MOUSE: Inference In Non-volatile Memory for Energy Harvesting Applications

Salonik Resch
resc0059@umn.edu

S. Karen Khatamifard
khatami@umn.edu

Zamshed I. Chowdhury
chowh005@umn.edu

Masoud Zabihi
zabih003@umn.edu

Zhengyang Zhao
zhaox526@umn.edu

Husrev Cilasun
cilas001@umn.edu

Jian-Ping Wang
jpwang@umn.edu

Sachin S. Sapatnekar
sachin@umn.edu

Ulya R. Karpuzcu
ukarpuzc@umn.edu

University of Minnesota, Twin Cities

***Abstract*—There is increasing demand to bring machine learning capabilities to low power devices. By integrating the computational power of machine learning with the deployment capabilities of low power devices, a number of new applications become possible. In some applications, such devices will not even have a battery, and must rely solely on energy harvesting techniques. This puts extreme constraints on the hardware, which must be energy efficient and capable of tolerating interruptions due to power outages. Here, we propose an in-memory machine learning accelerator utilizing non-volatile spintronic memory. The combination of processing-in-memory and non-volatility provides a key advantage in that progress is effectively saved after every operation. This enables instant shut down and restart capabilities with minimal overhead. Additionally, the operations are highly energy efficient leading to low power consumption.**

***Index Terms*—Intermittent computing, Processing-in-Memory**

I. Introduction

Machine learning is desirable for low-power, edge devices as it provides the capability to solve a wide variety of problems. As a result, much research has been devoted to optimizing hardware for machine learning inference on such devices [17], [51]. Going even further, energy harvesting techniques [43] remove the need for a battery, enabling the placement of such devices into almost any conceivable environment. There are many exciting possible applications, such as low power sensor networks [63], wearable tech, or even implants [30]. Previous work has already experimentally demonstrated machine learning capability on energy harvesting devices using commercially available hardware [29].

Energy harvesting applications present numerous and unique challenges. The energy harvested from the environment may be less than what can be supplied by a battery, making energy efficiency even more critical than in mobile applications. Significantly, the process of energy harvesting also introduces the requirement for *intermittent processing*. Energy sources (such as sunlight, heat, movement) may be unreliable, and a device will have to shut down when the power source goes away. Additionally, even when available, the power source may be insufficient to run the device continually. In order to operate within the power budget, the device must acquire energy over time and consume it in bursts [9].

Intermittent processing introduces new considerations and metrics for performance [53]. Significantly, correctness has to be guaranteed over shut down and restart operations. If the state is not properly stored – a process known as checkpointing – restarting a device can lead to memory inconsistencies and incorrect operation [14]. Additionally, the efficiency of these shut down and restart operations becomes critical, as they take away precious energy from operations that enable forward progress. Also critical, it has to be ensured that forward progress can be made during phases of power-on time. If the energy required between two checkpoints is too large, the device will be unable to complete the computation. This results in a program getting stuck, which is referred to as non-termination. Thus, effective energy harvesting devices must have efficient techniques which enable correctness and forward progress, all while remaining within a modest hardware budget.

A recently proposed spintronic processing in memory (PIM) substrate, CRAM [12], is uniquely well suited for energy harvesting applications. Operations on CRAM are highly energy efficient, enabling a low power budget. Further, as it is a PIM solution, it removes the need for energy hungry data transfers between processor logic and (volatile) memories. The main advantage, however, is that progress is automatically saved after every operation. CRAM consists entirely of non-volatile devices and the results of all computation are immediately stored in permanent memory. As there are very few variables required to maintain the architectural state, these can also be saved after each operation with minimal energy cost. Effectively, checkpointing occurs after every operation.

Checkpointing after each operation is not a new idea [59], and for most systems this would generally be considered inefficient [14]. However, as CRAM is a non-volatile PIM substrate and *all* of the computation occurs within the memory array, data backup for checkpointing happens automatically, i.e., non-volatile PIM is always performing data backup. Hence, CRAM can restart a program from the very last operation with fast and efficient shut down and restart. Additionally, CRAM is always in a state that can be recovered from. The power can be cut *instantly and unexpectedly*, and it will still restart correctly. The maximum penalty is repeating the last instruction. We refer to this capability as *instant restartability.* This provides a significant advantage, as shut down and restart procedures for more conventional energy harvesting devices introduce additional latency and energy, and significant complexity.

In this paper, we introduce MOUSE (**M**inimal **O**verhead Accelerator **U**tilizing **S**pintronic RAM for **E**nergy Harvesting Applications) which is built using CRAM [12]. While based on CRAM, MOUSE has a different cell design which reduces energy consumption during computation. For our applications, we implement support vector machines (SVM) and binary neural networks (BNN), which are widely used machine learning algorithms, especially promising in the energy harvesting domain due to their small footprint. We demonstrate how MOUSE can provide high performance and energy efficiency

This work was supported in part by NSF under Grant SPX-1725420

978-1-7281-7383-2/20/$31.00 ©2020 IEEE
DOI 10.1109/MICRO50266.2020.00042

on such applications while also having efficient shut down and restart procedures. Additionally, we consider how another modification to the CRAM cell, the addition of a spin-hall effect (SHE) channel [96], can further increase energy efficiency – by enabling independent optimization of the read and writes, which otherwise come with conflicting requirements.

The contributions of this paper are as follows:

- We demonstrate that logic operations performed with magnetic tunnel junctions (MTJs) are inherently idempotent.
- We utilize this property with processing-in-memory to create an energy-efficient and intermittent-safe machine-learning inference accelerator.

In Section II we provide the working principles of CRAM. In Section III we describe the the support vector machines and binary networks we use as applications. We introduce design specifics of MOUSE in Section IV and show how we guarantee correctness in Section V. A summary is provided in Section VI and an example of application mapping is shown in Section VII. We set up the evaluation in Section VIII, show our results in Section IX, discuss related work in Section X, and conclude in Section XI.

II. Spintronic PIM

Spintronic memory in the form of STT-MRAM is an emerging technology, with a few products already commercially available [1]. Due to its non-volatility, high density, speed, and endurance, STT-MRAM is being considered as a universal memory replacement [22]. STT-MRAM arrays use one magnetic tunnel junction (MTJ) and one access transistor per cell. Being based on CRAM [12], MOUSE maintains the same basic cell structure. By making light modifications to the array, CRAM is able to connect MTJs in such a way to enable logic operations to be implemented within the array. Therefore, MOUSE is capable of being used as both a standard STT-MRAM array and as a computational substrate. CRAM is unique in that the computation does not require any external logic circuits or the use of sense amplifiers, making the computation contained *entirely* within the array. In the following, we explain MTJ basics and show how they can be used in logic operations. Then we demonstrate how these operations can be performed within the array structure.

A. Magnetic Tunnel Junction (MTJ) Basics

STT-MRAM arrays are built with magnetic tunnel junctions (MTJ). The MTJ is a resistive memory device which consists of two magnetic layers (fixed layer and free layer) which are separated by an insulator. The polarity of the free layer can change but the fixed cannot. When the fixed and free layers are aligned, the MTJ is in the parallel (P) state, which has a low resistance and corresponds to logic value 0. When the layers are opposing, the MTJ is in the anti-parallel (AP) state, which has a high resistance and corresponds to logic value 1.

The state can be determined by applying a voltage across the device and sensing the amount of current that travels through it. If a sufficient amount of current is driven through the device, it will change state. *Importantly, the state it changes to depends on the direction of the current*. This is key to our ability to ensure correctness in spite of power outages. When current flows from the free layer (fixed layer) to the fixed layer (free layer), it switches the MTJ to the AP (P) state.

B. Implementing Logic Gates in Memory

Before showing how logic can be implemented in the MOUSE array, we demonstrate how CRAM performs logic gates on MTJs in principle. The configuration for a two-input logic gate is shown in Figure 1. The two MTJs in parallel are the inputs to the logic gate, and the MTJ in series with them is the output. The output must be preset to a known value. For example, the output is preset to 0 (low resistance) for a NAND gate. To implement a NAND gate, a voltage is applied across the two terminals, V_1 and V_2, such that current flows from the input MTJs to the output MTJ. If either of the input MTJs is 0 (low resistance) there will be sufficient current to switch the output MTJ to 1. If both input MTJs are 1 (high resistance), there will be insufficient current to change the state of the output MTJ, and it will remain at 0. Therefore, the state of the output MTJ follows the truth table for a NAND gate, it is 0 only if both inputs are 1. *Most importantly, per basic MTJ physics, current flowing in the supplied direction can only cause the output MTJ to switch to 1; it cannot cause it to switch to 0.*

Many other common gates –including universal ones– can be implemented similarly, such as AND and (N)OR. In order to implement other gates, we can change the number of inputs, the preset value of the output, or the direction of the current.

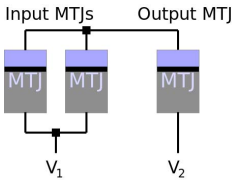


Fig. 1. MTJs connected to implement a 2-input logic gate. The preset value of the output MTJ and the polarity and magnitude of the voltage applied between V_1 and V_2 determines the type of logic gate. The fixed layer is colored in grey and the free layer in light blue.

More complex operations are broken down into these basic logic operations. For example, to perform a full-add in MOUSE, we can perform 9 NAND gates sequentially and use spare MTJs to hold 7 temporary bits. Using full-adds, full-subtracts, and other primitive operations we can perform integer or fixed-point arithmetic, thus enabling us to implement our benchmarks. Naturally, the latency for each complex operation is quite high, as it must be broken down into its constituent gates which are then performed sequentially. However, as we will show in later sections, this can be easily compensated for by performing many data independent operations in parallel, under intermittent power constraints. Due to space limitations, we will focus on MOUSE-specific CRAM adaptations next, but numerous papers [12], [94], [96] cover the details of using MTJs to perform more complex logic based on this basic CRAM principle.

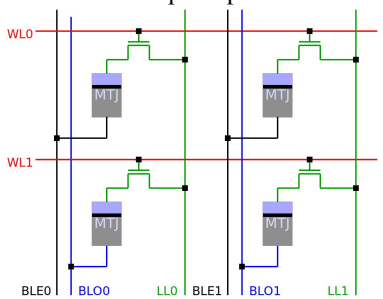


Fig. 2. 4 cells in 2 columns and 2 rows in 1T1M (one access transistor, one MTJ) STT configuration.

C. MOUSE Array Architecture

Based on CRAM, MOUSE essentially is an STT-MRAM array with some additional hardware. As an example, four cells located in adjacent rows and columns are shown in Figure

2. Each memory cell consists of one MTJ and one access transistor. In each column there are two bit lines, bit line even (BLE) and bit line odd (BLO), and a logic line (LL). In each row there is a wordline (WL) that controls the access transistor. Each MTJ is connected to the LL through the access transistor and to one of the two bit lines. Cells in even rows are connected to BLE and cells in odd rows are connected to BLO. We now describe how memory and logic operations are performed in the array.

Memory Operation: To read or write from row n, activate WLn and apply a voltage differential across LL and the bitlines. Current will only travel through the bitline with the same parity as n. Current can be sensed on the bitlines to perform a read, or a large current can be driven through the MTJ to perform a write.

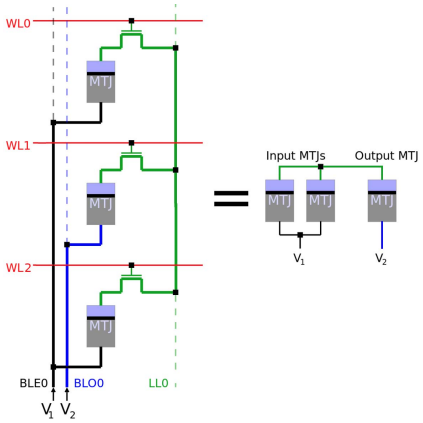


Fig. 3. Demonstration of how a (2-input) NAND gate is performed within the array.

