Adaptive Intelligence for Batteryless Sensors Using Software-Accelerated Tsetlin Machines

Abu Bakar*†
Georgia Institute of Technology
Atlanta, USA
abubakar@gatech.edu

Tousif Rahman*†
Newcastle University
Newcastle upon Tyne, UK
s.rahman@newcastle.ac.uk

Rishad Shafik
Newcastle University
Newcastle upon Tyne, UK
rishad.shafik@newcastle.ac.uk

Fahim Kawsar
Nokia Bell Labs and University of Glasgow
Cambridge, UK
fahim.kawsar@nokia-bell-labs.com

Alessandro Montanari
Nokia Bell Labs
Cambridge, UK
alessandro.montanari@nokia-bell-labs.com

ABSTRACT

Tsetlin Machine (TM) is a new machine learning algorithm that encodes propositional logic into learning automata—a set of logical expressions composed of boolean input features—to recognise patterns. The simplicity, efficiency, and accuracy of this logic-based algorithm encourage rethinking the application of traditional arithmetic-based neural networks (NNs) in intelligent sensors design. Indeed, TM is a promising candidate for embedding intelligence into tiny batteryless sensors with the potential to address two critical challenges: (1) computing under resource constraints and (2) demand for dynamic adaptation to the unpredictable nature of harvested energy. However, its structural model complexity manifests in two conflicting issues: large memory footprint and long latency. This paper addresses these shortcomings by proposing adaptive compression techniques exploiting the inherent redundancies observed in trained models. Through dynamically scaling the computational complexity based on available energy, our techniques significantly reduce the memory footprint and speed up the runtime execution. We evaluate our techniques against standard TMs and binarized neural networks (BNNs) for vision and acoustic workloads deployed on a TI MSP430 MCU operating under intermittent power supply conditions. We show that our techniques can achieve up to 99% compression of TM models and offer 13.5× latency and energy reductions when compared with the most efficient neural network configuration without compromising accuracy.

CCS CONCEPTS

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

*Both authors contributed equally to the work.
†Work done while both authors were at Nokia Bell Labs, Cambridge (UK).

SenSys '22, November 6–9, 2022, Boston, MA, USA

ACM ISBN 978-1-4503-9886-2/22/11…$15.00
https://doi.org/10.1145/3560905.3568512

KEYWORDS

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

ACM Reference Format:
Abu Bakar, Tousif Rahman, Rishad Shafik, Fahim Kawsar, Alessandro Montanari. 2022. Adaptive Intelligence for Batteryless Sensors Using Software-Accelerated Tsetlin Machines. In *The 20th ACM Conference on Embedded Networked Sensor Systems (SenSys '22), November 6–9, 2022, Boston, MA, USA.* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3560905.3568512

1 INTRODUCTION

Harvested ambient energy from solar, kinetic, radio frequency, or other sources has the potential to enable autonomous operation of batteryless sensors making edge applications maintenance-free and long-lived for a sustainable future [16, 48]. However, these devices are impractical without the pairing of an ML model which can compute meaningful results directly on the sensor, without the need for high-energy data transmission to the cloud [1, 12, 39].

Recent advances in software framework optimisations made deep neural networks (DNNs) the favoured approach to embed intelligence into microcontrollers given their high accuracy on complex tasks, such as vision and acoustic classification [12, 20, 25, 29, 38, 39]. The TinyML community has focused on addressing two main challenges faced when deploying DNNs on microcontrollers: large memory footprint and high latency. Techniques like weights quantisation [21], pruning and layer decomposition [22, 30], or neural architecture search [28, 29] could alleviate these issues by ensuring that models fit in memory and increasing execution speed. Nevertheless, they suffer accuracy drops when heavy compression is applied and require significant offline processing to produce and fine-tune these models [6, 15, 28, 29, 52].

Previous works on efficient DNNs deployment typically consider relatively powerful microcontrollers with 32bit architectures, floating point units (FPUs), SRAM up to 512kB and running at frequencies of 100MHz or higher. They also assume a constant power supply from a battery [28, 29]. These characteristics allow powerful models to run on such small devices. However, when considering batteryless systems, power efficiency is of paramount importance since the operation is supported only by nominal harvested energy. Hence, much smaller and efficient MCUs are used. For example a MSP430FR5xxx [19], a standard series of MCUs for

running intermittently-powered applications, has a 16-bit architecture, runs at maximum 16MHz, has no FPU and supports up to 8kB of SRAM, making DNN deployments even more challenging.

Another critical challenge of deploying DNNs on batteryless sensors is that energy is scarce, intermittent, and unpredictable, as such the device can suffer several power failures during a single inference. DNN models, typically monolithic, are unable to flexibly adapt to fluctuating energy availability, and once trained, they produce inferences only when there are enough resources (i.e., energy) to execute the entire model. While intermittence-safe implementations can guarantee that an inference could span several power failures [12], they do not automatically enable existing models to produce a valid output that meets the instantaneous energy envelope. Recent works started exploring this avenue but still incur significant inference latency [3, 20, 38]. A recognition approach that could easily scale to very resource constrained devices and naturally adapt to fluctuating energy availability is still highly desirable for batteryless systems.

To find an approach that satisfies the requirements of a batteryless system we turn our attention towards a logic-based learning algorithm called the Tsetlin Machine (TM), which is an emerging machine learning algorithm utilising the principles of learning automata and game theory [13]. 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). Despite the algorithm still being far from the level of sophistication of artificial neural networks, which have enjoyed more than 60 years of development from the Rosenblatt's Perceptron [44], several works reported promising results for Tsetlin Machines, especially in situations when the efficiency of the models is of utmost importance. Infact, in the last few years, extensive effort is being dedicated to studying the modeling and recognition performance of TMs compared to deep neural networks. TMs have demonstrated competitive performance compared to DNNs when applied to standard ML benchmarks [11, 13, 14, 27, 46] as well as applications like natural language processing [5, 45, 51], audio classification [26] and biomedical recognition [41].

In this work, we make the first step towards understanding if TMs could represent a viable option for recognition tasks in the very constrained environment of intermittently-powered systems. The architectural simplicity makes TM a promising candidate for intermittently-powered systems. From a developer perspective, it makes writing task-based applications, usually a daunting exercise for developers [32, 53], significantly simpler. From an efficiency perspective, since the computation of each class is independent of one another (contrary to the case with neural networks), no data has to be transferred in-between different computational units back and forth. Hence, only a small portion of the memory needs to be written to the non-volatile memory and restored at each power cycle, minimising the overhead introduced by power-failure-agnostic runtimes [53]. Nevertheless, TMs still suffer from substantial memory footprint and latency when deployed on constrained sensors [26].

To address these issues, we present the design and implementation of **Lite-TM**, a novel framework to enable the deployment of practical TM models on intermittently-powered systems. Lite-TM is built around three core techniques which: (1) reduce the memory footprint of a TM model, (2) speed up model execution, and (3) dynamically scale model complexity based on available energy. The first technique is based on the observation that the actual states of the Tsetlin Automata, the only memory elements required for TMs, are not necessarily used at run time for inferences, hence can be efficiently encoded to save memory. We then achieve a substantial reduction in inference latency by determining at deployment time which clause logic propositions, the computational elements of TMs, do not need to be executed at inference time because their output is already determined and does not depend on the current input. Finally, we enable TM models to adapt inference quality in proportion to the instantaneous energy supplied by ranking the clause propositions based on their contribution to a correct prediction and dropping less useful ones when energy is scarce. We evaluate our framework on realistic vision and acoustic datasets [23, 49] showing significant memory and latency improvements over the vanilla TM implementation and state-of-the-art binarized neural networks - e.g. 98.9% compression with 13.5× speedup for our CIFAR image recognition dataset. Additionally, when powered with real RF and solar energy harvesters our adaptation approach increases the inference throughput by up to 2.2× compared to static models while ensuring a maximum accuracy drop of less than 2.4%.

In summary, our contributions are the following:

- A novel framework for automated deployment of efficient and adaptive Tsetlin Machine models on intermittently-powered batteryless systems.
- Two encoding techniques to improve TMs' memory usage and latency without a drop in accuracy based on a newly discovered characteristic of trained TM models.
- A run-time adaptation technique to manipulate TM model complexity in accordance with variable energy.
- A detailed comparison of optimised TMs with state-of-the-art embedded binarized neural networks (BNNs) with constant and intermittent power from realistic scenarios.

This work represents a first step toward exploring an alternative classification algorithm to deep neural networks for intermittently-powered systems. Our findings demonstrate that using our proposed approaches, practical TM models can be deployed on embedded and batteryless devices achieving accurate predictions with high energy efficiency.

2 BACKGROUND AND RELATED WORK

2.1 Intermittent Computing

The vision of ubiquitous computing will require sensors to harvest ambient energy to ensure a long lifetime and low maintenance cost. The sporadic availability of harvested energy makes the continuous execution of the programs impossible [31]. 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 are 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 is approaching, the content of volatile memory is stored in non-volatile memory, and the execution is restored from the same point when the device reboots after a power outage. Several models exist to ensure memory consistency and robustness to sudden power failures [4, 17, 32, 42, 53], however, they typically lead to significant memory and latency overhead due to the continuous store/restore process.

2.2 Efficient and Adaptive Machine Learning

Deep Neural Networks (DNNs) have shown impressive performance on different recognition tasks, ranging from visual and acoustic understanding to human activity recognition, and are now becoming the preferred choice for adding intelligence to resource-constrained devices [35, 36]. Deploying such models, however, comes with significant challenges. In fact, several techniques have been proposed to modify traditional DNNs with the goal of fitting them in memory, increasing execution speed, and decreasing energy demand. Examples include pruning redundant weights and filters during or after training and later fine-tuning the model in order to decrease the model size without a high loss in accuracy [6, 15, 52]. Some of these techniques have been recently used to deploy DNN models on intermittently-powered systems with the objective of enabling useful applications (e.g., wildlife conservation, healthcare, and agriculture) while significantly reducing data transmission overhead [12, 20, 39]. However, these DNN workloads still come with a significant burden in terms of resources they require, i.e., memory and energy. In this paper instead, we adopt Tsetlin Machines (TMs), a different learning algorithm that does not rely on heavy floating-point arithmetic operations but is based on propositional logic and game theory [13], promising more efficient inferences. We offer an introduction to TMs and their fundamental components in §2.3.

