUs for running intermittently-powered applications.

In this paper, we explore the characteristics of Tsetlin Machines in a resource-constrained environment with particular focus on intermittent computing, and propose a compression scheme making use of run-length encoding (RLE) to compress the TM models by up to 99%, resulting in lower latency and memory footprint without loss of accuracy.

We evaluate our intermittence-aware compressed TM implementation on vision and acoustic datasets [13, 22] showing significant memory and latency improvements over standard TM implementation - e.g. 98.68% compression with 4.9× speedup for our Key Word Spotting (KWS) dataset. Additionally, through a comparison with state-of-the art binary neural networks we uncover interesting trade offs that could be exploited for the future use of TMs as the intelligence component on intermittently-powered devices.

In addition to advocating TMs as a good fit for battery-free inference, we make the following contributions:

- An initial exploration of a new direction for memory and energy-efficient batteryless TMs.
- Early insights into a first method for compressing TM models targeted for intermittently-powered systems.
- A detailed comparison of compressed TMs with state-of-the-art embedded binary neural networks (BNNs).

We begin by providing a background on intermittent computing and Tsetlin Machines, and then delve into our compression approach and intermittence-aware TM implementation. We continue with the evaluation of our approach against the standard TM implementation and binary neural networks, both with continuous and intermittent power, before offering concluding remarks

2 BACKGROUND

2.1 Intermittent Computing

The vision of ubiquitous computing will require sensors to harvest ambient energy to ensure long lifetime and low maintenance cost. The sporadic availability of harvested energy makes the continuous execution of the programs impossible [15]. Devices accumulate energy in a capacitor and run programs only when the level of charge is sufficient to keep the device operating. When the energy is depleted the device switches off until the next charging cycle is complete, resulting in very short uptime periods (e.g., few milliseconds) and potentially long downtimes (e.g., hours). This hampers the use of conventional programming models, designed for continuously-powered devices, to run correctly on batteryless sensors as the memory consistency and forward progress is compromised due to frequent power failures.

Intermittent computing models preserve forward progress and ensure memory consistency by inserting checkpoints throughout the program code. When a power failure approaches, the content of volatile memory is stored into a non-volatile memory, and the execution is restored from the same point when the device reboots after a power outage. Several models exists; from the ones which operate at compile-time [12], to more dynamic ones which react to current harvesting conditions [3], to approaches that rely on appropriate configuration from the developer [24]. All these techniques lead to significant memory and latency overhead due to continuous store/restore operations.

Recently, several works are focusing on bringing complex recognition tasks, based on deep neural networks (DNN), on intermittent systems with the objective of enabling useful applications (e.g., wild-life conservation [5], healthcare [4] and building management [1]). Understating sensor data through local inference running intermittently is crucial for these applications. It enables the creation of smaller and more cost-effective devices (i.e., without a battery) and it liberates them from expensive maintenance (e.g., periodic battery replacement) when deployed in remote or difficult to reach locations. Once we remove batteries from sensing devices, however, streaming raw data for offline processing becomes unfeasible since the cost of transmitting radio packets would render the device ineffective in doing any other useful work. For example, Gobieski et al. [7] found that the energy required to transmit an MNIST [6] image is approximately 360× greater than the energy needed to perform local inference intermittently using a neural network. Similarly, Nardello et al. [20] reported approximately 75x greater power draw when sending LoRa packets compared to local CNN inference. Despite the benefit compared to radio transmissions, these DNN workloads come with a significant burden in terms of resources they require, i.e., memory and energy, especially in situations where compute is happening on harvested energy. Hence, initial efforts are considering the dynamic adaptation of such workloads [2, 10, 18, 19].

In this paper we examine Tsetlin Machines as an alternative recognition algorithm which has the potential to reduce these requirements thanks to its logic-based formulation.

2.2 Tsetlin Machines

The Tsetlin Machine is an ML algorithm that uses a learning automata called *Tsetlin Automata* to form logic propositions with booleanized input features and their complements. These logic propositions are used to determine the classification. The simplistic logic-over-arithmetic approach of the TM opens pathways for more energy efficient embedded and hardware implementations [23].