Logic Operation: To perform a logic operation with inputs in rows n_1, n_2 and with output in row m, preset row m by performing a write operation. n_1 and n_2 must have the same parity (i.e., both even or both odd) and m the opposite. Activate WLn_1, WLn_2 and WLm. Apply a voltage differential across BLE and BLO. Due to the parity requirement, in Figure 1, if V_1 is connected to BLE, V_2 must be connected to BLO, and vice versa. In this case, the junction connecting the free layers (in light blue) of the inputs and the ouput corresponds to LL. Current travels from one bit line (be it BLO or BLE, depending on the parity of the input cells), through the MTJs in rows n_1 and n_2, through the LL, through the MTJ in row m, and back to the other bitline. Depending on the states of the MTJs in rows n_1 and n_2, the state of the MTJ in row m will either change or not. As an illustrative example, Figure 3 demonstrates the formation of a NAND gate.

Voltage which drives the operation is applied to every column (over the respective bitlines) in which the specified operation should take place. The peripheral circuitry determines which columns these are, which can be specified by dedicated instructions as will be described in Section IV-B. Hence, *while only one operation can be performed in a column at a time, an operation can be performed in many columns simultaneously*. This gives MOUSE *column level parallelism*, which bears some resemblance to bit-serial architectures.

D. Alternative Memory Cell Architecture

Augmenting each MTJ in the MOUSE cell with a Spin Hall Effect (SHE) channel can further improve energy efficiency. This is the same technology as Spin-Orbit Torque (SOT) MRAM [27], [66], which will likely replace STT-MRAM.

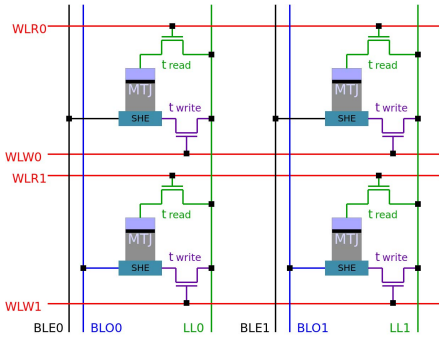


Fig. 4. 4 cells in 2 columns and 2 rows in 2T1M SHE configuration.

The SHE channel provides separate paths for reads and writes (where optimization targets conflict), allowing for separate optimization, thereby better energy-efficiency. SHE channels are CMOS/MTJ-compatible and fabricated prototypes exist [27]. Technology details of SHE integration in CRAM is covered in [96]. Four augmented cells in two rows and two columns are shown in Figure 4.

In this case, there are two word lines per row, word line for read (WLR) and word line for write (WLW). WLR connects the cell to the read path, via t_{read}. WLW connects the cell to the write path, via t_{write}. When t_{write} is activated, current only passes through the SHE channel (and not the respective MTJ). This current, while not affected by the state of the MTJ, can still change its state. This configuration is used when writing to the MTJ or when the MTJ is the target output of a logic operation. When t_{read} is activated, on the other hand, current passes through the SHE channel *and* the MTJ. This allows the MTJ state to affect the current that travels through it. This is used when reading the MTJ state and when the MTJ is used as an input to a logic operation.

The SHE channel has important benefits. Due to the separation of read and write paths, the required current density to induce switching is lower, allowing for a reduction in the energy of write and logic operations. This increased energy efficiency can provide a decrease in the overall execution time in energy harvesting scenarios, as will be shown in our evaluation. Additionally, as the output MTJ resistance no longer is in series with the input MTJ resistances in a logic operation, different input values become easier to distinguish, increasing the robustness of logic operations.

III. Case Studies

To show the capability of MOUSE, we implement Support Vector Machines (SVM) and Binary Neural Networks (BNN). Both are widely used machine learning algorithms. Generally speaking, whether SVMs or neural networks are a superior choice depends on the target problem, but applications overlap considerably and both are applicable in the energy harvesting domain, where both the energy and the area budget is stringent.

SVMs are effective and simple classifiers for typically smaller data sets. Particularly, we found SVMs to perform well on MNIST image recognition and human activity recognition. However, there is a trade-off, as SVMs can struggle with some problems. For example, we were unable to achieve reasonable accuracy on the speech recognition data set, which neural networks have performed well on [29].

For all SVM benchmarks we use a polynomial kernel with a degree of 2. For inference, the main computation is effectively performing the dot product between an input vector and each of the support vectors. The results of these dot products are then squared, multiplied by a set of coefficients, and finally

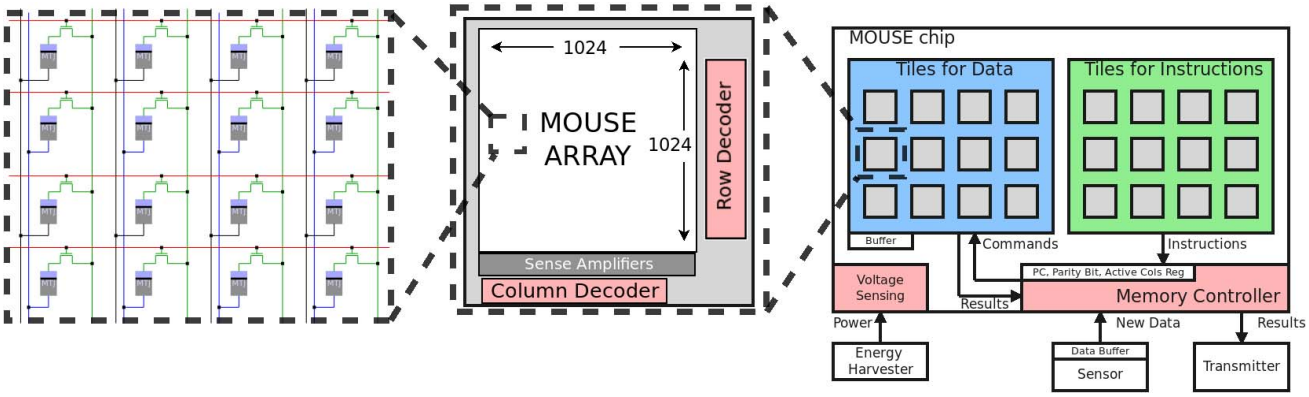


Fig. 5. Overview of MOUSE. Each tile contains an array of MTJs along with a row and column decoder. Sense amplifiers are required for read/writes but aren't used in computation. Shown here is the STT (1T1M) configuration.

summed together. By design, SVMs have two class outputs, where the sign of the output value is the classification.

In this work, we opt for the simplest extension to multi-class problems: we train a separate SVM for each possible output class. Each SVM has the task of identifying its assigned class. For example, MNIST has 10 different classes for digits 0-9. We train 10 SVMs each identifying each digit. The output is 10 scores for "how similar" the input is to each digit. We take the highest-score output of the 10 classifiers to be the final classification. We perform training offline in software and only consider inference acceleration on MOUSE.

BNNs are neural networks that have neurons and weights represented by a single bit each [18]. This enables multiplications to be replaced by XNOR operations and addition is simplified to a popcount operation. As a result, BNNs are much more energy efficient than full- or fixed-precision networks. They have been implemented efficiently in FPGAs in FINN [84] and FP-BNN [50]. We mimic their network configurations, modified only in transforming them to run on our PIM substrate. Hence, our accuracy is identical. Neural networks [11], [90] and BNNs [81], [92] have been previously mapped to PIM substrates for acceleration, including on CRAM [70]. However, those designs rely on continuous power and have not considered correctness in intermittent computing and thus are not capable of functioning in the targeted energy harvesting domain.

IV. MOUSE Design

Energy harvesting systems are powered by their environment. If the environment does not provide enough power, the system will have to accumulate energy over time and consume it in bursts [29]. Therefore, such devices must consume as little energy as possible and be capable of tolerating power outages while maintaining program correctness. MOUSE is a natural fit for such a paradigm as logic operations are highly energy efficient and the memory is entirely non-volatile. Additionally, all computation occurs within the memory so progress is effectively saved after each operation. This greatly simplifies strategies to maintain correctness. In this section, we detail a basic MOUSE design which is tightly tailored to energy harvesting applications.

A. Hardware Organization

MOUSE has a tiled architecture. Certain MOUSE tiles are dedicated for instructions, while all others are dedicated for data and computation, as shown in Figure 5. MOUSE has a larger storage capacity than is typical for energy harvesting devices. This is due to two reasons. First, MRAM is dense and has extremely low standby power, giving the memory a low area and energy impact. For example, NVSIM [23] reports the size of 64MB STT-MRAM array –which is nearly twice the size of our largest configuration– as 15.12mm^2. 256MB and 1GB STT-MRAM memory manufactured by Everspin [1], [2] comes in a package that is 130mm^2. For reference, just the MSP430FR5994 micro-controller itself, commonly used as a sub-component of energy harvesting systems [15], [29], [33]–[35], [74], consumes over 100mm^2. Second, as there is no need for external processor logic or area costly volatile memory (such as SRAM), and due to minimal peripheral circuitry, nearly the entire area budget of MOUSE is available for memory arrays. That said, the SHE configuration has more area overhead than the STT configuration due to the presence of a 2nd transistor, which we expand upon in Section VIII. However, the area budget still remains modest.

There are only five components of MOUSE that are not memory arrays:

1) A memory controller that reads instructions from the instruction tiles and issues all instructions;
2) An 128B memory buffer that facilitates reads and writes to the tiles;
3) A non-volatile register for Program Counter (PC);
4) A non-volatile register for buffering a single instruction;
5) Voltage sensing circuitry for monitoring the power source.

The memory controller only needs to differentiate between three instruction types as will be described in Section IV-B. All computation and memory operations are performed in the tiles, hence the controller need only broadcast the appropriate command to the tiles. The memory buffer is the same size as one row of the MOUSE tiles and is used for intermediate storage when transferring data to and from the tiles. The non-volatile registers are used for maintaining correctness during power outages, as will be described in Section IV-D. Finally, the voltage sensing circuitry is standard for energy harvesting systems, and is as described in [53].

B. Instructions

Instructions for MOUSE are 64-bit and the formats are shown in Figure 6. There are three types of instructions, logic operations, memory operations, and column activation. Memory operations are the same as standard read and write operations for MRAM. Instructions for logic operations specify the type of operation (which determines the applied voltage level) and the rows on which input and output cells reside. When a logic instruction is issued, it will be applied to every column that is currently active. Columns are activated by the *Activate Columns* instruction, which provides a list of column

addresses to a column decoder. Once columns are activated they are held active by a latching mechanism as proposed by [49]. This allows columns to remain active over multiple instructions. As columns need to be changed infrequently, typically staying active for many instructions, the peripheral cost for activation is amortized. This cost is further reduced by modifying the encoding to allow for bulk addressing, similar to the procedure in [78].

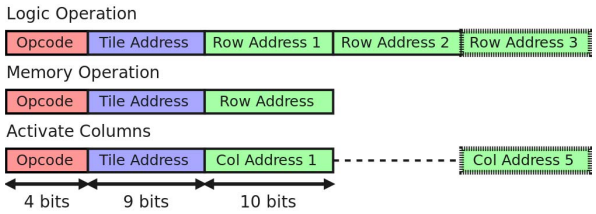


Fig. 6. MOUSE instruction formats. There are three types of instructions, logic, memory, and an additional activate columns instruction for configuration. Opcodes are 4 bits; tile addresses, 9 bits; and row and column addresses, 10 bits each. Dashed items are optional.

Compiling instructions for MOUSE requires some knowledge of the hardware to make efficient use of potential parallelism. This situation is analogous to compiling for GPU architectures from Open-CL or CUDA code. Unfortunately there is no generic equivalent for PIM. In the following we will provide pointers for efficient compilation, but inevitably leave detailed exploration of this rich design space to future work. Otherwise, architecture and data layout for MOUSE is similar to a number of other works which have mapped applications to PIM substrates [49], [78], including BNN implementations [70].

While operations can occur in multiple tiles simultaneously, tiles do not operate autonomously. All operations are triggered by the memory controller (discussed in more detail in Section V). Effectively, there is a single controlling "thread", and hence there are no concurrency concerns between individual tiles.

A subset of the tiles are dedicated to store the instructions. In the prototype MOUSE implementation, instruction and data tiles are homogeneous in design. The instructions are written into these tiles before deployment. Once active, the memory controller fetches each instruction from the instruction tiles, decodes it, and then broadcasts it to the tiles storing data. Instructions vary in the amount of time they take to complete. This is because specifying row and column addresses has an associated latency, and different instructions have different numbers of addresses. Logic operations can use 2 or 3 rows and column activation can specify up to 5 columns at a time. To ensure that every instruction finishes in time, the memory controller waits longer than the longest taking instruction needs before issuing the next. This time lapse forms a *cycle*. While this conservative approach comes at some cost of performance (more complex techniques could potentially issue instructions faster, in an event-driven fashion), MOUSE already is capable of extreme performance relative to other devices in this domain, as shown in Section VIII. Additionally, energy efficiency (rather than throughput) is the limiting factor for energy harvesting devices. Hence, we opt for simplicity at the cost of some performance loss.