A particularly important aspect when deploying machine learning workloads on intermittently-powered systems is their ability to adapt to fluctuating harvested energy. The techniques mentioned above to scale DNN models for constrained devices aim at reducing the complexity of a single model but do not consider the problem of adapting the model execution to match variable energy [6, 15, 28, 29, 52]. A few approaches have been proposed to adapt DNN model complexity at runtime in mobile systems [9] and in intermittently-powered systems [20, 38]. However, they either require expensive and complicated training procedures [9] or still suffer from large memory consumption or high-latency inferences [20, 37, 38]. In this work, for the first time, we propose a technique to adapt the complexity of a Tsetlin Machine model. Our approach does not involve complex re-training steps, as often the case for DNNs, but relies on intrinsic characteristics of a trained TM model to skip computation at runtime.

2.3 Tsetlin Machines Primer

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

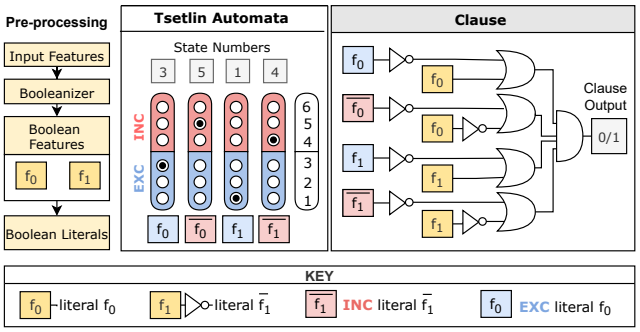

Figure 1: Tsetlin Automata and the Clause unit.

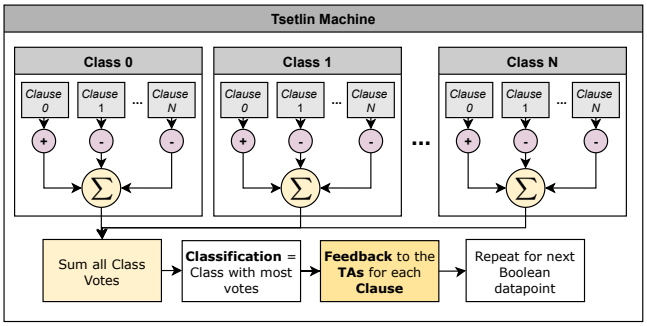

Figure 2: Tsetlin Machine Architecture.

propositions are used to determine the classification. In this section we offer an overview of the core components of the TM through Figures 1 and 2. Then show how these are algorithmically implmented through Algorithm 1.

Figure 1 shows the typical input pipeline and two of the fundamental elements of the TM: the *Tsetlin Automata (TA)* and the *Clause*. The first 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, i.e., if a floating point number is binarized to 4 bits it will have a most significant bit and least significant bit (1100), after booleanization it will be considered as individual bits (1,1,0,0). This is then converted to boolean literals where each element is recorded along with its binary negation (1,0,1,0,0,1,0,1) and used for clause computation. Therefore booleanization of the raw input data allows the user to control the granularity of the inputs i.e., the number of boolean literals.

Looking more into the details of a TM (Figure 1), each Tsetlin Automaton has a finite number of states, half of which correspond to *Include* and the other half corresponds to *Exclude*. The TAs can either be viewed by their state numbers (1 to 6 as seen in Figure 1) or by their binary include and exclude decisions. A TA is instantiated for every Boolean feature and its complement. This is referred to as the *Boolean Literal*. The Boolean Literals and their respective TA include/exclude decisions are used to create the propositional logic seen through the clause.

The clause represents the main element in the computational path that leads from the input data to the output classification. Clauses implement a fixed logic proposition as depicted in Figure 1.

Algorithm 1: High Level Pseudo-code for TM inference.

```
Input : M, N, C, L
/* M - no. classes, N - no. clauses, C - clause
   model, L - input boolean literals                */
Output : predicted_class
/* Evaluate the first class sum (Class 0). Clause
   computation is done using the logic block shown in
   Figure 1 (right).                                */
class_sum ← 0; predicted_class ← 0;
for j ← 0 to N do
    out ← clause_output(C_j^0, L) // 1 bit clause output
    /* Dealing with clause polarity:                */
    j mod 2 ? (out ← out) : (out ← −out);
    class_sum = class_sum + out;
/* Evaluate remaining classes' class sums.          */
for i ← 0 to M − 1 do
    new_class_sum = 0;
    for j ← 0 to N do
        out ← clause_output(C_j^i, L) // 1 bit clause output
        j mod 2 ? (out ← out) : (out ← −out);
        new_class_sum ← new_class_sum + out;
    if new_class_sum > class_sum then
        class_sum ← new_class_sum;
        predicted_class ← i
```

The number of clauses is a parameter the user will configure. Typically, higher clauses result in better accuracy as there is a greater likelihood of the TM finding the right propositions. Through the training process, the TM will attempt to find the best configuration of include/exclude decisions for the TAs such that a correct classification can be made after the clause computation. The simplicity of the main TM building blocks and the fixed structure of the clauses are the main aspects that make TMs suitable for constrained devices and amenable to be implemented on specialised silicon chips.

Figure 2 shows the complete architecture of a TM model. The clauses are grouped together for each class with an equal number of clauses per class. The clause outputs are multiplied with an alternating positive or negative polarity (pink circles in Figure 2) 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 predicted and actual class, *Feedback* is given to each TA to transition their state. The process repeats for all boolean data points with the convergence of the TM typically occurring within the first few epochs [26]. Describing the details of the TM's training procedure is outside of the scope of this paper since we focus on optimising the inference procedure. For this work we rely on the standard TM training procedure as defined by Granmo [13].

Algorithm 1 offers a high-level look into how the TM's inference process can be implemented. As seen, the process is driven by calculating the class sum for each class. The sum itself is created by computing clause outputs as shown by the *clause_output* function, which represents the compute happening in Figure 1. Following the clause computation, the polarity of the clause is used as a multiplier; the clauses of positive and negative polarity are organised in an alternating manner (as seen with Figure 2). The class sums are iteratively computed, and the max sum is the classification.

2.4 Benefits and Challenges of Intermittently-Powered Tsetlin Machines

Focusing on the hierarchical architecture of the TM in Figure 2, we observe how, from an intermittent-execution perspective, TMs allow for greater choice over task division. The developer can define tasks at the TA level, clause level, or class level depending on the application scale. This flexibility is important for task-based intermittently-powered systems [32, 53] as it minimises the overhead, and simplifies the selection of appropriate task division: both crucial aspects to consider to ensure the forward progress of the application. Additionally, since the computation of each unit is independent of the other, the data transfer between the units is minimal, unlike conventional neural networks. Hence, 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, reducing potential latency overheads.

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 and latency costs [13]. Much like the weights in DNNs, the number of TAs contributes to the main memory footprint of TM models. The size of a TM is defined as the number of TAs, also written as the number of classes × number of clauses × number of Boolean literals. An increase in any one of these terms will result in a 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 and its latency, far exceeding the capacity of a typical batteryless MCU [19]. We will study this tradeoff more in detail in §8. Therefore it is paramount that memory compression and latency reduction are explored.

Exploiting the boolean nature of TM models, simple run-length encoding (RLE) has been used for model compression [2]. However, RLE is naive and not scalable across different datasets which might present complex include/exclude patterns resulting in RLE not compressing the model to its maximum potential. Additionally, while RLE can achieve good memory compression, it does not improve the latency (and energy efficiency) of the models sufficiently to compete with optimised binary fully connected networks [2]. In this work instead we propose a complete framework and two novel techniques that further improve memory compression (μTM presented in §4) and significantly reduce latency (αTM presented in §5) compared to the RLE-based approach presented in [2]. We also introduce a method to make TM models adaptive (in terms of latency and energy consumption) to fluctuating energy availability (see §6), which was not considered in [2]. We offer a comparison between the compression approaches introduced in this work and the technique from [2] in §8.

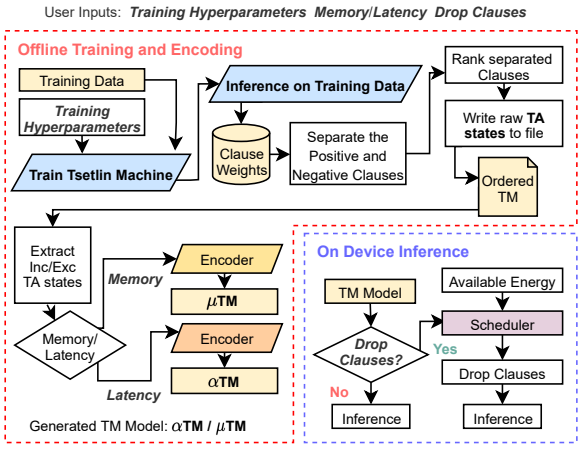

Figure 3: Framework for the automated transition of Tsetlin Machines from training to optimised inference on microcontrollers.

3 Lite-TM OVERVIEW

We have mentioned in the previous section (and we will see in more detail in § 8) that when TMs are naively deployed on intermittently-powered devices, they suffer from a large memory footprint and result in slow inferences. Moreover, similar to DNNs, vanilla TMs can produce a valid inference only when there is sufficient energy to execute the entire model (i.e., all clauses for all classes).