Booleanization. One fundamental difference separating TMs from traditional NNs is the need for *booleanization* of the input data. This differs from binarization as there is no longer any notion of place value when each boolean literal is considered in subsequent computations. Figure 2 demonstrates the data preparation pipeline on the left side. The raw features (integer or floating point values)

[1] https://tinyurl.com/MSP430FR5xxx

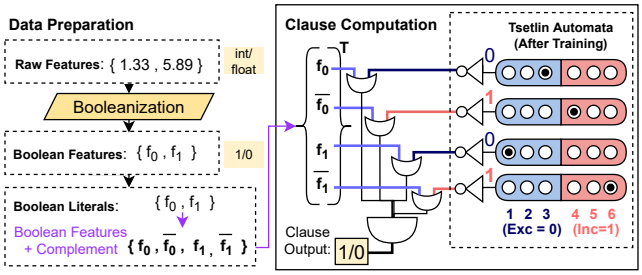

Fig. 2: Data preparation, Tsetlin Automata and the Clause unit.

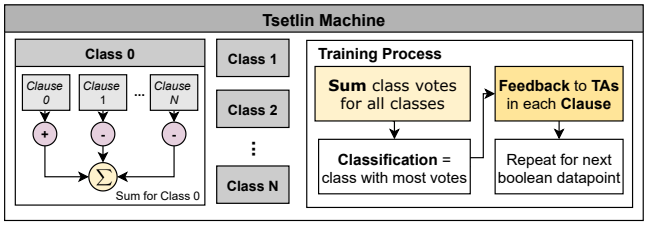

Fig. 3: Tsetlin Machine Architecture.

are booleanized into *Boolean Features* (1 or 0 values). For simplicity, in our example, we chose one bit representation for each raw feature, but this is a design choice that should be made depending on the application. Boolean features are then converted to *Boolean Literals*. This is simply the Boolean feature itself and its complement. These Boolean literals form the inputs to the TM. Therefore booleanization of the raw input data allows the user to control the granularity of the inputs to the TM i.e., the number of boolean literals.

Architectural Building Blocks. The right-hand side of Figure 2 shows two of the fundamental elements of the TM: the *Tsetlin Automata* and the *Clause*. Each Tsetlin Automaton (TA) has a finite number of states (6 in our figure), half of which correspond to *Exclude* and the other half corresponds to *Include*. The TAs can either be viewed by their state numbers (1 to 6 as seen in Figure 2) or by their binary include and exclude decisions (i.e., 0 for exclude and 1 for include). A TA is instantiated for every Boolean literal which is composed of each Boolean feature and its complement. The TA is the learning element that is optimized during training, much like the weights in a Neural Network. During training, the current state of each TA changes at each step and settles in near optimum include/exclude decisions at the end of the process, when good classification accuracy is reached. The Boolean Literals and their respective TA include/exclude decisions are then fed to the clause unit, hence creating the propositional logic.

The clause relates the input data (Boolean Literal) to its respective learning element (TA). As shown through Figure 2, the include/exclude decision of the TA is combined with its Boolean Literal through a simple logic circuit. The output of a clause is a single bit. The number of clauses is a parameter the user will configure much like the number of filters or layers in a Neural Network. Typically, higher number clauses result in better accuracy as there is greater likelihood of the TM finding the right propositions. The clause highlights why we refer to the data preparation step as Booleanization rather than binarization, all notion of place value is lost once the Boolean literal reach the clause proposition logic and each Boolean literal has equal weighting in the clause expression.

Full TM Model. Figure 3 shows the architecture of the TM for training. The clauses are grouped together for each class with an equal number of clauses per class. The one bit clause outputs are multiplied with a positive or negative polarity ($\times 1$ or $\times -1$) and summed for each class. The polarity allows each clause to learn both supporting and opposing propositions for their respective class. Upon summing the class votes across the whole system, the classification becomes the class with the most votes. The computation stops here at inference time. At training time instead, based on guessed and actual class, *Feedback* is given to each TA to transition their state. Feedback is only necessary in the training stage. Once trained the positions of the TA states are fixed. The process repeats for all boolean datapoints with the convergence of the TM typically occurring within few epochs [14]. For an in-depth explanation of Tsetlin Machines readers are encouraged to review [23] and [8].

In this paper we examine the benefits and challenges of using TMs for inference on intermittently-powered systems. So far the only attempts at embedded TMs have simply translated the C based TM implementation to a MCU [14].