Finally, as we are only performing inference in MOUSE, the sequence of instructions performed doesn't change as a function of inputs at runtime. Instructions are performed in sequential order one by one until the program repeats.

C. Power Draw

Most energy harvesting devices utilize an energy buffer (capacitor), rather than having the power source directly attached [54]. This prevents the need to match the power consumption with the power source. A switched-capacitor voltage converter can be used to apply the all appropriate voltages to the device [32], [42], [68], including the voltages required to perform all logic gates (explained further in Section VIII). MOUSE performs a single type of operation in each cycle. A portion of each cycle must be dedicated to changing the output voltage of the converter, if consecutive operations require different voltage levels. The converter may have an efficiency anywhere between 35-80%, hence the energy harvesting power source will have to provide energy in addition to that which MOUSE consumes.

By utilizing an energy buffer, MOUSE acquires energy over time and then consumes it in bursts. Hence, MOUSE could consume more power during power-on time than the energy-harvesting power source provides. However, we note that it is possible to reconfigure MOUSE to consume a specified power (to stay within a specified power budget), if this is known prior to deployment. By adjusting the amount of parallelism in the computation, the power consumption of MOUSE can be finely tuned. This enables a trade-off between latency and power draw. However, this can place strict limitations on potential parallelism. For example, if the power source can only deliver low power, e.g., 60 µW (an efficiency of 35% from a 171 µW power source), MOUSE would only be able to perform logic operations in 4 columns simultaneously (using the least energy efficient configuration described in Section VIII).

D. Intermittent Processing

As energy harvesting systems frequently experience power outages, they must be designed to perform intermittent processing. This involves addressing the challenge of maintaining correct state while repeatedly shutting down and restarting. The mechanism for maintaining correct state also needs to be efficient, as to avoid consuming the precious energy available for program execution. A number of techniques have been designed to ensure correctness [14], [28], [62], [71]. These studies have devised sophisticated techniques to ensure correctness while introducing minimal backup and restart overhead. In contrast, MOUSE maintains correctness with just a program counter (PC) and an additional non-volatile status bit. While extremely simple, and would be crude for other architectures, it is a natural fit for MOUSE. The simplicity of this technique is enabled by our novel architecture. More sophisticated techniques are unsuitable and unnecessary as MOUSE has no volatile data to backup. As MOUSE performs all computation within the non-volatile memory, progress is saved after each operation. This makes restarting after the last instruction possible and ideal.

When MOUSE restarts, only two pieces of information are required: the last instruction that was performed and the columns that were active. In order to restart from the last instruction, MOUSE writes; i.e., checkpoints, the PC into a non-volatile register after each instruction. When MOUSE gains sufficient power to restart, it simply reads the next instruction from the address in the PC. In the worst case, the power is cut after the last instruction is issued and performed, but before the update to the PC register. This does not break correctness as the same result is obtained if a single instruction is repeated multiple times, i.e., each such repetition is *idempotent* [37], [86] as will be shown in Section V-A. The only requirement is that the PC checkpoint happens strictly after each instruction is performed. Restarting after the very last

TABLE I
FOUR POSSIBLE CASES FOR RE-PERFORMING AN INTERRUPTED AND GATE. THE OUTPUT MTJ EITHER SHOULD OR SHOULD NOT SWITCH FOR CORRECT OPERATION, AND IT EITHER DID OR DID NOT PRIOR TO THE POWER BEING CUT.

	Output did not switch before interrupt	**Output did switch before interrupt**
Output should not switch	Repeating the operation is the same as performing it for the first time; no switching will occur (correct output).	Not possible. There cannot be sufficient current to induce switching at any point of the operation (be it before of after the interrupt). Repetition cannot induce switching by construction.
Output should switch	Repeating the operation is the same as performing it for the first time, and will now result in switching (correct output).	The output has already switched to 0 (correct output). Repetition, i.e., re-applying the same voltage will result in a larger current. Due to the *direction* of the current, however, staying the same (as before the interrupt), the output will remain at 0 (and cannot switch back to 1).

instruction not only minimizes the amount of work potentially lost on shutdown, but it also simplifies the restart process. The simple correctness guarantee, an operation being *idempotent*, does not hold if we were to repeat multiple instructions. This is because over the course of multiple instructions, temporary values can be created. These temporary values may be used later in the computation or periodically overwritten. Repeating multiple instructions on startup would require some method for ensuring correctness of these temporary values, such as performing additional presetting operations. This is certainly possible to do, but it introduces additional (and unnecessary, as we will see shortly) complexity.

The second requirement is to restore the previously active columns, for which we use a similar procedure. Whenever an *Activate Columns* instruction is issued, it is stored in an additional instruction register. Reissuing this last *Activate Columns* instruction is the first action on restart.

This scheme gives MOUSE minimal backup and restart overhead. To summarize, the cost is 1) continuous checkpointing of the program counter and *Activate Columns* registers and 2) an additional issue of an *Activate Columns* instruction on every restart. Both of these actions incur far less energy than a typical logic instruction. We make sure that operations happen in the correct order by performing them sequentially; updates to architectural state occur only after the current instruction is performed. It is noteworthy that MOUSE is always in a state which is safe to shut down in. Hence, MOUSE maintains correctness even if power is cut unexpectedly. We provide more detail on maintaining correct state in Section V-B.

There is an efficiency trade-off in the frequency of checkpointing [60]. Doing so more often results in less work potentially lost on shut-down, however this also increases the checkpointing overhead. The optimal approach will depend on the power source. MOUSE consumes energy on every cycle to perform checkpointing. If energy-harvesting is able to supply sufficient power, making power interruptions less frequent, it is possible that MOUSE would be more energy efficient performing checkpointing less often. However, we opt to checkpoint on every cycle as this keeps the design complexity minimal, which is enabled by the energy efficiency of MOUSE's checkpointing.

E. System Integration

During inference, MOUSE itself holds all static data required and performs all the computation. To be integrated into an energy harvesting system, MOUSE needs to receive energy from an energy harvester, receive input from a sensor, and send output to a transmitter. In this work, we assume input data is stored in a non-volatile buffer in the sensor prior to inference. The sensor's buffer is assigned a tile address and is treated as one of the tiles. Additionally, the buffer contains a non-volatile valid bit indicating that new input is ready. When MOUSE is ready for new input, the memory controller can check the valid bit and trigger a memory transfer. The memory transfer then consists of reads from the buffer and writes to the MOUSE data tiles. These reads and writes can be controlled by instructions at the beginning of the program. When MOUSE finishes inference, the memory controller reads out the data from the tiles. This data is then available to be transferred to the transmitter. In this work, we focus only on the accelerator and do not consider any overhead for the sensor or transmitter.

MOUSE can also handle potential sensor data corruption due to power outage. A dedicated non-volatile register along with an instruction to orchestrate sensor reads achieves this. When sensor read begins, this instruction stores current PC in a dedicated register. If power goes out during reading sensor data, on restart, MOUSE checks the valid bit in the sensor buffer (which stays zero under corruption). If zero, MOUSE goes back to first instruction handling sensor read (getting PC from the dedicated register). MOUSE can checkpoint such PC at any code location. As an alternative design point, MOUSE can offload this orchestration to software, as well. Otherwise, if power outage happens during computation, MOUSE does go back in time, but at most by one operation. Going further back is unnecessary. The most recent checkpoint is guaranteed to be correct. Idempotency ensures that neither following operation nor following checkpoint can corrupt it. All data remains consistent, hence there cannot be corruption on reboot.

V. CORRECTNESS GUARANTEE

We show that correctness is guaranteed in spite of power outages, even when unexpected. There are two components, the correctness of individual operations when interrupted or re-performed (Section V-A) and correctness of state variables in transitions between states (Section V-B).

A. Operation Level Correctness

In this section we show that correctness is maintained if a single operation is repeated, i.e., that repeating any single operation is idempotent [37], [86]. Given that the power may be cut at any moment, we must consider what happens when an operation is interrupted in all its possible stages. Since all operations in MOUSE are threshold operations, the two stages are pre- and post-switching. Additionally, switching of the output MTJ either should or should not occur depending on the inputs. To be explicit, we use AND as an example, however, our observations here apply to all gates.

The preset value for the output of an AND gate is 1, meaning the MTJ has a high resistance. During operation, current is applied in a direction that could change the output state to 0. If either of the two inputs is 0, there will be a sufficient current to change the state, otherwise it will remain at 1. We show the four possible cases in Table I: If, due to the inputs, the output is not supposed to switch, the output MTJ will not switch before the power is cut or after the power is restored. On the other hand, if the output is supposed to switch, it does not matter if it switches before the power outage or after.

If the output MTJ does not switch before the power outage, it will switch once power is restored and the operation is re-performed. If the output MTJ does switch to 0 before the power

outage, re-performing the operation once the power is restored will leave the output at 0. This is because the direction of the current can only change the output to 0, it cannot revert it back to 1 due to basic MTJ physics.

The catch here is that repeating a logic gate is effectively the same as performing the gate for a longer duration. Doing so results in an identical outcome, regardless of whether the output MTJ switched before interruption (i.e., power outage) or not. The case for writes is even simpler. The result of a write operation does not depend on the preset value, hence repeating a write is effectively writing the value twice. Such power interruptions can lead to wasted energy (as we may end up re-performing unnecessary work) but cannot result in corruption of logical values.

We do not require idempotency beyond a single logic gate as we perform only one logic gate per cycle (per column). More complex operations (such as additions or multiplications) are broken down into individual gate operations, which are then performed sequentially in consecutive cycles (hence separated by checkpoints, as will be explained in Section V-B). Our Boolean gate set is universal, allowing arbitrary computation.

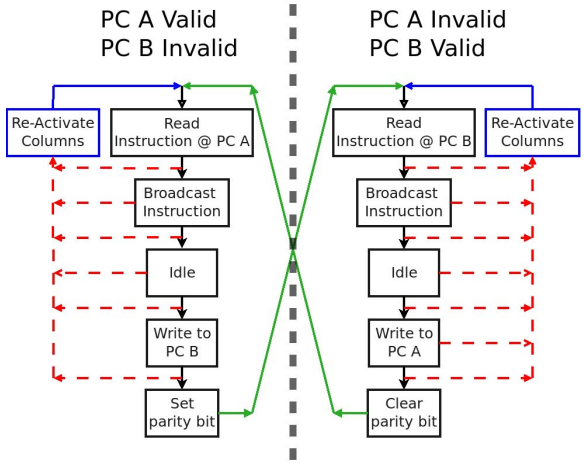


Fig. 7. Memory controller's state transitions to ensure correctness of the program counter as MOUSE transitions from one instruction to the next. Effect of interrupts are dashed and highlighted in red, corrective measures in blue, and forward progress (guaranteed completion of an instruction) in green.

B. Maintaining Correct State

It must also be ensured that the memory controller can tolerate unexpected interruptions and that MOUSE can maintain correctness as it transitions from one operation to the next during program execution. Here, we describe how correctness of the architectural state variables and data is guaranteed during this process.

1) Architectural State: The memory controller reads instructions from the address held in the non-volatile program counter (PC), decodes them, and broadcasts them to the data tiles. It then updates the PC. If power is cut during a write operation to the PC, the value may be corrupt. We solve this by using two PC registers and maintaining a parity bit. We refer to these two registers as PC-A and PC-B. If the parity bit is 0 then PC-A is valid and if the parity bit is 1 then PC-B is valid. The valid PC register points to the instruction currently being executed.

After an instruction is completed, the value stored in the valid PC register is read, updated (to point to the next instruction), and the new value is then written into the invalid PC register. At this point the invalid (valid) PC register keeps the address of the next (current) instruction that is to be executed (completed). After the PC register update, the parity bit is flipped. This process is depicted in Figure 7.