In this work we propose Lite-TM, a framework specifically designed for training and deploying TM models on intermittently-powered devices. The framework is built on top of three novel techniques to (1) reduce the memory footprint of a TM model enabling its deployment on constrained devices; (2) speed up model execution ensuring efficient inferences within the minimum energy budget; and (3) enable the model to flexibly adapt its computational complexity (i.e., latency) to accommodate variable energy.

At the core of the framework, we develop a pipeline that automates the transition from a trained TM model to the deployment on a microcontroller. Our pipeline allows the user to decide whether to place precedence on either minimising memory footprint to scale to larger recognition tasks, or minimising inference latency where energy efficiency is paramount. Once the model has been optimised for either memory footprint or latency, the user can optionally enable runtime adaptation to scale the model complexity at runtime. The pipeline block diagram is presented in Figure 3.

3.1 Training

Focusing on Figure 3 (top), the first step to use TM models on MCUs is to train a model with a given dataset. In our framework, the training of a TM model follows the standard procedure designed by Granmo et al. [13]. When the training is converged, our framework then assigns weights to the clause propositions based on their involvement in correct classifications by performing the inference routine on the entire training dataset as seen in Algorithm 2.

Each datapoint in the dataset is passed through the model, and a classification is produced for all of them. For each classification, our framework inspects the output of all clauses. If the model classified the current datapoint correctly, meaning the predicted class matches the actual class (i.e., $pred\_CL = Ground\_Truth$), the weight

Algorithm 2: Clause Weighting – Used on every datapoint.

```
1  Input : M; N; Ground_Truth
   /* M - no. classes, N - no. clauses, Ground_Truth -
      Actual Class                                      */
2  p_W ← 0; n_W ← 0; // positive and negative polarity
      Clause Weights
3  p_C; n_C; // positive and negative polarity Clauses
4  pred_Cl ← TM_inf(); // TM inference prediction
5  for j ← 0 to M do
6      if (pred_Cl == j)and(j == Ground_Truth) then
7          for k ← 0 to N do
8              if p_C_k^j == 1 then
9                  p_W_k^j ++;
10             if n_C_k^j == 1 then
11                 n_W_k^j −−;
12     if (pred_Cl == j)and(j! = Ground_Truth) then
13         for k ← 0 to N do
14             if p_C_k^j == 1 then
15                 p_W_k^j −−;
16             if n_C_k^j == 1 then
17                 n_W_k^j ++;
```

for clauses with positive polarity is incremented by one, while the weight for clauses with negative polarity is decremented. Instead, if the model produces a wrong classification, the weight for clauses with negative polarity is incremented, and the weight for clauses with positive polarity is decremented i.e., the opposite of when the class is predicted correctly. This is repeated for every data point to generate the clause weights. High-weighted positive polarity clauses are effective in supporting a classification for the class it belongs to, whereas high-weighted negative polarity clauses are effective in opposing a classification for the class it belongs to. We separate the positive clauses and negative clauses and rank them separately, we then join the two ranked clauses in the same signed ordering as shown in Figure 3. The separation of clauses into the two clause types (positive and negative) keeps track of polarity, this becomes a vital aspect in the inference where the ranked clause must be multiplied with its appropriate sign multiplier. After this, we write the TA states for this newly ordered TM to file (`Ordered TM`). This is the TM model that will be encoded and deployed on-device. The clause ordering we propose here is a computationally simple operation that needs to be performed only once after the model has been trained. However, it is crucial to enable inference complexity adaptation while ensuring a minimal accuracy drop. In fact, clauses with a lower contribution to a correct classification can be dropped earlier. We will see in more detail in §6 and §8 how we utilise the ordering to design an adaptive TM model and how this translates to latency reduction with limited accuracy degradation.

3.2 Encoding

In the next step of the pipeline, we introduce two encoding techniques for the TA states as a way to remedy the large model sizes

post-training and reduce inference latency. Both encoders in our framework exploit a fundamental property of the clause-based learning mechanism in the TM, i.e., the number of TA exclude decisions for literals far outnumber the number of TA include decisions. This occurs through the feedback process and is an important design choice as clauses pick out only a few key features and thus are less prone to overfitting. This results in a sparse boolean space that can undergo lossless compression into memory- and latency-optimised data structures using our encoders. Both encoders reduce the memory footprint compared to a vanilla TM but Micro TM (μTM) is particularly designed to optimise memory usage while Accelerated TM (αTM) specifically reduces latency (more details in §8). The user can select the trade-off between memory and latency for their application. This generates either the μTM or αTM models which are then deployed on the target microcontroller. While the two encoding approaches are currently mutually exclusive, the user can still decide to encode the same model with both approaches and then at runtime select which one to use based on the application requirements (e.g., faster inference or reduced memory footprint during inference). As we will see more in detail in §8 this is enabled by our encoding schemes, which both significantly reduce the model size allowing to store more than one model on the limited flash memory of a batteryless device. We leave the fusion of the two encoding schemes into a single one for future work.

3.3 Inference

After the encoding, the model is ready to be deployed. When the model is compiled for the target hardware platform, the appropriate runtime components to support μTM, αTM or both models are linked with the user application. Additionally, if the user requires to adapt the model complexity based on available energy, the Scheduler running on the device will estimate the amount of energy currently being harvested and select which clauses to drop to improve the system throughput. The clauses are dropped in reverse order compared to the weights computed at train time. Hence, the clauses that contribute the least to a correct prediction are dropped first. We will see in §8 how this approach increases the number of inferences completed in a unit of time while ensuring a minimal drop in accuracy. Notice that the adaptation technique we propose is independent of the two encoding schemes and can be applied on top of each of them or, disabled entirely.

The next three sections will present the two TM encoding methods and the runtime adaption that is achievable through dropping ordered clauses. It is important to state that while other compression schemes are possible, the designed schemes offer the best methods of compression based on TA decision sparsity and retaining as much spatial information from the original space, i.e. ability to refer back to the class and the clause polarity.

4 MEMORY-COMPACT TM: μTM

The first encoding technique is called micro TM (μTM) and it is designed to minimise the memory footprint of the model as much as possible. At the foundation of the μTM encoding, there are two intrinsic properties of TM models: 1) there is no need to store the actual value of each TA state but just their binary include/exclude

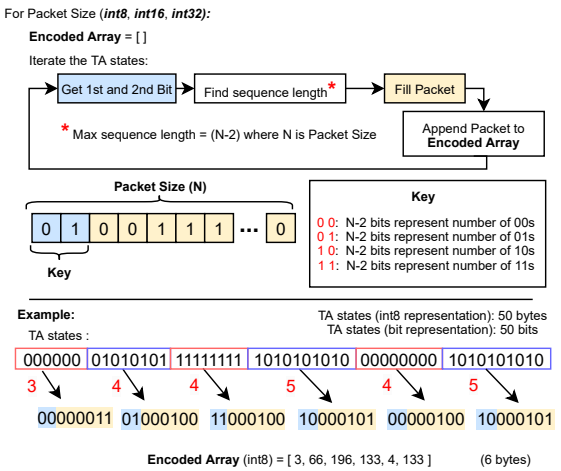

Figure 4: Micro TM (μTM) encoding method and packet structure (top). Toy example that demonstrates how a set of TA states is encoded (bottom).

decision, and 2) typically the number of exclude decisions far outnumber that of the include decisions[1]. This implies that if we represent exclude with 0 and include with 1, we will observe repeating patterns with very large runs of 0s separated by a single or few 1s. This is demonstrated in Figure 5 depicting the extracted TA states post-training for a keyword spotting (KWS) model. In a TM where each TA has 200 states and there are 45240 TAs altogether, only 84 TAs are include decisions. The substantial imbalance between the include and exclude decisions allow for very large runs of excludes separated by few includes, enabling significant compression ratios to be achieved. We are the first group to empirically observe this characteristic of TM models and to exploit it to design encoding schemes to efficiently represent TM models for on-device inference.

From these novel observations, we design an encoding scheme based on run-length encoding [43] as presented in Figure 4. The idea is that instead of simply tallying the runs of the 1s and 0s in the TA states sequence, as done with the traditional run-length encoding (RLE) [2], we count the number of repeating sequences of two bits (i.e., `00`, `01`, `10` or `11`). In practice, for the three integer types (`int8`, `int16`, `int32`), the TA states of the trained model are iterated and for each 2-bit pattern (i.e., **Key**), packets are filled with the **Key** and the length of the sequence. The packets are then appended to the **Encoded Array** as shown in the top part of Figure 4. As shown by Figure 4 we create the **Encoded Array** for each packet size. From these arrays, we use the encoded array that provides the best compression, i.e., has the fewest elements compared to the original TA states.

This method offers two advantages over the traditional run-length encoding [2]. Firstly, with μTM we can cater for alternating runs of 1s and 0s using the **Key** whereas for traditional run-length encoding we would have to store each alternating TA state into its own integer packet, as done in [2], resulting in a lower compression ratio. The encoding of alternating states is an important design choice for addressing datasets where there are more

[1]We remind the reader that in a TM model, the TA states completely define the model. In other words, all TA states need to be stored on-device to execute inferences on input data (somewhat analogously to weights in a neural network).

includes. Secondly, when decoding, the **Key** allows for two TA states to be decoded at once, benefitting inference latency. In fact, 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 the system clock and FRAM max speed. Hence, high compression ratios result in a reduced number of memory operations and consequent reduction in latency against vanilla TMs (see §8). It is also worth highlighting that we choose an RLE-based compression approach because of the trade-off between compile-time compression ratio and run-time decoding latency. The simplicity of μTM allows for faster decoding and therefore an energy-efficient inference.

Additionally, 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. This is a crucial feature of our framework because it ensures that the same accuracy achieved by the model at training time will be conserved at inference time. An important aspect when designing and deploying recognition tasks on severely constrained devices.