2.3 Benefits and Challenges of Intermittent Tsetlin Machines

Focusing on the hierarchical architecture of the TM in Figure 3, we observe how, from an intermittent-execution perspective, TMs allow for greater choice over task division [16, 24]. The developer can define tasks at the TA level, clause level or class level depending on the application scale. Since the computation of each unit is independent to the other, the data transfer between the units is minimum, unlike conventional neural networks (as shown in Figure 1). This flexibility is important for task-based intermittent systems [16, 24] as it minimizes the overhead, and simplifies the selection of appropriate task division: both crucial aspects to consider to ensure the application forward progress.

While the logic based clause propositions offer both complexity reduction and energy efficiency potential, in order to achieve sufficiently high accuracy we must increase the number of clause instances in the system, consequently increasing memory cost [8]. Much like the weights in DNNs, the number of TAs contribute to the main memory footprint of TM models. The size of a TM is defined as the number of TAs, also written as number of classes $\times$ number of clauses $\times$ number of Boolean literals. An increase in any one of these terms will result in substantial increase in the number of TAs. For example, increasing the number of clauses for better accuracy, increasing the number of classes for larger problems, or increasing the number of boolean literals for problems that require more granular feature representation will all have a significant effect on the model size. There is no clear trade-off formulation to this problem and the design choices for the number of clauses and number of Boolean literals is always application dependent.

3 INTERMITTENTLY-POWERED TM

We now shift our attention to the application of TM in an intermittent-power setting. This demands careful strategies to 1) encode the TA states to fit into the limited memory of MCUs common in intermittently-powered systems, and to 2) determine appropriate intermittent execution models. We reflect on these aspects below.

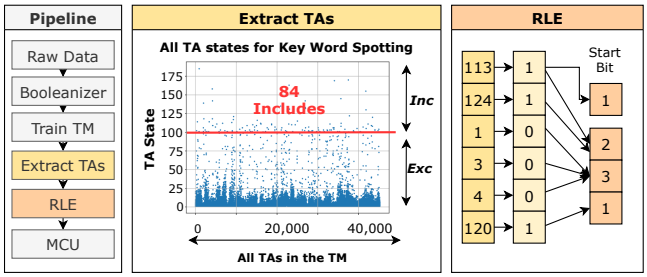

Fig. 4: Embedded TM pipeline using Run Length Encoding.

3.1 Encoding TM Models

We have seen that the memory requirement to store the state of each TA increases when the classification task becomes more complex (i.e., number of classes and input dimension increases) or when the classification capacity of the model needs to be increased (i.e., more clauses). However, we notice two intrinsic properties of the TM: 1) there is no need to store the actual value of each TA state but just their binary include/exclude decision, and 2) typically the number of exclude decisions far outnumber that of the include decisions. This implies that if we represent exclude with *0* and include with *1*, we will observe very large runs of *0s* separated by a single *1*. Hence we propose to use Run Length Encoding (RLE) to compress the TAs after training. With this approach, long sequences of the same value (e.g., *0*) are replaced by its count. The ratio of exclude to include TA decisions can be attributed to the granularity of the boolean inputs, a richer input feature representation may result in the TM selecting more include features, this may result in better accuracy but at the cost of a lower compression ratio.

Figure 4 shows the full pipeline for the embedded RLE-TM we propose. Through discretisation of the actual TA states followed by tallying the runs of 1s and 0s we are able to significantly reduce the model's memory requirements. Notice that this is a lossless compression since the original include/exclude sequence can always be recovered, resulting in zero accuracy loss at inference time. The Extracted TAs blocks show the TA states post-training, plotted in Figure 4 to visually demonstrate the ratio of include to exclude decisions. Note that the middle state separating includes from excludes here is 100, the number of states in the TA is a design choice. Using a greater number of states leads to more fine-grained decision-making but will also require more epochs for the TAs to settle into optimum include/exclude positions. In Figure 4 the TM shows each TA has 200 states and there are 45240 TAs altogether, only 84 TAs are include decisions for an example Key Word Spotting (KWS) application. The substantial imbalance between the include and exclude decisions allow for very large runs of excludes separated by one include enabling significant compression ratios to be achieved through RLE encoding.