With this scheme, a write is never performed on the currently valid PC, hence, a valid copy of the PC is maintained at all times. If power is cut after the update to the invalid PC but before the parity bit is flipped, the memory controller will consider the old PC to be valid on restart. This results in the previous instruction being re-performed, and cannot introduce errors (since individual instructions are idempotent), as explained in Section V-A. The register holding the last *Activate Columns* instruction is also duplicated, and is handled in an identical fashion to the PC. Hence, power can be cut at any point during the execution of an instruction and the memory controller can always guarantee correct operation upon restart.

2) Data: The broadcast from the memory controller –which initiates an operation in the data tiles, and is depicted as *Command(s)* in Figure 5– is not atomic, and thus can be interrupted at any stage. However, this cannot cause corruption as the broadcast itself is idempotent. There are two cases to consider, 1) a broadcast initiating memory and logic instructions and 2) a broadcast initiating *Activate Columns* instructions.

As explained in Section V-A, data in the MOUSE tile cannot be corrupted by an interruption during memory and logic operations –no matter what stage in its progression the operation gets interrupted. As a direct result, the broadcast cannot cause corruption as it's only effect is the initiation of the operation. Power can be cut before the broadcast reaches a tile, while the operation is being performed, or after the operation has finished –none of these cases can introduce error. The second case of *Activate Columns* instructions cannot result in any corruption either, as the peripheral circuitry is always re-configured after restart, which overwrites the action of the previous *Activate Columns* instruction. More fundamentally, no corruption can be the case if power was cut during an *Activate Columns* instruction, simply because no logic or memory operation can take place as an *Activate Columns* is in progress in the same tile.

VI. Putting It All Together

Energy-harvesting devices need energy efficient execution and checkpointing capabilities. MOUSE uses non-volatile PIM to provide both.

A high-level program can be converted to computational blocks, such as multiplications or additions. These blocks can be broken into individual gates. For example, n-bit addition can be implemented by performing n full-adds, each of which can be performed with 9 NAND gates. These individual gates can be performed in the columns of MOUSE's tiles. The gates are highly energy-efficient, and applications can exploit high-degrees of paralellism available in the MOUSE tiles to achieve performance.

Scheduling these gates in the tiles is a two-dimensional problem in space and time, where we leave the rich design space for automation and optimization for future work. A multi-dimensional trade-off exists between parallelism, data-transfer, area consumption, and energy efficiency. Mapping computation to use more columns can increase parallelism, as computation can proceed in each column simultaneously. However, not only does this increase memory usage, but it also incurs a potential data-transfer overhead. Intermediate values will have to be transferred between columns, via reads and writes, in order to produce the final result. This decreases the energy efficiency. Without loss of generality, in this paper we stick to greedy scheduling in minimizing energy consumption

and area usage by using the minimal number of columns; at a cost of latency.

For example, vector dot-products constitute the majority of SVM classification. We place as many as possible bits of the elements of two vectors into a single column, with extra rows available for scratch bits. The elements that do not fit are placed (aligned) into other columns. The vectors are next element-wise multiplied and then summed together with a sequence of gates. Finally, the partial sums are moved, via reads and writes, to a single column, where they are summed to produce the end result. By using many columns and multiple tiles, this can be performed for many vectors simultaneously.

Hence, a program on MOUSE consists of a sequence of logic gates in-memory, along with reads and writes to perform I/O and transfer data between tiles. These operations can be fully specified by instructions shown in Fig. 6. Memory instructions are either read or write. Logic instructions correspond directly to logic gates, such as NAND, NOT, etc. Activate Columns is a single instruction. Instructions of this format are stored in MOUSE's instruction tiles.

A simple memory-controller is responsible for reading the instructions, decoding the opcode, and then broadcasting the instruction (and necessary addresses) to the data tiles. After waiting a sufficient period of time (for instruction completion), the memory-controller updates the PC and "commits" the instruction by flipping the parity bit. The memory-controller needs only basic logic circuitry (for decoding) and a clock to keep track of time. Its functionality is analogous to the 1st, 2nd, and 5th stages of the classic 5-stage pipeline. It performs 1-Instruction Read, 2-Instruction Decode, and 5-Write Back (setting parity bit). The memory handles 3-Execution and 4-Memory Access.

MOUSE's strength is in its simplicity. All operations performed in-memory are inherently idempotent and automatically stored in non-volatile memory. Hence, MOUSE can be made intermittent-safe with lightweight additions to the memory controller. This includes duplicated, non-volatile copies of the PC and active columns registers.

VII. Application Mapping

We next provide a basic illustrative example for application mapping: 2-bit addition. Figure 8 shows the stages of converting high-level code to MOUSE instructions. The first step is conversion from high-level to intermediate-level, i.e., functional translation to required logic and memory operations for the underlying computation along with basic spatio-temporal optimization. Next, all of the specified operations get converted directly into MOUSE format instructions.

In Figure 8 two 2-bit integers are added to form a 3-bit integer. We can perform these additions in parallel, and choose for these to occur in columns 0 and 1 of tile 1. The first addends (a and c) are assigned to rows 0 and 2, and the second addends (b and d) are placed in rows 4 and 6. The sums (x and y) are chosen to be placed in rows 8, 10, and 12. The computation will require additional scratch bits (workspace), for which we assign the odd rows between the addends and sum, and some additional even and odd rows at higher row addresses, picked based on availability.

Given the location of the addends, sum, and workspace, the ADD function will generate the sequence of gates required to compute the sum. It does this by performing a half-add, and then as many full-adds required to complete the addition; in this case just 1. Note that all gates have inputs and outputs on opposite parity rows. The parallelism of these gates is determined by the active columns, which are set by the *Activate Columns* instruction. Since the operands are in columns 0 and 1, a single *Activate Columns* instruction is issued to activate these two columns for computation. Prior to this computation, the preset value for outputs for gates will need to be written into the respective columns. For simplicity we do not show this step, but it consists only of write instructions. After finishing all instructions, the sums reside in rows 8, 10, and 12, with x in column 0 and y in column 1.

Once the sequence of gates has been fully specified, it gets converted directly to instructions by replacing the gate with the corresponding opcode and inserting the tile and row addresses. The opcode specifies how many row or column addresses are required. As all instructions are 64-bit, a number of bits remain as don't care.

VIII. Evaluation Setup

Benchmarks: Energy harvesting systems are ideal for applications in which the system is difficult or inconvenient to power directly or with batteries. Examples include remote sensors and wearable tech. We choose benchmarks which are representative of different possible use cases, along with an additional standard benchmark.

MNIST [47], as an example small-scale image recognition for sensor networks, is a digit recognition data set, where there are 10 classes for digits 0-9. The input is a grey scale 28×28 pixel image with 8-bit precision. We use both BNNs and SVMs on this benchmark. For the SVM, the pixels are placed row wise into a 784 element vector. We also use a binarized version, where pixels that are greater than a threshold value are set to 1 and others to 0. This allows us to replace multiplications with AND gates for most parts of the computation. For BNNs, we tailor the network configuration of FPGA-based FINN [84] and FP-BNN [50] to function properly on MOUSE, by converting it to sequences of logic gates. Our logical configuration is exactly the same. Hence, our accuracy is identical. The FINN configuration only uses binarized input. It has three hidden layers of 1024 neurons (bits) each, and an output layer of 10 neurons with 10-bit precision. The FP-BNN configuration only uses 8-bit inputs. It has three hidden layers of 2048 neurons and the output layer has 10 neurons with 16-bit precision.

Human Activity Recognition (HAR) [3], as an example for wearable tech, is a data set containing measurements from an accelerometer and gyroscope embedded in a smartphone, which is carried by participants performing a variety of activities. The task is to classify each set of readings to which activity is being performed. We represent the input with fixed point integer format with 8-bit precision. Each input is a vector of 561 elements.

ADULT [44] is a commonly used benchmark for SVMs that contains census information and the task is to classify whether an individual makes greater than $50K per year or not. We use a reformatted version of the data set from libSVM [10]. Each input is a 15 element vector where each element is an 8-bit integer.

Our SVMs are trained and tested in R [67]. They are custom designed, however we do compare our results with libSVM [10] with the same inputs and obtain similar accuracy. In our custom implementation we do not use any operations that would be inefficient in MOUSE; all programs consist of bit-wise and integer arithmetic.

Performance and Energy Model: We simulate the benchmarks on MOUSE with an in-house simulator, also implemented in R. MOUSE has a tiled architecture. We set each tile to have a capacity of 128KB, which is an 1024x1024 array. We chose this size as it is a commonly recommended subarray

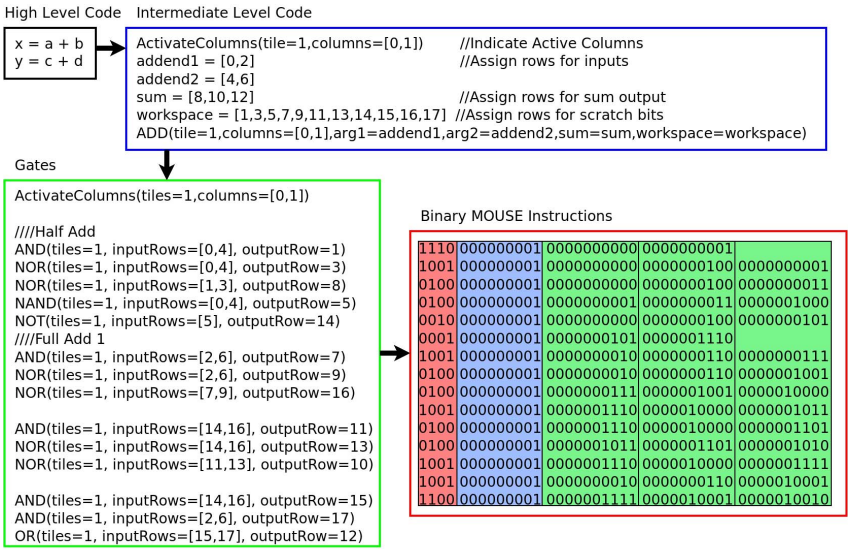


Fig. 8. An application mapping example of parallel 2-bit integer addition. Variables are assigned to rows and columns. Independent operations are mapped to separate columns for parallel execution. Computation is broken down into individual logic gates, which directly correspond to individual MOUSE instructions. During operation the memory controller issues each instruction in sequence. The MOUSE instructions shown are in the formats specified in Figure 6.

TABLE II
PARAMETERS FOR MTJ DEVICES.

Parameter	Modern	Projected
P State Resistance	3.15 kΩ	7.34 kΩ
AP State Resistance	7.34 kΩ	76.39 kΩ
Switching Time	3 ns [65], [72]	1 ns [39], [94]
Switching Current	40 µA [72]	3 µA [94]

size for non-volatile memories from NVSIM [23]. While only one logic gate can be performed in one column in each cycle, the gate can be performed in all 1024 columns simultaneously (column-parallelism) and in each tile simultaneously (tile-parallelism). Note that operating 1024 columns in parallel would require approximately 15 mW (on the least energy-efficient MOUSE configuration), which may exceed available energy. Additionally, high levels of parallelism can increase the restart cost during intermittent computing. This is because re-performing the last instruction on restart would cost more energy if it is highly parallel.

We experiment with both modern MTJ parameters [73] and projections of MTJ parameters in the next few years [94], [96]. Projected improvements in MTJ devices are expected to significantly increase energy efficiency. The MTJ parameters we use are shown in Table II. For projected MTJs, two techniques enable a reduction in the switching current, 1) decreasing the damping constant of ferromagnetic materials [24], [64], [76] and 2) using a dual-reference layer structure [21], [38]. To be conservative, we assume 3 µA, however, switching currents as low as 1 µA are possible. For projected MTJs, we test with both the STT and SHE based architectures. The main benefit of SHE is providing a write path through the SHE channel, rather than through the MTJ itself. To capture this effect, we assume a 1 kΩ resistance for the SHE channel, which is in series with the input MTJs in logic operations. This provides a conservative estimate of the SHE energy efficiency.

For Modern MTJs MOUSE operates at 30.3 MHz and for projected MTJs MOUSE operates at 90.9 MHz. This enables sufficient time for MTJ switching and peripheral circuitry latency.