5 ACCELERATED TM: αTM

The process of encoding is useful in reducing the memory footprint of the model but places an additional decoding overhead on the inference process. Therefore we also propose an encoding scheme designed to reduce the total inference latency as much as possible. We refer to this encoding as αTM. Both αTM and μTM encoding methods can provide upwards of 90% model size compression, but we design αTM for greater benefits in terms of latency reduction.

The reduction in latency is possible through the role of the include in the clause proposition as seen in Figure 5 (right). When calculating the clause output the exclude states make the value of the input literal redundant. Therefore only the includes need to be calculated when evaluating the clause output. In other words, if a TA state is an exclude (represented as a logic 0) the output of the logic OR between that TA state and the corresponding input literal will always be 1, regardless of the value of the literal, as shown in Figure 5 (right). This means that at runtime, the clause computations for exclude states can be avoided, saving time during the inference. For include states instead, the value of the input literal matters and needs to be carried forward in the clause computation. This is beneficial from a latency perspective due to the very low include to exclude ratio of trained models, as seen in Figure 5 (left). Through exploiting this observation we create the αTM encoding scheme as presented in Figure 6.

The basis of the encoding is centred on identifying the include decisions in the TA states post-training. Once we find to which literal, clause, and class each include state corresponds to in each clause, we sequentially encode this information along with the index of the next class to indicate where includes for a particular class end. This is shown through the example in Figure 6 along with a deconstruction of the TM such that each TA can be tracked back to its corresponding class. Notice that each feature has two literals, the feature itself and its negated form, and each literal has its own TA state. This is represented in Figure 6 by the fact that each feature in the `Features` array stretches across 2 TA states.

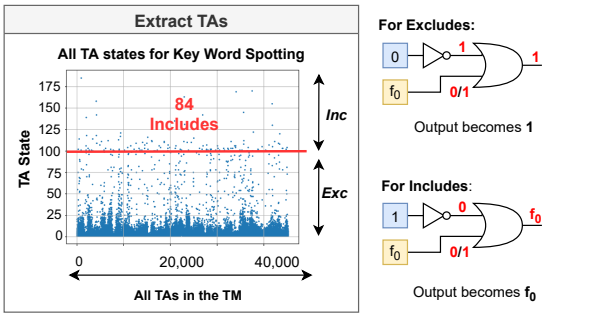

Figure 5: Rationale for the encoding methods. Heavy imbalance between include and exclude states is exploited in μTM to reduce the model size (left). Redundancy of computation for exclude states is used in αTM to reduce inference latency (right).

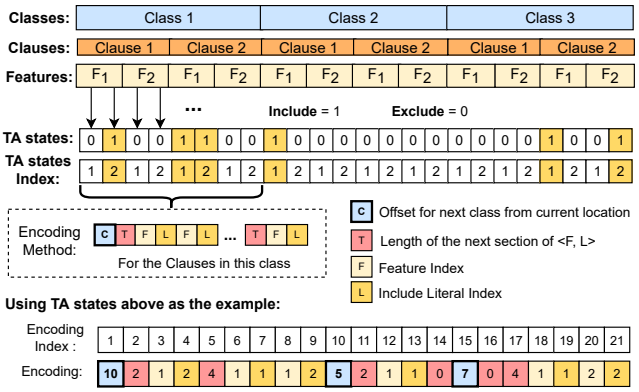

Figure 6: Accelerated TM (αTM) encoding method.

The αTM, encoding starts with `C` indicating the offset of the next class from the current location of the encoded array, observe that in Figure 6 the encoded array starts with 10, as the next class is 10 elements along. Then `T` is used to signify the number of feature (`F`) and corresponding literal (`L`) indexes present in each clause. For example the first `T` element in the encoded array is 2, this indicates there is only one feature and corresponding literal index in this clause. Notice that in the case where there are no includes within a clause this is indicated with a 0 in the `T` element. At runtime, we then iterate through only the includes present in each literal when performing the clause computations, hence, effectively skipping computations and reducing latency.

Similar to μTM, this encoding scheme is lossless since we can perfectly reconstruct the model after it has been encoded. Hence, also αTM ensures the accuracy of the model is not altered after its deployment on constrained devices.

6 POWER-AWARE ADAPTIVE TM

In the previous sections, we have described our encoding approaches to reduce the memory footprint and latency of TM models. In this section, we introduce a complementary technique to adapt the model complexity at runtime. Such technique can be applied to both μTM and αTM models.

The core learning and computational element of the TM is the clause proposition. The evaluation of clauses at runtime allows each class in the TM to determine if a particular datapoint intersects

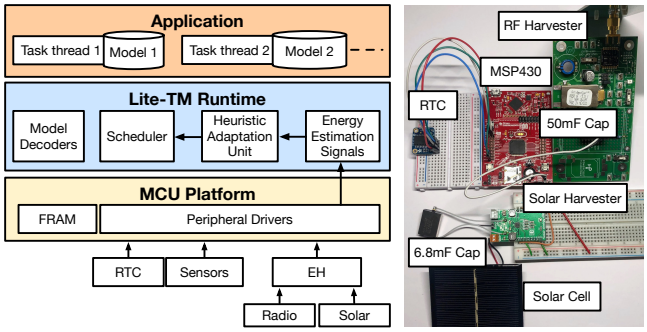

Figure 7: Software components (left) and hardware setup (right).

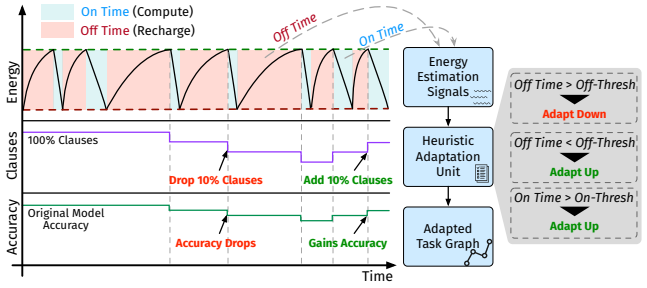

Figure 8: Adaptation policy runtime operation.

with it or not through the clause polarities used in the class sum. Affecting the clauses (i.e., their number or the latency to compute each clause) is the main approach to influencing the latency of the entire model. In our adaptation approach, we do this by adding weights to each clause proposition. Clause weighting is achieved by running the TM's inference process on the training data again (using clause outputs for computing class sums and then selecting the highest class sum for classification) after the training is completed. The the involvement of each clause can be tallied based on the number of times it produces a clause output of 1 across all datapoints in the training dataset as shown mentioned in 3. We then rank the clauses in each class in descending order based on their weights. This creates the ordered TA states model as seen in Figure 3. Through this ranking, the most useful or 'impactful' clause propositions are placed first.

The benefit of this clause ranking is seen at runtime; based on the available energy the framework can evaluate only a specified number of clauses in each class, given the most impactful clauses are evaluated first the accuracy degradation is minimal. We will see in the next section how this enables a single TM model to scale its latency at runtime to cope with fluctuating energy and ensures optimal throughput. Note that the clause ranking itself does not affect model size or latency as it only changes the order of computations at run-time.

7 Lite-TM IMPLEMENTATION

In the previous sections, we have described the three main techniques at the core of Lite-TM (Figure 3). They deal with how the model is trained and encoded, hence, they operate offline, before the model is deployed on the device. This section instead focuses on the software components that run on-device, depicted in Figure 7.

Hardware We have implemented the on-device components of Lite-TM on TI's MSP430FR5994 MCU which has 256KB of embedded FRAM, 8KB of SRAM, and 16 MHz CPU speed. We also connect an external RTC [33] to measure off-time for energy-aware run-time adaptation as described below.

Execution Model: We build Lite-TM on top of a popular intermittent execution model: InK [53]. In InK, applications consist of atomic *tasks*—inside a *task thread*—that can do the 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. The simple architecture of TMs makes task division easy without putting the burden on developers. Leveraging the fact that the compute of the classes is independent from one another, 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/restoring intermediate task buffers.

Input Booleanization: We use well-known and commonly used booleanization methods to process the input to TM models. This is done through either pre-defined functions such as *Adaptive Thresholding*, quantile binning based on the distributions of each feature or simply creating equally spaced fixed thresholds between the maximum and minimum input values [26, 50]. These are computationally simple methods that run efficiently on small microcontrollers. While more sophisticated approaches could lead to better recognition accuracy, their development is beyond the scope of this paper and we leave it for future work.

Model Decoders: Since Lite-TM supports models encoded with two different approaches (μTM and αTM), the on-device runtime needs to support the corresponding decoders. If all the models within the application have been encoded with the same approach, only the corresponding decoder is linked in order to save flash memory on the target platform. Otherwise, both decoders need to be available at runtime.

Adaptation Logic: The adaptation logic of Lite-TM is based on REHASH [3]. The principle behind the adaption is to estimate the amount of energy available from the environment and reduce the model complexity (i.e., drop more clauses) if the energy is scarce, and vice versa, add back the clauses if the energy is abundant. Following the REHASH model, we use the device on-time and off-time as estimates of available energy (i.e., adaptation signals). The on-time represents the time when the device is active and executing code, while the off-time is the interval during which the device is off and the energy storage is recharging. We use the internal MSP430's RTC to measure the on-time while we employ an external RTC for the off-time [33]. Alternative approaches to measure time while the device is off include a remanence-based timekeeper [7, 18].