When considering intermittently-powered systems, this compression approach not only reduces the memory requirement of a TM-based classification task but improves also its latency. Infact, the FRAM—used to store program code and non-volatile program state—on MSP430FR5xxx MCUs, is much slower than the on-chip SRAM and the compiler inserts wait cycles if there is a mismatch between system clock and FRAM max speed. Hence, compression ratios of up to 99% implies that memory operations are reduced by almost 99% too. This substantial reduction in memory accesses translates, on such architecture, to significant improvements in latency and energy efficiency compared to vanilla TMs. However, some of the improvement is sacrificed on decoding TAs. We discuss this speedup in § 4.

Model	Structure		
	MNIST	CIFAR	KWS
FC BNN	L1-512 L2-256 (96.97%)	L1-512 L2-256 L3-128 (80.91%)	L1-256 (81.70%)
Conv. BNN	L1-10 (95.55%)	L1-10 (85.39%)	L1-10 L2-10 (71.53%)

Table 1: BNNs model structure and accuracy for the three datasets. LX-Y indicates layer number X, and Y neurons for FC models and filters for conv. models, respectively.

3.2 Intermittence-Aware Implementation

Booleanization: We use well known and commonly used booleanization methods for the datasets evaluated. This is done through either pre-defined functions such as *Adaptive Thresholding* available in the OpenCV library, quantile binning based on the distributions of each feature or simply creating equally spaced fixed thresholds between the maximum and minimum input values [14, 23].

Execution Model: We use a popular intermittent execution model: InK [24]. In InK, applications consist of atomic *tasks*—inside a *task thread*—that can do computation, sensing, or other actions, and have access to shared memory. InK schedules these tasks and maintains memory consistency and progress of computation across power failures. Tasks within a thread communicate with each other by manipulating task-shared variables. InK allocates these variables in the non-volatile memory and keeps them double-buffered to preserve data consistency. Before running any task, the scratchpad buffer is initialized by the content of the original buffer, and then buffer pointers are swapped at task completion. This happens before the execution of each task even if only one variable in the buffer is modified, thus leading to significant overhead. However, the overhead is much lower than other systems [24]—making it the main reason for choosing this over other state-of-the-art works.

Implementation: For intermittent execution, the program code needs to be broken down into small chunks, *tasks*, that can be executed atomically. The simple architecture of TMs makes task division easy without putting burden on developers. Leveraging the fact that the compute of one class is independent of other classes, we choose an intuitive task division strategy and put all the operations related to one class in one task. This ensures minimal overhead for storing and restoring intermediate task buffers.

4 EVALUATION

4.1 Methodology

Datasets: We evaluate our RLE TM using three datasets.

MNIST [6] is a standard benchmark dataset composed of 70k 28x28 pixel images of handwritten digits.

CIFAR-2 is a 2-class variation of the common CIFAR-10 dataset[13] where we group all the vehicle images into one class and all the

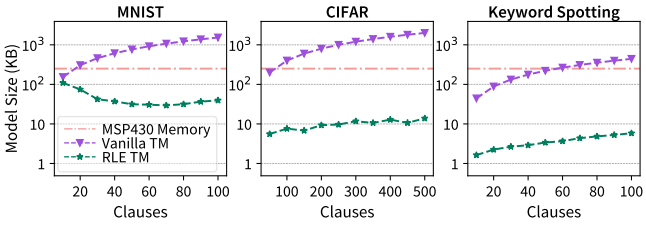

Fig. 5: Model size of vanilla TMs and RLE-TMs. RLE compresses the model by up to 99%. (Log scale on y-axis).

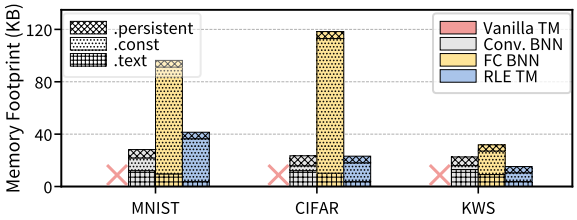

Fig. 6: Memory footprint of TMs and baseline BNNs.

animal images into another class. The dataset consists of 60k 32x32 colour images which we convert to grayscale for our evaluation.

Speech Commands [22] includes 105k 1-second long utterances of 35 spoken words. We use 6 keyword classes *yes, no, up, down, left, right* and the booleanization pipeline from [14].