To estimate latency and energy cost due to peripheral circuitry, we take data from NVSIM [23] which reports results for modern MRAM memories. We set our peripheral circuitry costs so that they consume the same percentage share of the total latency and energy as reported by NVSIM. In addition to the latency and energy required for performing the instructions, we also account for the overhead involved in reading the instructions from the tiles, updating the program counter and valid bits, specification of row and column addresses, storing the most recent *Activate Columns* instruction, and the re-issuing of the last *Activate Columns* instruction whenever the system restarts.

We first evaluate the performance of MOUSE under continuous power. Then, we evaluate MOUSE under energy harvesting conditions. We model our energy harvester as a constant power source which is filling an energy buffer (capacitor). MOUSE will start executing when the voltage on the capacitor is sufficiently high, and will shutdown when the voltage drops to a pre-determined level.

For Modern MTJs, the voltage on the capacitor fluctuates between 320 mV and 340 mV, and for Projected MTJs the range is 100 mV to 120 mV. We use switched-capacitor converters for upconversion and downconversion [32] to supply the required voltage for the operations. By using conversion ratios of 0.75, 1, 1.5, and 1.75 [42], [68], we can supply all voltages required. We evaluate MOUSE on the power supplied by the converter, the evaluation does not include regulator efficiency overhead. The converter may have an efficiency anywhere between 35-80%, hence the energy harvester may need to provide roughly 1.25-2.85× the energy that MOUSE consumes.

While energy harvesters can fluctuate in the amount of power they provide (e.g., amount of sunlight), this model captures a representative operation. We sweep the power source over a wide range, from levels well below the operating power of MOUSE, incurring numerous power outages, up to levels where MOUSE can nearly be continuously powered. Additionally, MOUSE assumes no knowledge when power will run out or when it will be restored. Effectively all outages are "unexpected".

Following metrics provided in [75], we report energy dedicated to different components. In addition to total energy,

TABLE III
AREA REQUIRED FOR MOUSE FOR DIFFERENT BENCHMARKS AND CONFIGURATIONS. UNITS ARE IN mm^2.

Benchmark	Total Memory	Modern STT [95]	Projected STT [95]	SHE
SVM MNIST	64MB	50.98	38.67	77.35
Binarized	8MB	5.43	4.13	8.24
SVM HAR	16MB	10.86	8.24	16.48
SVM ADULT	1MB	0.71	0.53	1.06
BNN FINN MNIST	8MB	5.43	4.13	8.24
BNN FPBNN MNIST	16MB	10.86	8.24	16.48

we report Backup energy, Dead energy, and Restore energy. Backup captures operations performed prior to shut down to save state. For us, this is the continual writing of the PC, parity bit, and storing each *Activate Columns* instruction in an additional instruction register. Dead energy is energy spent re-performing work that was lost during shut down, which in this case is repeating the last instruction on restart. Restore energy includes any operation needed to prepare MOUSE for computation on restart. For us, this is issuing the most recent *Activate Columns* instruction.

We also report Dead latency, which is the latency associated with re-performing instructions, and Restore latency, which is the time it takes to re-activate the columns on restart. There is no Backup latency, as backup operations occur during the same cycle as each instruction.

Area Overhead: MOUSE tiles have a similar area overhead to MRAM arrays. MOUSE has an extra bit line per column for the STT configuration. For the SHE configuration, it has an extra transistor and a SHE channel for each cell. The impact of the additional bit line is minor but the additional transistor has significant overhead. To estimate area overhead for MOUSE, we create estimates for cell size keeping access transistor resistance less than $1\,\mathrm{k}\Omega$. The access transistors dominate the area overhead. This is for two reasons: 1) the MTJs and SHE channel can be placed on a separate layer from the access transistors and 2) the access transistors are much larger. As the SHE design has twice as many access transistors, the cell area is approximately twice as large. To estimate peripheral circuitry area overhead, we take NVSIM [23] results for area efficiency for the same sized arrays and adjust our estimates by the same ratio. As NVSIM only works with memory capacities that are a power of 2, we assign the smallest memory size for which the entire benchmark will fit. For example, SVM MNIST requires only 34.5MB yet we assume MOUSE will consume 64MB of memory to perform this benchmark. Our conservative area estimates are shown in Table III.

IX. EVALUATION

Continuous Power: Results for MOUSE under continuous power are summarized in Table IV. Also reported are results for the same benchmarks using both our custom SVM and libSVM on a CPU, and a representative energy harvesting system SONIC [29] under continuous power. The CPU implementations are run on a supercomputing cluster using Intel Haswell 5-2680v3 processors. To be conservative, we account only for the processor power consumption and assume it operates at its idle power. SONIC uses a TI-MSP430FR5994 microcontroller and is powered by a Powercast P2210B energy harvester.

Overall, MOUSE shows significant energy efficiency advantages over other implementations, and competitive latencies. MOUSE does require more memory than SONIC, however, we believe this to be reasonable given that MOUSE is implemented in high density MRAM and does not need external processing logic or area costly volatile memory. MOUSE benefits greatly from binarizing the MNIST input. One bit inputs enable us to replace multiplications with AND gates, which significantly reduces the amount of computation required. This comes at a small cost in accuracy. The libSVM implementation struggles on the binarized MNIST inputs, and attempts to increase accuracy by adding many more support vectors. This increases the latency and energy of inference.

Let us next look into the significant difference in performance between MOUSE and SONIC [29]. SONIC is implemented on a conventional, low performance microprocessor. That design is highly economical, makes use of very scarce memory capacity, uses currently commercially available hardware, and has been proven experimentally. Additionally, the authors note that there is room for significant improvement in the efficiency. While we are reporting a significant latency and energy advantage, MOUSE is not fabricated yet. However, MTJ based logic has been experimentally demonstrated [87]. That said, MOUSE uses roughly the same area budget –that SONIC allocates for energy-hungry volatile memory and relatively complex logic– for more non-volatile memory (within which computation can be performed)[1].

Energy Harvesting: Now we consider MOUSE in a more realistic energy harvesting scenario. We test MOUSE with a range of power sources, from $60\,\mu\mathrm{W}$, approximately what can be harvested from a $1\mathrm{cm}^2$ thermal energy harvester running on body heat [43], [48], up to $5\,\mathrm{mW}$, the power harvested by SONIC [29]. The power source charges an energy buffer (capacitor) on chip. We allow the voltage to fluctuate between $320\,\mathrm{mV}$ and $340\,\mathrm{mV}$ when using Modern MTJs and between $100\,\mathrm{mV}$ and $120\,\mathrm{mV}$ when using Projected MTJs. When the voltage drops below the desired range, MOUSE shuts down and waits until the voltage reaches the upper end of the range. We assume that MOUSE starts with a capacitor with less charge than what would correspond to the shutdown voltage. Hence, all benchmarks begin with an initial charging time. We use a $100\,\mu\mathrm{F}$ capacitor (energy buffer) with Modern MTJs and a $10\,\mu\mathrm{F}$ capacitor for Projected MTJs. The optimal capacitor size depends on the technology and the program being executed. When deployed, a system such as Capybara [16] could be used to tune the parameters of the energy buffer.

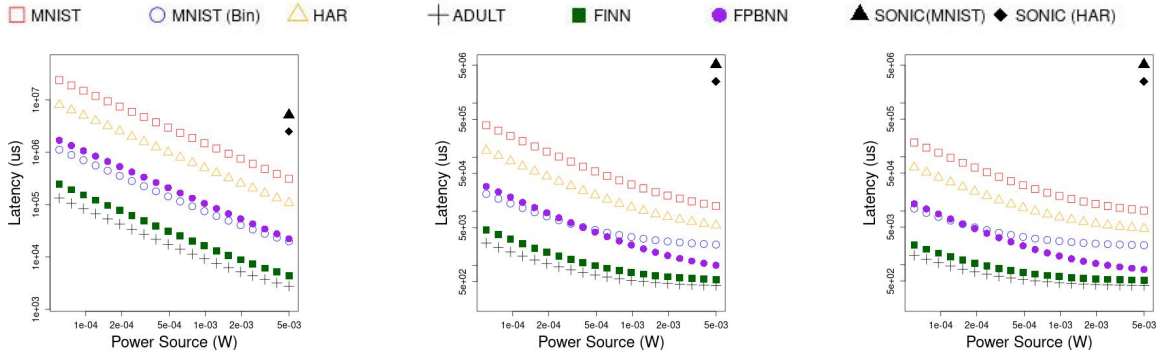
Results for latency are plotted and compared to SONIC for Modern STT, Projected STT, and SHE in Figure IX. Consistent with and as noted by [29], the latency is mostly determined by energy efficiency. This is because the majority of the latency is spent powered off, waiting for the capacitor to charge. The fewer recharges required, the lower the latency. Hence, latency increases significantly as the power source is reduced. Because the SHE design is more energy efficient, it draws significantly less power and thus drains the capacitor less often. This results in fewer power outages, fewer shutdown and restart operations, and hence also requires fewer operations to complete the program. This gives SHE a latency advantage over STT while in energy harvesting conditions. However, with all configurations, MOUSE achieves a significantly lower latency than SONIC, even with a much lower power budget.

The dependency of latency on energy efficiency results in a cross-over of the latency between FP-BNN and SVM MNIST (Bin) benchmarks. FP-BNN costs more total energy, and hence has a higher latency at lower power sources. However, due to

[1]Our largest configuration uses 35MB (fits in an 64MB array). Everspin's commercially available 64MB STT-MRAM device (similarly sized) is $130\mathrm{mm}^2$ [1]. Everspin's new 1GB product is the same (package) size [2]. In comparison, SONIC's microcontroller takes $100\mathrm{mm}^2$, which is only a sub-component.

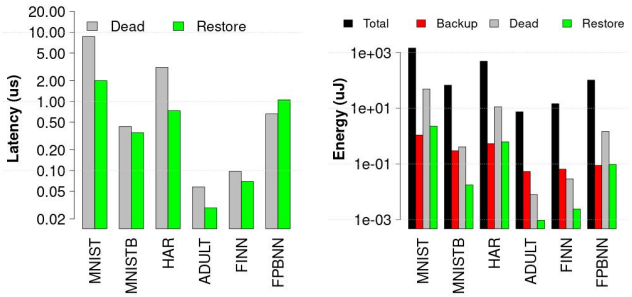
TABLE IV
CONTINUOUSLY POWERED MOUSE (USING STT DESIGN AND MODERN MTJ DEVICES) AND RELATED WORK UNDER CONTINUOUS POWER. THE CPU DOES NOT BENEFIT FROM MNIST BINARIZATION AS IT STILL PERFORMS 64-BIT INTEGER MULTIPLICATION.

Benchmark	Latency (µs)	Energy (µJ)	#SV	I/D Mem (MB)	Area (mm^2)	Accuracy
SVM (CPU)						
MNIST	169,824	5,094,702	11,813	-	-	97.55
MNIST (Binarized)	192,370	5,771,085	12,214	-	-	97.37
HAR (integer) [3], [83]	127,494	3,824,822	2,809	-		95.96
ADULT	4,368	131,052	1,909	-	-	76.12
MOUSE SVM (Modern STT)						
MNIST	23,936	1,384	11,813	4.5 / 30.0	50.98	97.55
MNIST (Binarized)	6,575	65.49	12,214	1.25 / 6.0	5.43	97.37
HAR (integer) [3], [83]	11,805	468.6	2,809	2.25 / 10.0	10.86	94.57
ADULT	1,189	7.24	1,909	0.25 / 0.5	0.71	76.12
MOUSE BNN (Modern STT)						
MNIST (Binarized) FINN	1,485	14.33	NA	3.15/1.71	5.43	98.4
MNIST FP-BNN	2,007	99.9	NA	4.20 / 8.00	10.86	98.24
libSVM [10]						
MNIST	7,830	234,900	8,652	-	-	98.05
MNIST (Binarized)	19,037	571,116	23,672	-	-	92.49
HAR (integer)	1,701	51,042	2,632	-	-	93.69
ADULT	379	11,370	15,792	-	-	78.62
SONIC [29]						
MNIST	2,740,000	27,000	NA	0.256	> 100	99
HAR	1,100,000	12,500	NA	0.256	> 100	88



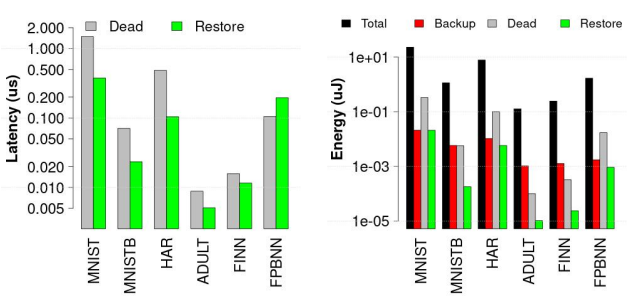
(a) Modern STT (b) Projected STT (c) SHE

Fig. 9. Latency (µs) vs. Power Source (W) for each MOUSE configuration and SONIC [29].