Figure 8 shows how the adaptation operates at runtime. Two thresholds are configurable for the on-time and off-time, called *On-Threshold* and *Off-Threshold*, respectively. At each reboot of the system, the Heuristic Adaptation Unit will check the off-time and if it is greater than the off-threshold, indicating that less energy is being harvested, it will drop 10% of the clauses. Instead, if the off-time is lower than the threshold it will add clauses back to the model, if any were dropped previously, since more energy is being harvested and the energy storage is recharging faster. Another

Dataset	Network Architecture		
	FC. BNN	Conv. BNN	TM
MNIST	Layer1-512 Layer2-256 (96.97%)	Layer1-10 (95.55%)	200 Clauses 3136000 TAs (95.13%)
CIFAR	Layer1-512 Layer2-256 Layer3-128 (80.91%)	Layer1-10 (85.39%)	1000 Clauses 4096000 TAs (84.58%)
KWS	Layer1-512 Layer2-256 (85.42%)	Layer1-10 Layer2-10 (71.53%)	420 Clauses 2714400 TAs (86.09%)

Table 1: BNN and TM model architectures, and accuracy for the three datasets. LayerX-Y indicates layer number X, and Y neurons for FC models and filters for conv. models, respectively.

important instance when the model can compute more clauses is when the harvested energy is sufficiently high to keep the system running continuously, without interruption. Hence, Lite-TM also uses the on-time to detect such condition and add clauses back into the model. The on-time is checked at the end of each inference.

This is a practical approach, with low overhead, to estimate the current amount of energy available and it is used to take a short-term decision on the next model configuration to run. Long-term duty-cycling approaches [10] are not viable for battery-free systems as energy is scarce and intermittent. Running inferences more or less continuously whenever energy is available is a better alternative to wasting energy on keeping the MCU in a sleep state for long periods (seconds and minutes), which will deplete the energy stored in the capacitor bank due to leakage.

8 EVALUATION

8.1 Experimental Setup

Datasets: We use three datasets for the evaluation:

MNIST [24] 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[23] where we group all the vehicle images into one class and all the animal images into another class. The dataset consists of 60k 32x32 colour images which we convert to grayscale for our evaluation.

Speech Commands (KWS) [49] includes 105k 1-second long utterances of 35 spoken words. We use 6 keyword classes *yes, no, up, down, left, right* for our evaluation.

Following common practices from the machine learning community, we use the MNIST dataset to establish a baseline for our evaluation. The CIFAR-2 and Speech Commands datasets instead, represent image and audio recognition tasks that could potentially be achieved by low-power devices as the first classification stage before triggering a more powerful system.

Models: Through hyperparameter search, we train three TM models, as reported in Table 1, which we then use for the evaluation of Lite-TM. Several previous works showed the feasibility and benefits of using DNNs on intermittently-powered systems [12, 20, 25, 38]. The high accuracy and the fact that, in most cases, there is no need for feature engineering as in shallow classifiers, made DNNs the preferred choice for complex tasks, such as image and audio classification, becoming the *de facto* standard for these tasks. Hence, to appropriately compare Lite-TM with state-of-the-art recognition systems we use deep neural networks as baselines. 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. BNNs are the most energy/memory efficient class of DNNs and given their binary nature are amenable for custom HW implementation [40, 47]. These characteristics, in addition to their high recognition accuracy, make BNNs a good baseline.

For our evaluation, we use the optimised implementation and models provided by McDanel et al. [34]. To date, this is the only open source method of porting BNNs to resource constrained devices. This implementation minimises the memory footprint of temporary results; in a standard BNN the memory size of temporaries generated is equal to the output dimensions of the largest layer which are stored in floating point form, McDanel et al's implementation reorders operations such that the size of temporaries can be stored in binary. As described in § 2.4, the limited size of temporary buffers is also a characteristic of TMs, making the comparison between the two types of models fairer.

Given the difficulty in splitting DNN workloads into tasks, we use task-tiling for the BNNs as done by Gobieski et al. [12]. 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 the 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 and TMs 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 TM models optimised with our framework against vanilla TMs, RLE-TMs and the BNN baselines. We use the term *vanilla TM* to refer to the standard TM inference algorithm as seen in [13] and *RLE-TM* for models encoded with simple run-length encoding [2].

We offer evaluation with continuous and intermittent power from solar and radio-based harvesters. The performance of our adaptation technique is evaluated using the number of inferences completed during a fixed interval and the drop in accuracy compared to static models when the device is powered by variable energy traces. For the on-device evaluation, we use the hardware setup described in §7 and all the metrics (e.g. memory footprint, latency and energy) are measured after models have been deployed on the target MCU.

8.2 Memory Usage

Model Size: Our first evaluation results focus on comparing our μTM and αTM encoding methods against vanilla TM models. Since the number of clauses is one of the main hyperparameters for TMs, we train 10 models with an increasing number of clauses. This allows us to study how the performance of the models changes in relation to their recognition capacity. Figure 9 along with Table 2

Dataset	Encoding Scheme	Compression Ratio with Different Clauses (C) %				
		0.2C	0.4C	0.6C	0.8C	1.0C
MNIST	α	71.40	93.00	95.50	96.20	96.30
	μ	90.40	96.70	97.60	97.80	97.90
CIFAR	α	97.18	98.14	98.49	98.65	98.69
	μ	98.37	98.63	98.78	98.89	98.90
KWS	α	91.71	93.60	94.91	95.67	96.05
	μ	95.87	96.67	97.24	97.58	97.74

Table 2: Compression ratio of model size of μTM and αTM compared to vanilla TMs for different clauses. C is different for each dataset.

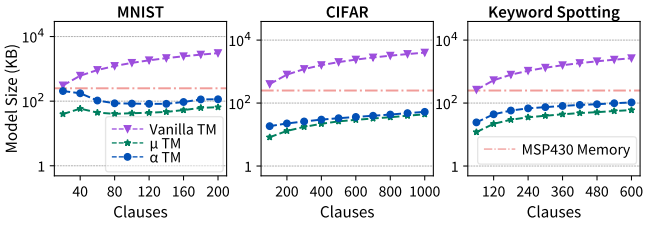

Figure 9: Model size of vanilla TMs compared with μTMs and αTMs (Log scale on y-axis).

show the advantage of the two encoding methods on the model size when compared to vanilla TM. Both encoding methods allow the TM to scale to a larger number of clauses for the three datasets without exceeding the MSP430 memory. The μTM offers the best compression compared to vanilla TM across all three datasets with αTM always slightly higher but in the same range. For example, for 1000 clauses per class for CIFAR-2, μTM offers 98.90% compression while αTM offers 98.69%.

For MNIST we notice that with a lower number of clauses (<80) the size of models compressed with μTM and αTM is higher than models with more clauses. This is due to the include to exclude ratio. There is a larger proportion of includes to excludes when the clauses are few and the problem is well defined. For models with a small number of clauses, the key include literals are concentrated close to each other, hence resulting in a lower compression rate. Instead, when there is a greater number of clauses, the key include literals are spread more evenly throughout the clauses and our encoding schemes can achieve greater compression. This is visible for MNIST which is a simpler dataset compared to the other two, and where the TM can very easily pick out the key include literals already with few clauses. Note that for more difficult classification problems like CIFAR-2 and KWS the TM is unable to pick out as many include literals. After sufficient clauses have been added to the learning problem we notice that the number of includes starts to saturate, this is seen across all three datasets where the model sizes start to increase less rapidly for a larger number of clauses.

From Table 3, we also observe that μTM compression is always better than run-length encoding as done in [2], with a model size reduction of up to 41% for MNIST. On the contrary, the model size of αTM is slightly higher than RLE models, but not excessively big to prevent models to fit in the available memory. We will see later in this section that even with lower compression ratio, αTM is the most energy efficient of all thanks to its very low latency.

Memory Footprint: Figure 10 compares the overall memory footprints of the encoded TMs and Vanilla TM with the baseline BNNs. This includes: *.text* (code), *.const* (model), and *.persistent* (non-volatile buffer and runtime management) sections of the memory. Table 1 lists the BNN and TM models we use for comparison.

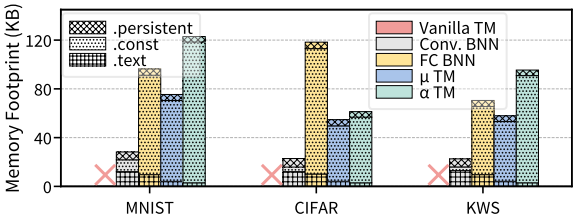

Figure 10: Memory footprint of αTM and μTM compared to vanilla TM and BNNs. Vanilla TMs do not fit in the memory.

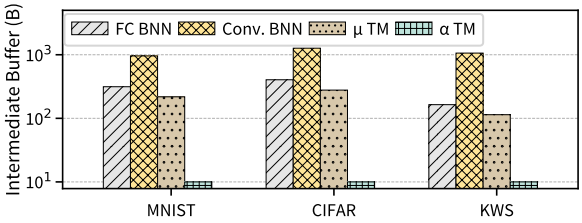

Figure 11: Size of the intermediate buffers of the μTM and αTM models compared to the baselines BNNs (Log scale on y-axis).

We first notice the effectiveness of both encoding methods compared to vanilla TM, which is too large for the MCU. The μTM always offers a lower memory footprint compared to FC BNN across all datasets. The intrinsic properties of the Conv. BNN makes it more memory efficient but, as we will see in the next section, it performs poorly in terms of inference latency; instead a strong point of both μTM and αTM. Notice that the memory footprint for αTM is higher for both MNIST and KWS but almost the same as μTM for CIFAR. This highlights the fundamental differences in the two encoding approaches; μTM exploits the long-running patterns in the TA states while αTM works best with high TA include sparsity. In the case of CIFAR we have the best of both conditions, there are long runs of 0s interlaced with occasional 1s (includes), as such, both methods benefit and offer very similar compression.

Focusing on Table 1, which shows the number of TA states in each TM for the three datasets, we notice that even if the TA states were stored with 1-bit representation (an obvious solution) the sizes would be 383KB, 500KB, 232KB for MNIST, CIFAR, and KWS, respectively. However, our encoding approaches drastically reduce this, for the μTM, the same models can be represented in 66.6KB, 45KB, and 49.2KB, respectively.