We use MNIST to establish a baseline for our evaluation models as typically done in the machine learning community. The other two datasets instead, represent image and audio recognition tasks that could potentially be achieved by low-power devices as first classification stage before triggering a more powerful system.

Models: Firstly, our evaluation focuses on comparing our RLE Tsetlin Machine against the vanilla TM. Since the number of clauses is one of the main hyperparameters for TM models, we train 10 models with an increasing number of clauses. This allows to study how the model's performance changes in relation to its capacity.

For the baselines, we use deep neural networks (DNN) which are the state of the art for image and audio classification. In particular, given that TMs use booleans for the input data and the internal representations, we select binary neural networks (BNN) as the closest model to a TM. For our evaluation we use the optimised implementation and models provided by McDanel et al. [17] and given the difficulty in splitting DNN workloads into tasks, we use task-tiling as done by Gobieski et al. [7]. Task-tiling splits loop iterations into tasks executing a fixed number of iterations. In our implementation, we compute one neuron and apply one filter per task in fully-connected (FC) and convolutional BNNs, respectively. This division might not be ideal as its optimisation depends on the characteristics of each model (e.g., number of layers and neurons/filters), deployment environment, harvester, and the size of capacitor. However, as we did for our TM implementation, we opted for an intuitive task split that allows application developers—having less domain specific knowledge—to write programs that can run intermittently without any memory inconsistencies. Table 1 reports the BNNs details for each dataset with their respective accuracy.

Performance metrics: We use model size, model accuracy, runtime memory, latency and energy per inference as metrics to compare the vanilla TM, our RLE TM and the BNN baselines, both with continuous and intermittent power. For the on-device evaluation we use the MSP430FR5994 equipped with 256KB of memory.

4.2 Memory Usage

We first compare the model size of vanilla TM with RLE TM. Figure 5 shows that as clauses increase, the vanilla TM model grows linearly while the encoded model size increases at a much lower rate. This means the compression ratio gets better when the number of clauses increases. RLE achieves maximum 97.42%, 99.31%, and 98.68% compression ratio for MNIST, CIFAR, and KWS, respectively. However, for MNIST, where the ratio of exclude to include decisions is lower than CIFAR and KWS, there is poor compression at lower clauses. One interesting point to note is that for MNIST between 20 to 60 clauses the RLE-TM model size seems to decrease despite an increase in clauses. This is due to the spreading of the include decisions among the clauses. This results in larger runs of 0s and 1s and thus better compression. This effect is not noticeable in CIFAR or KWS as these are harder problems to learn and therefore the number of includes is relatively small even at low clauses. Our results show how the model size of RLE TM is much lower than the total available memory in MSP430FR5994, paving the way for the development of more accurate and realistic models with higher number of clauses that would not have been possible without RLE.

We also examine the overall memory footprint of vanilla TMs, RLE TMs and BNNs. This includes: *.text* (code), *.const* (model), and *.persistent* (non-volatile buffer and runtime management) sections of the memory. For BNNs, we use the models shown in Table 1. For TM, we use 50, 400 and 100 clauses for MNIST, CIFAR and KWS, respectively, as they offer comparable accuracy to the BNNs. Figure 6 highlights the effectiveness of RLE TM against the vanilla TM, which is too large for the MCU. We observe that for datasets with a higher ratio of exclude to include decisions, such as CIFAR and KWS, the RLE TM memory footprint is smaller than both BNN benchmarks. In the case of MNIST, where the dataset contains a lower ratio of exclude/include, the RLE TM still outperforms the FC BNN but less sparse includes lead to a larger encoded model, resulting in higher memory usage compared to the Conv. BNN.

4.3 Continuous Power Evaluation

RLE TM vs Vanilla TM: As mentioned earlier, the RLE is a lossless compression. Figure 7 shows that both vanilla and RLE TMs have the same accuracy since the encoded models can be extracted completely at run-time. It can also be observed that the latency and energy of vanilla TM is much higher than RLE TM. This is expected because of the reduced memory accesses, as mentioned in § 3.