(a) Latency (b) Energy

Fig. 10. Latency/Energy Breakdown: Modern STT.



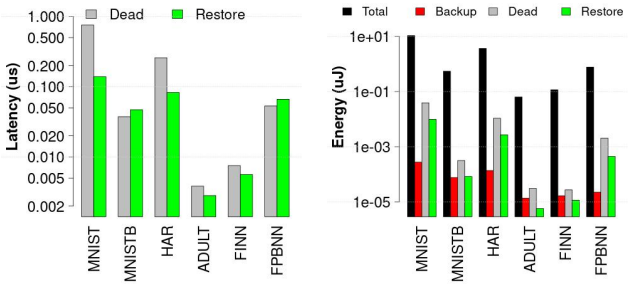
(a) Latency (b) Energy

Fig. 11. Latency/Energy Breakdown: Projected STT.

a higher degree of exploited parallelism, FP-BNN has a lower latency if sufficient power can be supplied.

As MOUSE spends negligible amounts of energy while powered off, the energy consumption is nearly independent of the power supply. The total energy is plotted in Figure 10(b) for Modern STT; in Figure 11(b) for Projected STT; and in Figure 12(b) for SHE; assuming a 60 µW power source.

Also of interest, as noted by [75], is the Backup, Restore, and Dead latency and energy. These are also reported in Figure 10 for Modern STT; in Figure 11, for Projected STT; and in Figure 12, for SHE. Note that the y-axis is log scale. The total energy encapsulates all energy used for computation, as well as Backup, Restore, and Dead energy. An efficient intermittent computing system will have low Backup, Restore, and Dead energy relative to the total energy, which can be seen is the case for MOUSE. Also note the total latency is provided for all architectures in Figure 9 – where the breakdown figures capture the data for the 60 µW power source.



(a) Latency (b) Energy

Fig. 12. Latency/Energy Breakdown: SHE.

Dead latency and energy is due to the possible re-execution of the previous instruction on restart. Dead latency and energy also increase with the number of restarts required and is also variable on when the interrupt occurred. The best case is if an interrupt occurred immediately after the parity bit is flipped (indicating the completion of an instruction per Section V-B and Figure 7). In this case, there is effectively no penalty. The worst case is if the interrupt happened just before the flipping of the parity bit, where the current instruction has been fully performed but not considered complete. In this case, the entire instruction will be re-performed on restart.

As Modern STT is the least energy efficient, it must restart the most and hence has the largest relative Dead energy. At the extremely low power of $60\,\mu W$, on average, across all benchmarks, Dead energy is 7.4% of the total energy. Higher energy efficiencies reduce this significantly, where Dead energy (on average) becomes 2.52% of the energy for Projected STT and 0.61% of the total for SHE. Dead latency, on the other hand, is 0.47% of the total for Modern STT, 0.09% of the total for Projected STT, and 0.044% of the total for SHE.

Restore is the time and energy required to re-activate the columns every time MOUSE restarts. Restore latency and energy naturally increase with the number of restarts required, but is also variable depending on where in the program an interrupt occurred. The more columns that were active at the time of an interrupt, the higher the respective Restore cost of re-activating them will be. However, overall, as the restore process is fast and energy efficient, the Restore latency and energy remain a small fraction of the total. On average across all benchmarks, Restore is only 0.91% of the latency and 0.50% of the energy for Modern STT; 0.14% of the latency and 0.13% of the energy for Projected STT; and 0.04% of the latency and 0.13% of the energy for SHE. As Restore latency and energy is due to peripheral circuitry, SHE has no advantage over STT for an individual restart. However, SHE still requires fewer restart operations due to its overall increased energy efficiency.

Backup energy entails the continual writing of architectural state variables (PC and parity bit). Backup energy corresponds to writing only a few bits on every cycle. An interrupt can cause at most a single additional write. Hence, Backup energy is not significantly affected by interrupts and is determined mostly by the length of the program. Backup energy is, on average across all benchmarks, 0.24% for Modern STT; 0.27% for Projected STT; and 0.007% for SHE. Backup has no associated latency as it is performed at the same time as each instruction on every cycle.

Restore and Dead latency and energy are all zero for the case of a continuously powered system. This is because there are no power outages and, hence, never a need to restart the system or re-perform any potentially unfinished instructions.

X. Related Work

Non-volatile processors (NVP) [53], [59] are uniquely designed for intermittent computing by integrating non-volatile memory near the compute units. Unlike MOUSE, these devices have a structure similar to traditional CPUs. MOUSE has three advantages over these architectures. Due to PIM, it is capable of high degrees of parallelism in order to achieve performance. Additionally, MOUSE doesn't need to perform energy-hungry loads and stores in order to operate on data. Finally, MOUSE doesn't perform additional backup operations prior to shut-down, making it effectively immune to unexpected power outages. The authors of [53] propose a system using a THU1010N non-volatile processor for energy harvesting applications. They describe trade-offs in designing such a system and demonstrate its capability on a number of benchmarks. There is follow up work in [57], [58] which makes the NVPs more resistant to power interruptions. The NVP in [57] can complete the FFT benchmark from MiBench [31] in $4.2\,\mathrm{ms}$. A recent paper [19] has evaluated FFT implementations on CRAM, the same substrate which MOUSE uses. Performing a similarly sized problem, the best latency they were able to achieve is $1.63\,\mathrm{ms}$. Naturally, adapting this implementation to be intermittent safe in the same manner in MOUSE would introduce a latency penalty. Another non-volatile processor is presented in [79] which features PIM components. There is a controlling CPU that performs logic and control. A few RRAM arrays are used to accelerate computing in neural networks. In this case, the PIM is a sub-component of the system, which also contains more traditional logic circuitry and an external processor. Due to this complexity, this implementation cannot make use of the same checkpointing strategy used in MOUSE.

A recent paper, ResiRCA, proposes an adaptable RRAM crossbar accelerator for MAC (multiply+accumulate) operations for CNNs in energy harvesting environments. It proposes clever methods to adapt the power consumption to varying power sources. While the crossbar is powered by an energy harvester, it assumes a battery powered host processor. Unlike MOUSE, computation also occurs outside the memory array (only MACs are processed by the memory). Hence, the automatic check-pointing mechanism we use is not applicable to this design. A number of RRAM PIM technologies also exist [80], [82], [91], [93]. However, the RRAM array is used as an accelerator as a sub-component of the system. Hence, there is much additional circuitry and logic that occurs outside the memory. This significantly increases the difficulty to adapt to intermittent processing. Additionally, many RRAM accelerators rely heavily on ADCs (analog to digital converters), which have a significant area and energy overhead. RRAM typically suffers from a lower endurance, as well.

Capybara uses a re-configurable hardware energy storage mechanism and a software interface that allows the specification of energy needs for different tasks. This gives the system more flexibility in satisfying the requirements of different kinds of tasks. While we do not focus on the power delivery system in this work, systems such as Capybara could be used to optimally supply MOUSE with power. Hibernus [7], on the other hand, is a system that reactively hibernates and wakes up. This is a similar shutdown policy to MOUSE. However, Hibernus performs an additional back-up operation before shutting down, whereas MOUSE does not need to.

A number of techniques have been developed to enable intermittent computation on more traditional hardware. For

example, CleanCut [14] works with LLVM to compile programs with checkpoints, and uses a statistical energy model to find potential non-terminating paths. Chinchilla [62] uses adaptive checkpointing, where the frequency of checkpoints is a function of the number of interrupts. Coati [71] developed methods to ensure correctness in the presence of interrupts for intermittent systems. The What's Next Intermittent Architecture [26] uses approximation to improve performance. Rather than following an all-or-nothing approach, What's Next computes approximate results and continually improves the output. If an acceptable output is achieved it will skip to processing the next input. This enables the device to process more inputs as it does not waste time and energy achieving unnecessary accuracy.

The EH model [75] facilitates early design space exploration for energy harvesting architectures. It helps finding a good balance to achieve minimal overhead for allowing maximal forward progress. As noted by the authors of [75], energy harvesting systems can generally be divided into two types, multi-backup, which perform many backups between power outages, and single back-up, which only save state once before a power outage. Multi-backup systems include Mementos, [69], DINO [55], Chain [13], Alpaca [61], Mayfly [36], Ratchet [86], and Clank [37]. Single-backup systems include Hibernus [6], QuickRecall [40], and many others [4], [5], [8], [52], [56]. According to this categorization, MOUSE fits under a multi-backup system as we are constantly saving the architectural state.

PIM has been studied for non-volatile memories with Pinatubo [49], for DRAM with Ambit [78], and for SRAM with Neural Cache [25]. These technologies are meant to be integrated into the memory hierarchy of traditional CPUs and have not been considered for energy harvesting applications. Ambit and Neural Cache are not suitable for energy harvesting as they are volatile technologies. Pinatubo could be adapted and used similarly as CRAM in MOUSE. However, Pinatubo uses logic external to the memory array for some operations. This adds complexity as these circuits would need to be protected against errors in intermittent computing. Additionally, Pinatubo uses sense amplifiers to perform computation, which is less energy efficient than the logic operations in CRAM.

The Phoenix processor [77] is an extremely low power processor with a sophisticated sleep strategy. However, it is not designed to be safe for intermittent processing. Similarly, while not safe for intermittent computing (and where adding this functionality would likely incur a significant performance and efficiency cost), a number of accelerators have demonstrated high performance and energy efficiency on inference. PuDianNao [51] is an ASIC accelerator which also targets SVM. A microcontroller based system was used as a BNN accelerator in [17]. An in/near memory SRAM substrate is proposed in [88], which performs bit-serial arithmetic, and which was shown to have high performance and efficiency on the AlexNet [46] network. A few analog PIM accelerators also exist. For example, the accelerator in [41] uses BNN to perform Cifar-10 image classification. The accelerator in [97] uses SRAM cells and analog computation to achieve high energy efficiency while classifying MNIST. Another example is [85] for MNIST and Cifar-10 recognition. Generally, adapting such accelerators to support safe intermittent computing is not straight-forward and would likely come –if at all possible– at significant performance and efficiency cost.

Orthogonal to our work, recent papers have made progress on problems relevant in the energy harvesting domain. Low power and accurate time keeping was developed in [20]. SRAM was used as a an efficient check-pointing memory, being able to maintain state for short periods of power off time [89]. A new platform for intermittent computing is proposed in [45] which simplifies the task of adapting pre-existing embedded applications to work in intermittent environments.

XI. Conclusion

In this paper we presented MOUSE, a machine learning accelerator in (non-volatile) memory for energy harvesting applications. The requirements for energy harvesting applications are extreme energy efficiency, efficient shut down and restart procedures, and correctness during intermittent execution. MOUSE provides all of these by having highly energy efficient logic operations with simple and effective shut down and restart procedures. The non-volatility combined with processing in memory provides a natural progress saving mechanism which demands very little overhead. By simulation, we demonstrated that such a device would provide significant latency and energy efficiency advantages over state of the art approaches, and is a promising candidate to bring machine learning to new domains.

References

[1] https://www.everspin.com/supportdocs/EMD3D256M08G1-150CBS1, 2019, accessed: 2019-08-10.

[2] https://www.everspin.com/family/emd4e001g?npath=3557, 2019, accessed: 2019-11-25.

[3] D. Anguita, A. Ghio, L. Oneto, X. Parra, and J. L. Reyes-Ortiz, "A public domain dataset for human activity recognition using smartphones." in *Esann*, 2013.

[4] F. A. Aouda, K. Marquet, and G. Salagnac, "Incremental checkpointing of program state to nvram for transiently-powered systems," in *2014 9th International Symposium on Reconfigurable and Communication-Centric Systems-on-Chip (ReCoSoC)*. IEEE, 2014, pp. 1–4.