Intermediate Buffers: These buffers either contain the results of computations that are not fully completed or, for BNNs, keep a copy of the output of one layer that is to be passed to the next layer as input, in case a power failure occurs. Through Figure 11 we make the case for TMs being good candidates for intermittently-powered systems where large intermediate buffers result in overhead during frequent reboots. For the αTM, the buffer contains only individual class sums. Whereas, μTM requires some other encoding information to be stored as well. The buffer size for αTM is significantly smaller compared to the intermediate results of the BNNs across all three datasets with only 10 bytes. The μTM has a similar but lower size to FC BNNs for all three datasets, e.g. 278 bytes for μTM on CIFAR2 compared to 404 bytes for FC BNN (using the configurations in Table 1). While the Conv. BNN offers the best memory footprint in terms of overall memory usage, we now see this comes

Dataset	Model Size (KB)			Latency (ms)			Energy (mJ)		
	RLE-TM	αTM	μTM	RLE-TM	αTM	μTM	RLE-TM	αTM	μTM
MNIST	111	114	65	4450	168	3310	33.13	1.22	24.99
CIFAR	49	53	44	4448	81	3800	32.75	0.59	28.88
KWS	81	87	48	2650	167	2134	19.45	0.97	10.7

Table 3: αTM and μTM are compared with simple run-length-encoded TM models (RLE-TM) [2]. Comparisons are made using the TM models listed in Table 1.

with a trade-off in intermediate buffer size because the results of the convolution operations must be stored in this buffer.

This is where the αTM demonstrates its clear advantage through the simplicity of the clause operation combined with the shortcut shown in Figure 5 where we show that only the include TA states need to be encoded and evaluated. The focus on evaluating only the include decisions for the clause output means that there are significantly fewer computations required for evaluating the clause output compared to μTM, resulting in simpler encoding and thus smaller intermediate results.

8.3 Constant Power Evaluation

In this section, we evaluate the performance of the techniques offered by Lite-TM when the TM models are deployed on a continuously-powered device.

Accuracy, Latency and Energy: Figure 12 compares the αTM and μTM with fully connected and convolutional BNN equivalents for the three datasets. We use the same 10 models as before, trained with different number of clauses to study how the performance metrics change in accordance with this hyperparameter.

We first notice that the vanilla TM, μTM, and αTM models have the same accuracy since our encoding techniques are lossless and the entire model can be reconstructed exactly at run-time. With a sufficient number of clauses, the TM models reach a very similar accuracy compared to the baselines BNNs or slightly higher for the KWS dataset. In particular, the TM accuracy is within 2% of the BNNs on MNIST, is 4% higher than FC and within 1% of Conv. on the CIFAR dataset, and 14% higher than Conv. on the KWS dataset while being within 1% of the FC accuracy.

When considering latency and energy per inference, the vanilla TM models cannot be deployed on the MSP430 due to the excessive size of the models. Instead, the energy efficiency of the αTM models is highlighted in Figure 12; across all three datasets, the αTM has the lowest latency and therefore energy expenditure compared to both BNNs and μTM models. μTM models have latency that is significantly better than Conv. BNNs and in the same range as FC BNNs, however, as we have seen in the previous section, offer an interesting tradeoff in terms of memory usage. The most significant finding from this figure is that the encoded TMs can provide equal or better accuracy to BNNs for a much lower energy cost.

We also compare the latency and energy of αTM and μTM with simple run-length-encoded TM models [2]. From Table 3, we notice that both αTM and μTM are more energy efficient than RLE TMs. In case of MNIST, RLE takes 1.34x more energy than μTM. Similarly, in case of CIFAR, RLE takes 55x more energy than αTM. These improvements show the benefit of our novel encoding schemes.

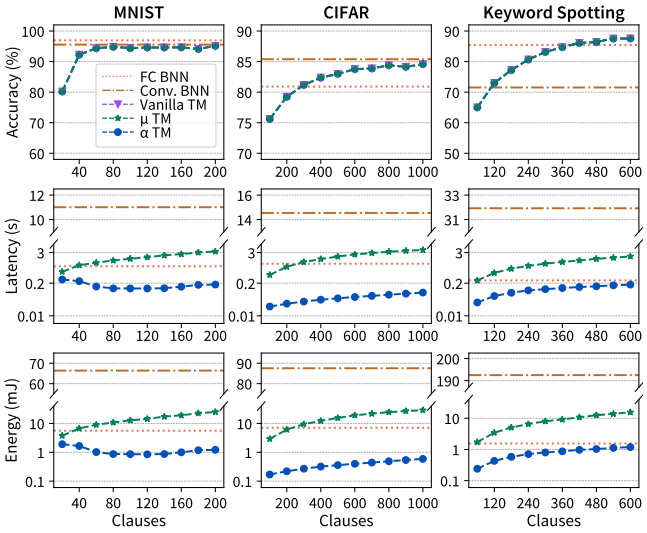

Figure 12: Accuracy, latency, and energy of TM models when compared to baselines BNNs. Vanilla TM exceeded the limited memory of MSP430 for the number of clauses used in the evaluation. (Note broken y-axes in latency/energy plots. Also, note the log scale on y-axes in the lower part of latency/energy plots).

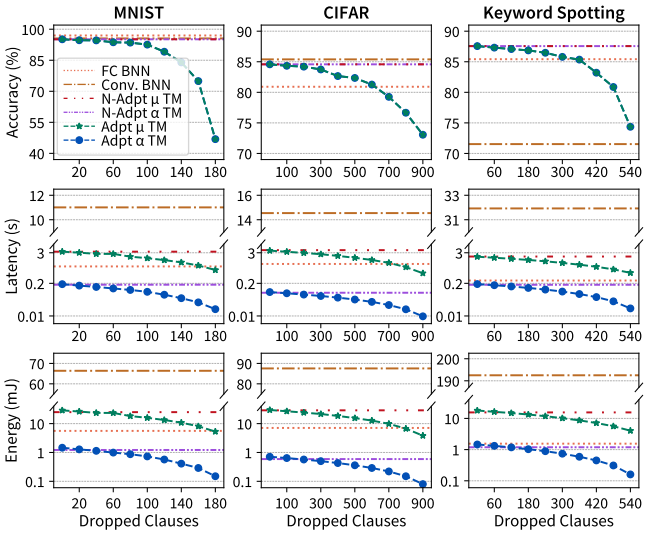

Figure 13: Accuracy, latency, and energy of the encoded TM models when the clause dropping adaptation is applied. (Note broken y-axes in latency/energy plots. Also, note the log scale on y-axes in the lower part of latency/energy plots).

Adaptive Inference: To study the performance of our adaptation technique, we use the TM models which achieved the best accuracy, as reported in Table 1. We then iteratively drop 10% of the clauses until we reach 90% of dropped clauses. This allows us to study the behaviour of the models at different levels of dropped clauses. Figure 13 shows the results in terms of accuracy, latency, and energy per inference. The robustness of the ranked clauses is made clear through the relatively slow accuracy degradation across the three datasets. It is only when nearly half the total number of clauses are dropped that the accuracy starts to reduce more significantly. This leaves a large window of clause-dropping options for the scheduler

when the power is not constant. We also note that for MNIST the performance drop is a lot more extreme compared to CIFAR and KWS. This suggests that the clause weights for MNIST are quite similar and therefore the loss of these clauses has a greater impact. However, for CIFAR and Keyword Spotting there are fewer impactful clauses (the weightings are more extreme), therefore the accuracy degradation is not as harsh.

In terms of latency and energy, Figure 13 shows dropping clauses results in a smooth and linear reduction in inference latency and, consequently, energy. This enables great flexibility at run-time in selecting the appropriate number of clauses to drop to cope with variable energy provided by the harvesters.

8.4 Intermittent Power Evaluation

We use solar and radio energy harvesters to evaluate the performance of Lite-TM under intermittent and variable power scenarios.

Energy Traces. In order to have reproducible evaluation under intermittent power we recorded energy traces using different harvesters and then used those traces with different conditions (e.g., TM models, encoding, adaptive/non-adaptive configuration) to thoroughly evaluate Lite-TM. We use two kinds of energy harvesters for our evaluation; a radio-frequency harvester (Powercast TX91501B, 915 MHz, 3W transmitter) combined with the Powercast P2110B controller and a solar energy management chip (TI BQ25570) paired with a solar cell. The hardware setup is shown in Figure 7.

To record these energy traces, we placed the harvesters in an indoor office space; the RF transmitter and receiver were positioned at a distance of 2m from one another in a breakout area where people often dwell, and the solar harvester was placed on a desk close to a window such that people could shade the solar panel when passing by or stopping next to the desk. We attached to each harvester an MSP430 running our Lite-TM framework to act as load and recorded the time intervals when the MSP430 was on, powered by the harvester, and when it was off, during the recharge of the capacitor. We used a 6.8mF supercapacitor for the solar harvester and a 50mF supercapacitor for the RF harvester and recorded 4-minute long traces for both configurations. We ensured that for each trace there were people in-between the RF transmitter and receiver or they were shading the solar panel to obtain traces with variable harvested energy.

The recorded traces are then loaded on a separate MCU (a Teensy 3.6), acting as an energy emulator, which controls the input supply to the MSP430FR5994, with the actual Lite-TM benchmarks running, accordingly to the on and off intervals. The use of pre-recorded traces allows repeatability in our intermittence evaluation. Ekho [8] can also be used for recording and repeatably emulating power traces, but it is more useful in scenarios where the load is continuously changing, i.e., when an external sensor or radio module is turned on/off. In our case, however, the load is only the MCU which is always on whenever energy is available. Therefore both methods would have resulted in the same recorded and emulated trace. We opted for the former approach since it is simpler to implement and easier to reproduce. A similar setup has been used for evaluating other intermittently-powered systems [38, 53]. The characteristics of the traces we recorded and emulated are listed in Table 4. In order to keep testing conditions consistent, both adaptive and non-adaptive runs of μTM and αTM are powered using the same RF and solar energy traces. It is also important to note that, for all adaptive runs, the on/off periods of RF and solar traces are not affected in any way. It is only the amount of compute that changes at run-time.