RLE TM vs FC BNN: Figure 7 shows that TMs offer similar accuracy within 5% of BNNs. FC BNNs offer better accuracy for MNIST and KWS, however, we have shown in Figure 6 that this comes with a larger memory cost. The FC BNN does however offer better energy efficiency and latency compared to the RLE TM. This comes from the cost of decoding the RLE during inference. Through exploration into more efficient decoding procedures we can remedy the energy and latency figures. We leave this for future work.

RLE TM vs Conv BNN: When comparing accuracy, TMs are within ± 5% of Conv. BNNs. However it is worth noting the accuracy

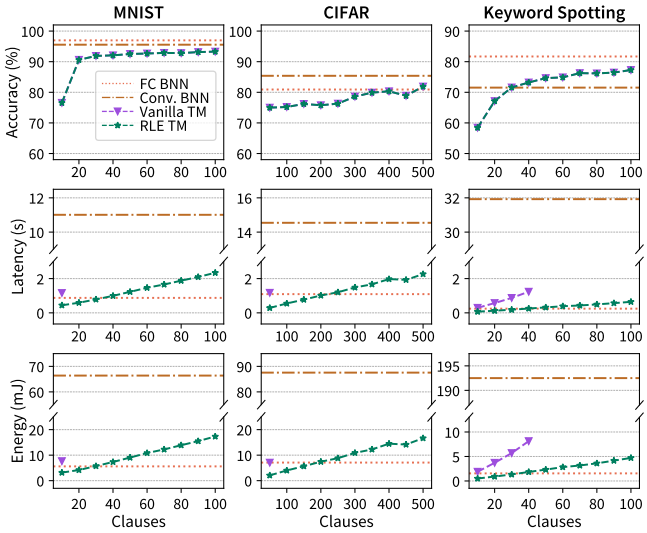

Fig. 7: Accuracy, latency and energy per inference for vanilla and encoded TMs, and BNN baselines. Vanilla TM exceeded the limited memory of MSP430 for increasing number of clauses. (Note broken y-axes in latency/energy plots.)

Dataset	Duty Cycle (%)	Latency (s) / Energy (mJ)			
		FC. BNN	Con. BNN	Vanilla TM	RLE TM
MNIST	50	1.74 / 5.61	24.00 / 73.58	✖ / ✖	2.49 / 9.22
	75	1.16 / 5.51	16.22 / 72.88	✖ / ✖	1.63 / 8.92
CIFAR	50	2.13 / 7.29	31.58 / 97.01	✖ / ✖	4.14 / 15.27
	75	1.46 / 7.00	20.00 / 90.12	✖ / ✖	2.79 / 15.14
KWS	50	0.49 / 1.57	100.00 / 307.2	✖ / ✖	1.30 / 4.79
	75	0.33 / 1.53	60.00 / 270.0	✖ / ✖	0.87 / 4.71

Table 2: Latency and energy per inference for BNNs and TMs on different intermittent power settings

advantage of the Conv. BNN comes at the cost of larger latency and energy where TMs perform better across all three datasets. We have already seen that if a given application presents a very high exclude to include ratio, the TM will have better latency and smaller memory footprint.

4.4 Intermittent Power Evaluation

We emulate intermittent power by generating a square wave (10s pulse-width) with different duty cycles using a Teensy 3.5[2] and a transistor, to compare the energy-efficiency of TMs and BNNs on different power levels. This approach has been used for evaluating recently developed systems [11, 24].

Table 2 shows the latency and energy per inference. for the models reported in Table 1. Note that vanilla TM versions of these configurations cannot fit in the 256KB FRAM of MSP430FR5994. With increasing duty cycle, the latency and energy decreases as more power leads to less downtime and less store/restore operations. Once again, we see that the RLE TM offers a good middle ground between the FC BNN and the Conv BNN for both energy and latency, with the RLE decoding overhead leading to poorer performance compared to FC BNNs. For our initial exploration, we used 50% and 75% duty cycle only. However, we aim to provide an in-depth and in-the-wild evaluation with actual harvesters in future work.

[2]https://www.pjrc.com/store/teensy35.html

From Table 2 we observe that the energy consumption using our encoding scheme is only a few milli-joules which is much lower than state-of-the-art convolutional BNNs and is quite low for batteryless devices. For comparison, SONIC & TAILS report between 40mJ and 30mJ of energy for a single inference on the MNIST dataset [7], while our approach requires less than 10mJ. This is only the energy used for the model inference. Energy consumed during sensor data acquisition needs to be added regardless of the model used (i.e., DNN-based or TM-based).