[5] D. Balsamo, A. Das, A. S. Weddell, D. Brunelli, B. M. Al-Hashimi, G. V. Merrett, and L. Benini, "Graceful performance modulation for power-neutral transient computing systems," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 35, no. 5, pp. 738–749, 2016.

[6] D. Balsamo, A. S. Weddell, A. Das, A. R. Arreola, D. Brunelli, B. M. Al-Hashimi, G. V. Merrett, and L. Benini, "Hibernus++: a self-calibrating and adaptive system for transiently-powered embedded devices," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 35, no. 12, pp. 1968–1980, 2016.

[7] D. Balsamo, A. S. Weddell, G. V. Merrett, B. M. Al-Hashimi, D. Brunelli, and L. Benini, "Hibernus: Sustaining computation during intermittent supply for energy-harvesting systems," *IEEE Embedded Systems Letters*, vol. 7, no. 1, pp. 15–18, 2014.

[8] G. Berthou, T. Delizy, K. Marquet, T. Risset, and G. Salagnac, "Peripheral state persistence for transiently-powered systems," in *2017 Global Internet of Things Summit (GIoTS)*. IEEE, 2017, pp. 1–6.

[9] A. P. Chandrakasan, D. C. Daly, J. Kwong, and Y. K. Ramadass, "Next generation micro-power systems," in *2008 IEEE Symposium on VLSI Circuits*. IEEE, 2008, pp. 2–5.

[10] C.-C. Chang and C.-J. Lin, "Libsvm: A library for support vector machines," *ACM transactions on intelligent systems and technology (TIST)*, vol. 2, no. 3, p. 27, 2011.

[11] L. Chen, J. Li, Y. Chen, Q. Deng, J. Shen, X. Liang, and L. Jiang, "Accelerator-friendly neural-network training: Learning variations and defects in rram crossbar," in *Proceedings of the Conference on Design, Automation & Test in Europe*. European Design and Automation Association, 2017, pp. 19–24.

[12] Z. Chowdhury, J. D. Harms, S. K. Khatamifard, M. Zabihi, Y. Lv, A. P. Lyle, S. S. Sapatnekar, U. R. Karpuzcu, and J.-P. Wang, "Efficient in-memory processing using spintronics," *IEEE Computer Architecture Letters*, vol. 17, no. 1, pp. 42–46, 2017.

[13] A. Colin and B. Lucia, "Chain: tasks and channels for reliable intermittent programs," in *ACM SIGPLAN Notices*, vol. 51, no. 10. ACM, 2016, pp. 514–530.

[14] ——, "Termination checking and task decomposition for task-based intermittent programs," in *Proceedings of the 27th International Conference on Compiler Construction*. ACM, 2018, pp. 116–127.

[15] A. Colin, E. Ruppel, and B. Lucia, "A reconfigurable energy storage architecture for energy-harvesting devices," in *ACM SIGPLAN Notices*, vol. 53, no. 2. ACM, 2018, pp. 767–781.

[16] ——, "A reconfigurable energy storage architecture for energy-harvesting devices," in *ACM SIGPLAN Notices*, vol. 53, no. 2. ACM, 2018, pp. 767–781.

[17] F. Conti, P. D. Schiavone, and L. Benini, "Xnor neural engine: A hardware accelerator ip for 21.6-fj/op binary neural network inference," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 37, no. 11, pp. 2940–2951, 2018.

[18] M. Courbariaux, I. Hubara, D. Soudry, R. El-Yaniv, and Y. Bengio, "Binarized neural networks: Training deep neural networks with weights and activations constrained to+ 1 or-1," *arXiv preprint arXiv:1602.02830*, 2016.

[19] H. Cılasun, S. Resch, Z. I. Chowdhury, E. Olson, M. Zabihi, Z. Zhao, T. Peterson, J.-P. Wang, S. S. Sapatnekar, and U. Karpuzcu, "Crafft: High resolution fft accelerator in spintronic computational ram," in *Proceedings of the 57th Annual ACM/IEEE Design Automation Conference*, 2020.

[20] J. de Winkel, C. Delle Donne, K. S. Yildirim, P. Pawełczak, and J. Hester, "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.

[21] Z. Diao, A. Panchula, Y. Ding, M. Pakala, S. Wang, Z. Li, D. Apalkov, H. Nagai, A. Driskill-Smith, L.-C. Wang *et al.*, "Spin transfer switching in dual mgo magnetic tunnel junctions," *Applied Physics Letters*, vol. 90, no. 13, p. 132508, 2007.

[22] X. Dong, X. Wu, G. Sun, Y. Xie, H. Li, and Y. Chen, "Circuit and microarchitecture evaluation of 3d stacking magnetic ram (mram) as a universal memory replacement," in *2008 45th ACM/IEEE Design Automation Conference*. IEEE, 2008, pp. 554–559.

[23] X. Dong, C. Xu, Y. Xie, and N. P. Jouppi, "Nvsim: A circuit-level performance, energy, and area model for emerging nonvolatile memory," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 31, no. 7, pp. 994–1007, 2012.

[24] P. Dürrenfeld, F. Gerhard, J. Chico, R. K. Dumas, M. Ranjbar, A. Bergman, L. Bergqvist, A. Delin, C. Gould, L. W. Molenkamp *et al.*, "Tunable damping, saturation magnetization, and exchange stiffness of half-heusler nimnsb thin films," *Physical Review B*, vol. 92, no. 21, p. 214424, 2015.

[25] C. Eckert, X. Wang, J. Wang, A. Subramaniyan, R. Iyer, D. Sylvester, D. Blaauw, and R. Das, "Neural cache: Bit-serial in-cache acceleration of deep neural networks," in *Proceedings of the 45th Annual International Symposium on Computer Architecture*. IEEE Press, 2018, pp. 383–396.

[26] K. Ganesan, J. San Miguel, and N. E. Jerger, "The what's next intermittent computing architecture," in *2019 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 2019, pp. 211–223.

[27] K. Garello, F. Yasin, S. Couet, L. Souriau, J. Swerts, S. Rao, S. Van Beek, W. Kim, E. Liu, S. Kundu *et al.*, "Sot-mram 300mm integration for low power and ultrafast embedded memories," in *2018 IEEE Symposium on VLSI Circuits*. IEEE, 2018, pp. 81–82.

[28] G. Gobieski, N. Beckmann, and B. Lucia, "Intermittent deep neural network inference," 2018.

[29] G. Gobieski, B. Lucia, and N. Beckmann, "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*. ACM, 2019, pp. 199–213.

[30] H. Greenspan, B. Van Ginneken, and R. M. Summers, "Guest editorial deep learning in medical imaging: Overview and future promise of an exciting new technique," *IEEE Transactions on Medical Imaging*, vol. 35, no. 5, pp. 1153–1159, 2016.

[31] M. R. Guthaus, J. S. Ringenberg, D. Ernst, T. M. Austin, T. Mudge, and R. B. Brown, "Mibench: A free, commercially representative embedded benchmark suite," in *Proceedings of the fourth annual IEEE international workshop on workload characterization. WWC-4 (Cat. No. 01EX538)*. IEEE, 2001, pp. 3–14.

[32] R. Harjani and S. Chaubey, "A unified framework for capacitive series-parallel dc-dc converter design," in *Proceedings of the IEEE 2014 Custom Integrated Circuits Conference*. IEEE, 2014, pp. 1–8.

[33] J. Hester, T. Peters, T. Yun, R. Peterson, J. Skinner, B. Golla, K. Storer, S. Hearndon, K. Freeman, S. Lord *et al.*, "Amulet: An energy-efficient, multi-application wearable platform," in *Proceedings of the 14th ACM Conference on Embedded Network Sensor Systems CD-ROM*. ACM, 2016, pp. 216–229.

[34] J. Hester, L. Sitanayah, and J. Sorber, "Tragedy of the coulombs: Federating energy storage for tiny, intermittently-powered sensors," in *Proceedings of the 13th ACM Conference on Embedded Networked Sensor Systems*. ACM, 2015, pp. 5–16.

[35] J. Hester and J. Sorber, "Flicker: Rapid prototyping for the batteryless internet-of-things," in *Proceedings of the 15th ACM Conference on Embedded Network Sensor Systems*. ACM, 2017, p. 19.

[36] J. Hester, K. Storer, and J. Sorber, "Timely execution on intermittently powered batteryless sensors," in *Proceedings of the 15th ACM Conference on Embedded Network Sensor Systems*. ACM, 2017, p. 17.

[37] M. Hicks, "Clank: Architectural support for intermittent computation," in *2017 ACM/IEEE 44th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2017, pp. 228–240.

[38] G. Hu, J. Lee, J. Nowak, J. Sun, J. Harms, A. Annunziata, S. Brown, W. Chen, Y. Kim, G. Lauer *et al.*, "Stt-mram with double magnetic tunnel junctions," in *2015 IEEE International Electron Devices Meeting (IEDM)*. IEEE, 2015, pp. 26–3.

[39] G. Jan, L. Thomas, S. Le, Y.-J. Lee, H. Liu, J. Zhu, R.-Y. Tong, K. Pi, Y.-J. Wang, D. Shen *et al.*, "Demonstration of fully functional 8mb perpendicular stt-mram chips with sub-5ns writing for non-volatile embedded memories," in *2014 Symposium on VLSI Technology (VLSI-Technology): Digest of Technical Papers*. IEEE, 2014, pp. 1–2.

[40] H. Jayakumar, A. Raha, and V. Raghunathan, "Quickrecall: A low overhead hw/sw approach for enabling computations across power cycles in transiently powered computers," in *2014 27th International Conference on VLSI Design and 2014 13th International Conference on Embedded Systems*. IEEE, 2014, pp. 330–335.

[41] H. Jia, H. Valavi, Y. Tang, J. Zhang, and N. Verma, "A programmable embedded microprocessor for bit-scalable in-memory computing," in *2019 IEEE Hot Chips 31 Symposium (HCS)*. IEEE, 2019, pp. 1–29.

[42] W. Jung, S. Oh, S. Bang, Y. Lee, D. Sylvester, and D. Blaauw, "23.3 a 3nw fully integrated energy harvester based on self-oscillating switched-capacitor dc-dc converter," in *2014 IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC)*. IEEE, 2014, pp. 398–399.

[43] S. Kim, R. Vyas, J. Bito, K. Niotaki, A. Collado, A. Georgiadis, and M. M. Tentzeris, "Ambient rf energy-harvesting technologies for self-sustainable standalone wireless sensor platforms," *Proceedings of the IEEE*, vol. 102, no. 11, pp. 1649–1666, 2014.

[44] R. Kohavi, "Scaling up the accuracy of naive-bayes classifiers: A decision-tree hybrid." in *Kdd*, vol. 96. Citeseer, 1996, pp. 202–207.

[45] V. Kortbeek, K. S. Yildirim, A. Bakar, J. Sorber, J. Hester, and P. Pawełczak, "Time-sensitive intermittent computing meets legacy software," in *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*, 2020, pp. 85–99.

[46] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, 2012, pp. 1097–1105.

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

[48] V. Leonov, "Thermoelectric energy harvesting of human body heat for wearable sensors," *IEEE Sensors Journal*, vol. 13, no. 6, pp. 2284–2291, 2013.

[49] S. Li, C. Xu, Q. Zou, J. Zhao, Y. Lu, and Y. Xie, "Pinatubo: A processing-in-memory architecture for bulk bitwise operations in emerging non-volatile memories," in *Proceedings of the 53rd Annual Design Automation Conference*. ACM, 2016, p. 173.

[50] S. Liang, S. Yin, L. Liu, W. Luk, and S. Wei, "Fp-bnn: Binarized neural network on fpga," *Neurocomputing*, vol. 275, pp. 1072–1086, 2018.

[51] D. Liu, T. Chen, S. Liu, J. Zhou, S. Zhou, O. Teman, X. Feng, X. Zhou, and Y. Chen, "Pudiannao: A polyvalent machine learning accelerator," in *ACM SIGARCH Computer Architecture News*, vol. 43, no. 1. ACM, 2015, pp. 369–381.

[52] Q. Liu and C. Jung, "Lightweight hardware support for transparent consistency-aware checkpointing in intermittent energy-harvesting systems," in *2016 5th Non-Volatile Memory Systems and Applications Symposium (NVMSA)*. IEEE, 2016, pp. 1–6.