Harvester	On Time (ms)			Off Times (ms)			Power
	Max	Min	Avg	Max	Min	Avg	Outages
RF	2473	147	1872	51947	890	10626	20
Solar	76055	1308	6093	19302	1669	5727	33

Table 4: Maximum, minimum and average on/off times, and number of power outages recorded using RF and solar energy harvesters.

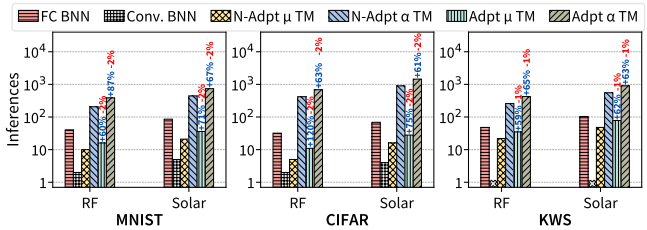

Figure 14: The number of inferences completed under intermittent power with RF and solar harvesters. The blue and red numbers indicate the increase in the number of inferences and the decrease in accuracy, respectively, of the adaptive TM models against the non-adaptive TM models. Log scale on the y-axis.

Baselines and Metrics. As baselines, we use the same BNNs adopted throughout our evaluation and non-adaptive versions of the TM models encoded with μTM and αTM (Table 1). All baselines have a fixed latency and cannot adapt their computation at runtime.

For the evaluation metrics, we use the total number of inferences computed and the estimated accuracy obtained by the models. Since it is impossible to acquire reliable ground truth while running the system intermittently, the models' accuracy is estimated by considering their performance on the test set. For the static models, this corresponds directly to the test set accuracy. For the adaptive models, we compute a weighted average accuracy based on the number of inferences performed for each configuration (i.e., number of clauses used for an inference) and their accuracy on the test set.

Results. Figure 14 reports the number of inferences completed by static and adaptive models. We notice how the adaptive αTM models achieve the highest number of inferences, across all datasets and for both energy traces, thanks to their low inference latency and to the adaptation at runtime. Analysing the results more carefully, we observe that adaptive αTM models complete between 61% and 87% more inferences compared to non-adaptive αTM models (blue annotations in Figure 14) with a drop in accuracy between 1% and 2% (red annotations). A similar increase in inference throughput, with 120% more inferences at max, and a drop in accuracy is also shown for adaptive μTM models compared with the respective non-adaptive versions. This demonstrates that regardless of which encoding is used (μTM for memory footprint or αTM for latency) our adaptation increases inference throughput with minimal accuracy drop when running intermittently.

Table 5 shows the latency and energy expenditure per inference for our approaches with and without adaptation compared with BNNs. Through these experiments, we make a strong case for using

Dataset	Energy Trace	Latency (s) / Energy (mJ)					
		FC. BNN	Conv. BNN	Non-Adpt. μ TM	Non-Adpt. α TM	Adpt. μ TM	Adpt. α TM
MNIST	RF	6.00 / 8.73	120.0 / 174.6	24.00 / 34.92	1.17 / 1.70	15.0 / 21.83	0.62 / 0.90
	Solar	2.79 / 8.12	48.00 / 139.7	11.43 / 33.26	0.54 / 1.58	6.67 / 19.40	0.33 / 0.95
CIFAR	RF	7.50 / 10.9	120.0 / 174.6	48.00 / 69.84	0.57 / 0.83	21.8 / 31.75	0.35 / 0.51
	Solar	3.48 / 10.1	60.00 / 174.6	15.00 / 43.65	0.27 / 0.78	8.57 / 24.94	0.17 / 0.48
KWS	RF	5.00 / 7.28	✖ / ✖	10.91 / 15.87	0.93 / 1.35	6.86 / 9.98	0.56 / 0.82
	Solar	2.31 / 3.36	✖ / ✖	5.00 / 14.55	0.43 / 1.27	3.08 / 8.95	0.27 / 0.78

Table 5: Latency and energy per inference for BNNs and non-adaptive TMs compared with adaptive TMs on different energy traces.

αTM for intermittently powered systems. Once again we see that even without adaptation the αTM provides better latency and energy efficiency compared to BNNs and this is further improved with the run-time adaptation. The experiments highlight the computational intensity of the Conv. BNN: the on-time is not sufficient to complete any inferences for KWS. The μTM offers a middle ground between FC BNN and Conv. BNN and for a relative sacrifice in energy frugality we have the potential to scale to larger problems.

9 DISCUSSION

On Flexibility: Aside from the first automated transition of Tsetlin Machines from software to optimised MCU implementations complete with runtime adaptation, the main advantage of Lite-TM is the flexibility to different applications. The user has the design knobs to determine whether memory is more of a priority than latency and if runtime adaption is necessary or not. As classification tasks become more complex and the number of features increases, so will the memory footprint of the models. To achieve good performance for these applications more clauses will be required to derive the necessary propositions. In cases like this, the μTM is better suited. For applications where real-time response is important the αTM is preferred since it is better suited to smaller problems where the memory footprint will be small but low inference latency is paramount. A substantial advantage of Lite-TM is that the accuracy of the models is preserved since both encoding methods are lossless.

Similarly, adapting the runtime complexity of the model might be preferable in applications where energy is not constant, as considered in this paper. However, the flexibility of Lite-TM allows to easily adopt the same approach in other settings. For example, to reduce the inference latency when a battery is draining (in a battery-powered device) or when the user is demanding high resource utilisation from other applications (i.e., gaming) and inference complexity needs to be scaled down.

On Scalability: The purpose of this work is to explore an emerging machine learning algorithm and optimise its inference efficiency for intermittently-powered devices. The choice of recognition tasks reflected this target execution environment. This however, does not allow us to make any claim on the scalability performance of TM models to much larger and complex tasks (e.g., cloud-scale image or acoustic recognition). For such problems, TMs are at a very early stage compared to artificial neural networks, which enjoyed many years of research and development. Recently, a community is building around the understanding of the limits of TMs and their comparison with deep learning approaches [5, 11, 27, 41, 46, 51], however, this is beyond the scope of this paper.

The algorithmic and architectural disparity between TMs and DNNs make fair comparisons very challenging. Nevertheless, in this paper we compared TMs with the closed inference implementation possible (i.e., binary neural networks) and offered a balanced look at the advantages and disadvantages of our proposed optimisations against them. As shown in Figure 12 the maximum accuracy reached by the encoded TMs is very close to the highest accuracy achieved by BNNs (either fully connected or convolutional) for all the datasets we considered. This shows that, for what accuracy is concerned, TMs do not offer a better alternative to BNNs yet. However, in terms of latency and energy consumption, our encoding schemes (αTM in particular) offer significant benefits compared to BNNs especially when considering intermittently-powered and embedded devices. This is thanks to the very simple architecture of the clause computation unit (Figure 1) and the fact that TM models do not have multi-layer structures which complicate their inference execution. We believe our work can be easily applied to future iterations of the TM algorithm since they operate after the model has been trained. This could open the doors to more efficient TMs for large and complex recognition tasks.

On Encoding and Training: Both encoding methods rely on the imbalance of TA include decisions to exclude decisions; through the experiments, we have shown that this can be incredibly useful in reducing the memory size as well as reducing the inference latency when the number of includes is low. Through the weighting and ranking of clauses during training combined with the runtime adaptation we also examined the impact of dropping less impactful clauses from the classification. In doing so we show that some includes are more important in the classification than others. Future work will look deeper into exploiting this to develop an "include-aware" training procedure that can lead to even smaller memory footprints and inference latency and open opportunities for exploring larger classification problems.

10 CONCLUSION

In this paper, we have explored the practical possibilities of Tsetlin Machines in adaptive batteryless systems and experimentally shown their advantages compared to binarized neural networks in terms of energy efficiency. We developed Lite-TM, a framework for automating the deployment of Tsetlin Machines on resource-constrained microcontrollers through encoding techniques that offer the user the trade-off between memory footprint and energy efficiency, as well as runtime adaptability. Lite-TM has been evaluated with vision and acoustic workloads demonstrating the efficacy of our principled approaches in improving the memory, latency, and energy efficiency of TMs while also maximising the overall system availability. We believe this work represents a useful first step towards the exploration of an alternative classification algorithm to deep neural networks, not only in the context of batteryless systems but also in the larger area of embedded machine learning.

REFERENCES

[1] Antonini, M., Vu, T. H., Min, C., Montanari, A., Mathur, A., and Kawsar, F. Resource characterisation of personal-scale sensing models on edge accelerators. In *Proceedings of the First International Workshop on Challenges in Artificial Intelligence and Machine Learning for Internet of Things* (2019), AIChallengeIoT'19.

[2] Bakar, A., Rahman, T., Montanari, A., Lei, J., Shafik, R., and Kawsar, F. Logic-based intelligence for batteryless sensors. In *Proceedings of the 23rd Annual International Workshop on Mobile Computing Systems and Applications* (2022), pp. 22–28.

[3] Bakar, A., Ross, A. G., Yildirim, K. S., and Hester, J. 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.

[4] Balsamo, D., Weddell, A. S., Merrett, G. V., Al-Hashimi, B. M., Brunelli, D., and Benini, L. Hibernus: Sustaining computation during intermittent supply for energy-harvesting systems. *IEEE Embedded Systems Letters 7*, 1 (2014), 15–18.