5 OUTLOOK

This paper presented an initial exploration of the logic-based, Tsetlin Machines learning algorithm in the context of batteryless sensors.

Key Takeaways: We have found TMs to be particularly suited for intermittent computing thanks to their simple architecture, which alleviates the burdensome duty of splitting computation into tasks; and thanks to the limited memory overhead used for intermediate results. However, the overall memory requirement for a TM model is still prohibitive for systems characterised by limited memory.

As proposed in this paper, RLE represents a promising approach to compress a vanilla TM up to 99%, without loss in accuracy, and resulting in latency speed ups of up to 4.9×. This results in small TM models which fit into typical batteryless HW platforms and consume significantly less energy than DNN-based counterparts. When evaluated against state-of-the-art BNNs the TMs can provide a middle ground between FC and convolutional models in terms of offering better memory utilisation and similar latency for applications where a slight drop in accuracy is acceptable. The FC BNN offers better accuracy and energy/latency but at a cost of higher memory footprint, the TM offers similar accuracy and energy expenditure but at a much lower memory footprint.

Future Directions: In our approach we have exploited the natural imbalance between include and exclude decisions at the TA level to compress the model representation. The key to exploiting RLE relies on manipulating the exclude to include ratio of the TAs to find a balance between compression and accuracy. This include/exclude ratio is a by-product of both input boolean granularity and hyperparameter choice. These two factors determine the number of state transitions that happen in the Feedback stage during training. Future work will explore the impact of both booleanization and the hyperparameters that control the Feedback on the include/exclude ratio and subsequently the RLE performance. This would enable the creation of models that are amenable to RLE-based compression directly from the training process.

Despite the high compression rates we achieve with RLE, one drawback is manifested when there are sequences of alternating 0s and 1s. In this case RLE cannot compress the sequence further. More sophisticated encoding techniques could be explored with the aim of reducing this inefficiency, resulting in even higher compression and potentially lower latency due to fewer memory accesses.

Finally, an in-depth analysis of RLE TM is needed on other HW platforms with different memory architectures. This additional exploration would assess if the befits of TMs and the compression technique we propose in this paper, could be applied to a larger set of devices, which are not necessarily intermittently powered.

REFERENCES

[1] Omid Ardakanian, Arka Bhattacharya, and David Culler. 2016. Non-intrusive techniques for establishing occupancy related energy savings in commercial buildings. In *Proceedings of the 3rd ACM International Conference on Systems for Energy-Efficient Built Environments*. 21–30.

[2] Abu Bakar, Alexander G Ross, Kasim Sinan Yildirim, and Josiah Hester. 2021. REHASH: A Flexible, Developer Focused, Heuristic Adaptation Platform for Intermittently Powered Computing. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies* 5, 3 (2021), 1–42.

[3] Domenico Balsamo, Alex S Weddell, Geoff V Merrett, Bashir M Al-Hashimi, Davide Brunelli, and Luca Benini. 2014. Hibernus: Sustaining computation during intermittent supply for energy-harvesting systems. *IEEE Embedded Systems Letters* 7, 1 (2014), 15–18.

[4] Gregory Chen, Hassan Ghaed, Razi-ul Haque, Michael Wieckowski, Yejoong Kim, Gyouho Kim, David Fick, Daeyeon Kim, Mingoo Seok, Kensall Wise, et al. 2011. A cubic-millimeter energy-autonomous wireless intraocular pressure monitor. In *2011 IEEE International Solid-State Circuits Conference*. IEEE, 310–312.

[5] Andy Rosales Elias, Nevena Golubovic, Chandra Krintz, and Rich Wolski. 2017. Where's the bear? Automating wildlife image processing using iot and edge cloud systems. In *2017 IEEE/ACM Second International Conference on Internet-of-Things Design and Implementation (IoTDI)*. IEEE, 247–258.

[6] LeCun et al. 1998. The MNIST database of handwritten digits. http://yann.lecun.com/exdb/mnist/. Accessed: 10-21-2021.

[7] Graham Gobieski, Brandon Lucia, and Nathan Beckmann. 2019. Intelligence beyond the edge: Inference on intermittent embedded systems. In *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*. 199–213.