[53] Y. Liu, Z. Li, H. Li, Y. Wang, X. Li, K. Ma, S. Li, M.-F. Chang, S. John, Y. Xie *et al.*, "Ambient energy harvesting nonvolatile processors: from circuit to system," in *Proceedings of the 52nd Annual Design Automation Conference*. ACM, 2015, p. 150.

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

[55] B. Lucia and B. Ransford, "A simpler, safer programming and execution model for intermittent systems," in *ACM SIGPLAN Notices*, vol. 50, no. 6. ACM, 2015, pp. 575–585.

[56] G. Lukosevicius, A. R. Arreola, and A. S. Weddell, "Using sleep states to maximize the active time of transient computing systems," in *Proceedings of the Fifth ACM International Workshop on Energy Harvesting and Energy-Neutral Sensing Systems*. ACM, 2017, pp. 31–36.

[57] K. Ma, X. Li, J. Li, Y. Liu, Y. Xie, J. Sampson, M. T. Kandemir, and V. Narayanan, "Incidental computing on iot nonvolatile processors," in *2017 50th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 2017, pp. 204–218.

[58] K. Ma, X. Li, H. Liu, X. Sheng, Y. Wang, K. Swaminathan, Y. Liu, Y. Xie, J. Sampson, and V. Narayanan, "Dynamic power and energy management for energy harvesting nonvolatile processor systems," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 16, no. 4, pp. 1–23, 2017.

[59] K. Ma, Y. Zheng, S. Li, K. Swaminathan, X. Li, Y. Liu, J. Sampson, Y. Xie, and V. Narayanan, "Architecture exploration for ambient energy harvesting nonvolatile processors," in *2015 IEEE 21st International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 2015, pp. 526–537.

[60] ——, "Architecture exploration for ambient energy harvesting nonvolatile processors," in *2015 IEEE 21st International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 2015, pp. 526–537.

[61] K. Maeng, A. Colin, and B. Lucia, "Alpaca: intermittent execution without checkpoints," *Proceedings of the ACM on Programming Languages*, vol. 1, no. OOPSLA, p. 96, 2017.

[62] K. Maeng and B. Lucia, "Adaptive dynamic checkpointing for safe efficient intermittent computing," in *13th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 18)*, 2018, pp. 129–144.

[63] M. Manic, K. Amarasinghe, J. J. Rodriguez-Andina, and C. Rieger, "Intelligent buildings of the future: Cyberaware, deep learning powered, and human interacting," *IEEE Industrial Electronics Magazine*, vol. 10, no. 4, pp. 32–49, 2016.

[64] S. Mizukami, D. Watanabe, M. Oogane, Y. Ando, Y. Miura, M. Shirai, and T. Miyazaki, "Low damping constant for co 2 feal heusler alloy films and its correlation with density of states," *Journal of Applied Physics*, vol. 105, no. 7, p. 07D306, 2009.

[65] H. Noguchi, K. Ikegami, K. Kushida, K. Abe, S. Itai, S. Takaya, N. Shimomura, J. Ito, A. Kawasumi, H. Hara *et al.*, "7.5 a 3.3 ns-access-time 71.2 μw/mhz 1mb embedded stt-mram using physically eliminated read-disturb scheme and normally-off memory architecture," in *2015 IEEE International Solid-State Circuits Conference-(ISSCC) Digest of Technical Papers*. IEEE, 2015, pp. 1–3.

[66] F. Oboril, R. Bishnoi, M. Ebrahimi, and M. B. Tahoori, "Evaluation of hybrid memory technologies using sot-mram for on-chip cache hierarchy," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 34, no. 3, pp. 367–380, 2015.

[67] R Core Team, *R: A Language and Environment for Statistical Computing*, R Foundation for Statistical Computing, Vienna, Austria, 2016. [Online]. Available: https://www.R-project.org/

[68] Y. K. Ramadass and A. P. Chandrakasan, "Voltage scalable switched capacitor dc-dc converter for ultra-low-power on-chip applications," in *2007 IEEE Power Electronics Specialists Conference*. IEEE, 2007, pp. 2353–2359.

[69] B. Ransford, J. Sorber, and K. Fu, "Mementos: system support for long-running computation on rfid-scale devices," in *ACM SIGARCH Computer Architecture News*, vol. 39, no. 1. ACM, 2011, pp. 159–170.

[70] S. Resch, S. K. Khatamifard, Z. I. Chowdhury, M. Zabihi, Z. Zhao, J.-P. Wang, S. S. Sapatnekar, and U. R. Karpuzcu, "Pimball: Binary neural networks in spintronic memory," *ACM Transactions on Architecture and Code Optimization (TACO)*, vol. 16, no. 4, p. 41, 2019.

[71] E. Ruppel and B. Lucia, "Transactional concurrency control for intermittent, energy-harvesting computing systems," in *Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation*. ACM, 2019, pp. 1085–1100.

[72] D. Saida, S. Kashiwada, M. Yakabe, T. Daibou, N. Hase, M. Fukumoto, S. Miwa, Y. Suzuki, H. Noguchi, S. Fujita *et al.*, "Sub-3 ns pulse with sub-100 μa switching of 1x–2x nm perpendicular mtj for high-performance embedded stt-mram towards sub-20 nm cmos," in *2016 IEEE Symposium on VLSI Technology*. IEEE, 2016, pp. 1–2.

[73] ——, "Sub-3 ns pulse with sub-100 μa switching of 1x–2x nm perpendicular mtj for high-performance embedded stt-mram towards sub-20 nm cmos," in *2016 IEEE Symposium on VLSI Technology*. IEEE, 2016, pp. 1–2.

[74] A. P. Sample, D. J. Yeager, P. S. Powledge, A. V. Mamishev, and J. R. Smith, "Design of an rfid-based battery-free programmable sensing platform," *IEEE transactions on instrumentation and measurement*, vol. 57, no. 11, pp. 2608–2615, 2008.

[75] J. San Miguel, K. Ganesan, M. Badr, C. Xia, R. Li, H. Hsiao, and N. E. Jerger, "The eh model: Early design space exploration of intermittent processor architectures," in *2018 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 2018, pp. 600–612.

[76] H. Sato, E. Enobio, M. Yamanouchi, S. Ikeda, S. Fukami, S. Kanai, F. Matsukura, and H. Ohno, "Properties of magnetic tunnel junctions with a mgo/cofeb/ta/cofeb/mgo recording structure down to junction diameter of 11 nm," *Applied Physics Letters*, vol. 105, no. 6, p. 062403, 2014.

[77] M. Seok, S. Hanson, Y.-S. Lin, Z. Foo, D. Kim, Y. Lee, N. Liu, D. Sylvester, and D. Blaauw, "The phoenix processor: A 30pw platform for sensor applications," in *2008 IEEE Symposium on VLSI Circuits*. IEEE, 2008, pp. 188–189.

[78] V. Seshadri, D. Lee, T. Mullins, H. Hassan, A. Boroumand, J. Kim, M. A. Kozuch, O. Mutlu, P. B. Gibbons, and T. C. Mowry, "Ambit: In-memory accelerator for bulk bitwise operations using commodity dram technology," in *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture*. ACM, 2017, pp. 273–287.

[79] F. Su, W.-H. Chen, L. Xia, C.-P. Lo, T. Tang, Z. Wang, K.-H. Hsu, M. Cheng, J.-Y. Li, Y. Xie *et al.*, "A 462gops/j rram-based nonvolatile intelligent processor for energy harvesting ioe system featuring nonvolatile logics and processing-in-memory," in *2017 Symposium on VLSI Technology*. IEEE, 2017, pp. T260–T261.

[80] X. Sun, X. Peng, P.-Y. Chen, R. Liu, J.-s. Seo, and S. Yu, "Fully parallel rram synaptic array for implementing binary neural network with (+ 1,-1) weights and (+ 1, 0) neurons," in *Proceedings of the 23rd Asia and South Pacific Design Automation Conference*. IEEE Press, 2018, pp. 574–579.

[81] T. Tang, L. Xia, B. Li, Y. Wang, and H. Yang, "Binary convolutional neural network on rram," in *2017 22nd Asia and South Pacific Design Automation Conference (ASP-DAC)*. IEEE, 2017, pp. 782–787.

[82] ——, "Binary convolutional neural network on rram," in *Design Automation Conference (ASP-DAC), 2017 22nd Asia and South Pacific*. IEEE, 2017, pp. 782–787.

[83] https://archive.ics.uci.edu/ml/datasets/human+activity+recognition+using+smartphones, 2019, accessed: 2019-06-02.

[84] Y. Umuroglu, N. J. Fraser, G. Gambardella, M. Blott, P. Leong, M. Jahre, and K. Vissers, "Finn: A framework for fast, scalable binarized neural network inference," in *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*. ACM, 2017, pp. 65–74.

[85] H. Valavi, P. J. Ramadge, E. Nestler, and N. Verma, "A 64-tile 2.4-mb in-memory-computing cnn accelerator employing charge-domain compute," *IEEE Journal of Solid-State Circuits*, vol. 54, no. 6, pp. 1789–1799, 2019.

[86] J. Van Der Woude and M. Hicks, "Intermittent computation without hardware support or programmer intervention," in *12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16)*, 2016, pp. 17–32.

[87] J.-P. Wang, M. Jamaliz, A. K. Smith, and Z. Zhao, "Magnetic tunnel junction based integrated logics and computational circuits," *Nanomagnetic and Spintronic Devices for Energy-Efficient Memory and Computing*, p. 133, 2016.

[88] J. Wang, X. Wang, C. Eckert, A. Subramaniyan, R. Das, D. Blaauw, and D. Sylvester, "A 28-nm compute sram with bit-serial logic/arithmetic operations for programmable in-memory vector computing," *IEEE Journal of Solid-State Circuits*, vol. 55, no. 1, pp. 76–86, 2019.

[89] H. Williams, X. Jian, and M. Hicks, "Forget failure: Exploiting sram data remanence for low-overhead intermittent computation," in *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*, 2020, pp. 69–84.

[90] L. Xia, T. Tang, W. Huangfu, M. Cheng, X. Yin, B. Li, Y. Wang, and H. Yang, "Switched by input: Power efficient structure for rram-based convolutional neural network," in *2016 53nd ACM/EDAC/IEEE Design Automation Conference (DAC)*. IEEE, 2016, pp. 1–6.

[91] ——, "Switched by input: power efficient structure for rram-based convolutional neural network," in *Proceedings of the 53rd Annual Design Automation Conference*. ACM, 2016, p. 125.

[92] S. Yu, Z. Li, P.-Y. Chen, H. Wu, B. Gao, D. Wang, W. Wu, and H. Qian, "Binary neural network with 16 mb rram macro chip for classification and online training," in *2016 IEEE International Electron Devices Meeting (IEDM)*. IEEE, 2016, pp. 16–2.

[93] ——, "Binary neural network with 16 mb rram macro chip for classification and online training," in *Electron Devices Meeting (IEDM), 2016 IEEE International*. IEEE, 2016, pp. 16–2.

[94] M. Zabihi, Z. I. Chowdhury, Z. Zhao, U. R. Karpuzcu, J.-P. Wang, and S. S. Sapatnekar, "In-memory processing on the spintronic cram: From hardware design to application mapping," *IEEE Transactions on Computers*, vol. 68, no. 8, pp. 1159–1173, 2018.

[95] M. Zabihi, A. K. Sharma, M. G. Mankalale, Z. I. Chowdhury, Z. Zhao, S. Resch, U. R. Karpuzcu, J.-P. Wang, and S. S. Sapatnekar, "Analyzing the effects of interconnect parasitics in the stt cram in-memory computational platform," *IEEE Journal on Exploratory Solid-State Computational Devices and Circuits*, vol. 6, no. 1, pp. 71–79, 2020.

[96] M. Zabihi, Z. Zhao, D. Mahendra, Z. I. Chowdhury, S. Resch, T. Peterson, U. R. Karpuzcu, J.-P. Wang, and S. S. Sapatnekar, "Using spin-hall mtjs to build an energy-efficient in-memory computation platform," in *20th International Symposium on Quality Electronic Design (ISQED)*. IEEE, 2019, pp. 52–57.

[97] J. Zhang and N. Verma, "An in-memory-computing dnn achieving 700 tops/w and 6 tops/mm 2 in 130-nm cmos," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 9, no. 2, pp. 358–366, 2019.