[5] Berge, G. T., Granmo, O.-C., Tveit, T. O., Goodwin, M., Jiao, L., and Matheussen, B. V. Using the tsetlin machine to learn human-interpretable rules for high-accuracy text categorization with medical applications, 2018.

[6] Cai, H., Gan, C., Wang, T., Zhang, Z., and Han, S. Once-for-all: Train one network and specialize it for efficient deployment. *arXiv preprint arXiv:1908.09791* (2019).

[7] de Winkel, J., Delle Donne, C., Yildirim, K. S., Pawełczak, P., and Hester, J. Reliable timekeeping for intermittent computing. In *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems* (2020), pp. 53–67.

[8] et al., H. Ekho: Realistic and repeatable experimentation for tiny energy-harvesting sensors. In *Proceedings of the 12th ACM Conference on Embedded Network Sensor Systems* (2014), pp. 330–331.

[9] Fang, B., Zeng, X., and Zhang, M. Nestdnn: Resource-aware multi-tenant on-device deep learning for continuous mobile vision. In *Proceedings of the 24th Annual International Conference on Mobile Computing and Networking* (2018), pp. 115–127.

[10] Geissdoerfer, K., Jurdak, R., Kusy, B., and Zimmerling, M. Getting more out of energy-harvesting systems: Energy management under time-varying utility with preact. In *Proceedings of the 18th International Conference on Information Processing in Sensor Networks* (2019), pp. 109–120.

[11] Glimsdal, S., and Granmo, O.-C. Coalesced multi-output tsetlin machines with clause sharing, 2021.

[12] Gobieski, G., Lucia, B., and Beckmann, N. 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* (2019), pp. 199–213.

[13] Granmo, O.-C. The tsetlin machine–a game theoretic bandit driven approach to optimal pattern recognition with propositional logic. *arXiv preprint arXiv:1804.01508* (2018).

[14] Granmo, O.-C., Glimsdal, S., Jiao, L., Goodwin, M., Omlin, C. W., and Berge, G. T. The convolutional tsetlin machine, 2019.

[15] Han, S., Mao, H., and Dally, W. J. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149* (2015).

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

[17] Hester, J., Storer, K., and Sorber, J. Timely execution on intermittently powered batteryless sensors. In *Proceedings of the 15th ACM Conference on Embedded Network Sensor Systems* (2017), pp. 1–13.

[18] Hester, J., Tobias, N., Rahmati, A., Sitanayah, L., Holcomb, D., Fu, K., Burleson, W. P., and Sorber, J. Persistent clocks for batteryless sensing devices. *ACM Transactions on Embedded Computing Systems (TECS) 15*, 4 (2016), 1–28.

[19] Instruments, T. MSP430FRxx FRAM Microcontrollers. http://www.ti.com/lsds/ti/microcontrollers_16-bit_32-bit/msp/ultra-low_power/msp430frxx_fram/overview.page. Accessed: 10-21-2021.

[20] Islam, B., Luo, Y., and Nirjon, S. Zygarde: Time-sensitive on-device deep intelligence on intermittently-powered systems.

[21] Jacob, B., Kligys, S., Chen, B., Zhu, M., Tang, M., Howard, A., Adam, H., and Kalenichenko, D. Quantization and training of neural networks for efficient integer-arithmetic-only inference. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (2018).

[22] Kim, Y.-D., Park, E., Yoo, S., Choi, T., Yang, L., and Shin, D. Compression of deep convolutional neural networks for fast and low power mobile applications. *arXiv preprint arXiv:1511.06530* (2015).

[23] Krizhevsky, A. Learning multiple layers of features from tiny images.

[24] LeCun, Y. The mnist database of handwritten digits. http://yann.lecun.com/exdb/mnist/, 1998. Accessed: 10-21-2021.

[25] Lee, S., and Nirjon, S. Neuro. zero: a zero-energy neural network accelerator for embedded sensing and inference systems. In *Proceedings of the 17th Conference on Embedded Networked Sensor Systems* (2019), pp. 138–152.

[26] Lei, J., Rahman, T., Shafik, R., Wheeldon, A., Yakovlev, A., Granmo, O.-C., Kawsar, F., and Mathur, A. Low-power audio keyword spotting using tsetlin machines. *Journal of Low Power Electronics and Applications 11*, 2 (2021), 18.

[27] Lei, J., Wheeldon, A., Shafik, R., Yakovlev, A., and Granmo, O.-C. From arithmetic to logic based ai: A comparative analysis of neural networks and tsetlin machine. In *2020 27th IEEE International Conference on Electronics, Circuits and Systems (ICECS)* (2020), pp. 1–4.

[28] Lin, J., Chen, W.-M., Cai, H., Gan, C., and Han, S. Mcunetv2: Memory-efficient patch-based inference for tiny deep learning. In *Annual Conference on Neural Information Processing Systems (NeurIPS)* (2021).

[29] Lin, J., Chen, W.-M., Lin, Y., Gan, C., Han, S., et al. MCUNet: Tiny deep learning on iot devices. *Advances in Neural Information Processing Systems 33* (2020), 11711–11722.

[30] Liu, Z., Sun, M., Zhou, T., Huang, G., and Darrell, T. Rethinking the value of network pruning. *arXiv preprint arXiv:1810.05270* (2018).

[31] Lucia, B., Balaji, V., Colin, A., Maeng, K., and Ruppel, E. Intermittent computing: Challenges and opportunities. In *2nd Summit on Advances in Programming Languages (SNAPL 2017)* (2017), Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik.

[32] Maeng, K., Colin, A., and Lucia, B. Alpaca: Intermittent execution without checkpoints. *Proceedings of the ACM on Programming Languages 1*, OOPSLA (2017), 1–30.

[33] Maxim Integrated. DS3231 real time clock (rtc). https://datasheets.maximintegrated.com/en/ds/DS3231.pdf, Mar. 2008. Last accessed: Dec. 20, 2021.

[34] McDanel, B., Teerapittayanon, S., and Kung, H. Embedded binarized neural networks. *arXiv preprint arXiv:1709.02260* (2017).

[35] Min, C., Mathur, A., Montanari, A., and Kawsar, F. Sensix: A system for best-effort inference of machine learning models in multi-device environments. *IEEE Transactions on Mobile Computing* (2022).

[36] Min, C., Montanari, A., Mathur, A., and Kawsar, F. A closer look at quality-aware runtime assessment of sensing models in multi-device environments. In *Proceedings of the 17th Conference on Embedded Networked Sensor Systems* (2019).

[37] Montanari, A., Alloulah, M., and Kawsar, F. Degradable inference for energy autonomous vision applications. In *Adjunct Proceedings of the 2019 ACM International Joint Conference on Pervasive and Ubiquitous Computing* (2019), UbiComp/ISWC '19.

[38] Montanari, A., Sharma, M., Jenkus, D., Alloulah, M., Qendro, L., and Kawsar, F. ePerceptive: Energy Reactive Embedded Intelligence for Batteryless Sensors. In *Proceedings of the 18th Conference on Embedded Networked Sensor Systems* (2020), pp. 382–394.

[39] Nardello, M., Desai, H., Brunelli, D., and Lucia, B. Camaroptera: a batteryless long-range remote visual sensing system. In *Proceedings of the 7th International Workshop on Energy Harvesting & Energy-Neutral Sensing Systems* (2019), pp. 8–14.

[40] Qin, H., Gong, R., Liu, X., Bai, X., Song, J., and Sebe, N. Binary neural networks: A survey. *Pattern Recognition 105* (2020), 107281.

[41] Rahman, T., Shafik, R., Granmo, O.-C., and Yakovlev, A. Resilient biomedical systems design under noise using logic-based machine learning. *Frontiers in Control Engineering 2* (2022).

[42] Ransford, B., Sorber, J., and Fu, K. Mementos: System support for long-running computation on rfid-scale devices. In *Proceedings of the sixteenth international conference on Architectural support for programming languages and operating systems* (2011), pp. 159–170.

[43] Robinson, A. H., and Cherry, C. Results of a prototype television bandwidth compression scheme. *Proceedings of the IEEE 55*, 3 (1967), 356–364.

[44] Rosenblatt, F. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review 65*, 6 (1958), 386.

[45] Saha, R., Granmo, O.-C., and Goodwin, M. Using tsetlin machine to discover interpretable rules in natural language processing applications. *Expert Systems n/a*, n/a, e12873.

[46] Sharma, J., Yadav, R., Granmo, O.-C., and Jiao, L. Drop clause: Enhancing performance, interpretability and robustness of the tsetlin machine.

[47] Simons, T., and Lee, D.-J. A review of binarized neural networks. *Electronics 8*, 6 (2019), 661.

[48] Sparks, P. The route to a trillion devices. White Paper, ARM, 2017.

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

[50] Wheeldon, A., Shafik, R., Rahman, T., Lei, J., Yakovlev, A., and Granmo, O.-C. Learning automata based energy-efficient ai hardware design for iot applications. *Philosophical Transactions of the Royal Society A 378*, 2182 (2020), 20190593.

[51] Yadav, R. K., Jiao, L., Granmo, O.-C., and Olsen, M. G. Interpretability in word sense disambiguation using tsetlin machine. In *ICAART* (2021).

[52] Yang, T.-J., Howard, A., Chen, B., Zhang, X., Go, A., Sandler, M., Sze, V., and Adam, H. Netadapt: Platform-aware neural network adaptation for mobile applications. In *Proceedings of the European Conference on Computer Vision (ECCV)* (2018), pp. 285–300.

[53] Yildirim, K. S., Majid, A. Y., Patoukas, D., Schaper, K., Pawelczak, P., and Hester, J. Ink: Reactive kernel for tiny batteryless sensors. In *Proceedings of the 16th ACM Conference on Embedded Networked Sensor Systems* (2018), pp. 41–53.