[8] Ole-Christoffer Granmo. 2018. The Tsetlin Machine–A Game Theoretic Bandit Driven Approach to Optimal Pattern Recognition with Propositional Logic. *arXiv preprint arXiv:1804.01508* (2018).

[9] Josiah Hester and Jacob Sorber. 2017. The future of sensing is batteryless, intermittent, and awesome. In *Proceedings of the 15th ACM Conference on Embedded Network Sensor Systems*.

[10] Bashima Islam and Shahriar Nirjon. 2019. Zygarde: Time-sensitive on-device deep inference and adaptation on intermittently-powered systems. *arXiv preprint arXiv:1905.03854* (2019).

[11] Vito Kortbeek, Abu Bakar, Stefany Cruz, Kasim Sinan Yildirim, Przemysław Pawełczak, and Josiah Hester. 2020. Bfree: Enabling battery-free sensor prototyping with python. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies* 4, 4 (2020), 1–39.

[12] Vito Kortbeek, Kasim Sinan Yildirim, Abu Bakar, Jacob Sorber, Josiah Hester, and Przemysław Pawełczak. 2020. Time-sensitive intermittent computing meets legacy software. In *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*. 85–99.

[13] Alex Krizhevsky. 2009. Learning multiple layers of features from tiny images. (2009).

[14] Jie Lei, Tousif Rahman, Rishad Shafik, Adrian Wheeldon, Alex Yakovlev, Ole-Christoffer Granmo, Fahim Kawsar, and Akhil Mathur. 2021. Low-power audio keyword spotting using Tsetlin machines. *Journal of Low Power Electronics and Applications* 11, 2 (2021), 18.

[15] Brandon Lucia, Vignesh Balaji, Alexei Colin, Kiwan Maeng, and Emily Ruppel. 2017. Intermittent computing: Challenges and opportunities. In *2nd Summit on Advances in Programming Languages (SNAPL 2017)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik.

[16] Kiwan Maeng, Alexei Colin, and Brandon Lucia. 2017. Alpaca: Intermittent execution without checkpoints. *Proceedings of the ACM on Programming Languages* 1, OOPSLA (2017), 1–30.

[17] Bradley McDanel, Surat Teerapittayanon, and HT Kung. 2017. Embedded binarized neural networks. *arXiv preprint arXiv:1709.02260* (2017).

[18] Alessandro Montanari, Mohammed Alloulah, and Fahim Kawsar. 2019. Degradable inference for energy autonomous vision applications. In *Adjunct Proceedings of the 2019 ACM International Joint Conference on Pervasive and Ubiquitous Computing and Proceedings of the 2019 ACM International Symposium on Wearable Computers*. 592–597.

[19] Alessandro Montanari, Manuja Sharma, Dainius Jenkus, Mohammed Alloulah, Lorena Qendro, and Fahim Kawsar. 2020. ePerceptive: energy reactive embedded intelligence for batteryless sensors. In *Proceedings of the 18th Conference on Embedded Networked Sensor Systems*. 382–394.

[20] Matteo Nardello, Harsh Desai, Davide Brunelli, and Brandon Lucia. 2019. Camaroptera: A batteryless long-range remote visual sensing system. In *Proceedings of the 7th International Workshop on Energy Harvesting & Energy-Neutral Sensing Systems*. 8–14.

[21] Philip Sparks. 2017. The route to a trillion devices. White Paper, ARM.

[22] Pete Warden. 2018. Speech commands: A dataset for limited-vocabulary speech recognition. *arXiv preprint arXiv:1804.03209* (2018).

[23] Adrian Wheeldon, Rishad Shafik, Tousif Rahman, Jie Lei, Alex Yakovlev, and Ole-Christoffer Granmo. 2020. Learning automata based energy-efficient AI hardware design for IoT applications. *Philosophical Transactions of the Royal Society A* 378, 2182 (2020), 20190593.

[24] Kasım Sinan Yıldırım, Amjad Yousef Majid, Dimitris Patoukas, Koen Schaper, Przemyslaw Pawelczak, and Josiah Hester. 2018. Ink: Reactive kernel for tiny batteryless sensors. In *Proceedings of the 16th ACM Conference on Embedded Networked Sensor Systems*. 41–53.