On the Sweet Spot of Intermittent Inference in the Battery-less Internet of Things

Rei Barjami*, Antonio Miele*, and Luca Mottola*,+,‡
*Politecnico di Milano (Italy), +RI.SE Sweden, ‡Uppsala University (Sweden)

***Abstract*—We present a measurement and performance analysis of system-level settings to improve the energy efficiency of Deep Neural Network (DNN) inference on battery-less Internet of Things (IoT) devices. To do so, we *deliberately* trade a small, controllable reduction in inference accuracy for energy gains. Battery-less IoT devices are severely resource-constrained platforms powered by energy harvesting, where execution becomes *intermittent* as it alternates between bursts of computation and periods of energy recharge. To survive frequent energy failures, devices persist their system state into non-volatile memories, incurring significant energy costs. We leverage aggressive current scaling offered by Spin-Transfer Torque Magnetic Random-Access Memory (STT-MRAM) during state writes to reduce energy consumption, intentionally allowing controlled write errors that affect inference outcomes. Through an extensive experimental campaign comprising over 2.2+ trillion data points across 4 microcontroller units (MCUs) and 8 benchmarks, we demonstrate that by tolerating a limited accuracy loss we can obtain up to 40% energy savings. We release our framework and toolset to foster further research in this emerging design space.**

I. Introduction

Internet of Things (IoT) devices powered by ambient energy eliminate the need for traditional batteries [1], reducing maintenance costs and enabling unattended deployments [2], [3], [4]. Yet, harvested energy is typically scarce and unpredictable, causing frequent energy failures.

Challenge. Despite these constraints, existing work shows that efficient Deep Neural Network (DNN) inference is feasible even on battery-less IoT devices [5], [6]. These devices operate *intermittently*: computation proceeds in bursts powered by small energy buffers, such as capacitors, and is repeatedly interrupted by energy failures and corresponding recharge periods [7]. As shown in Fig. 1, devices start operating once the capacitor voltage exceeds V_{on}, and energy failures occur when the capacitor voltage drops below V_{off} and the device shuts down.

Battery-less IoT devices are usually resource-constrained devices powered by 16- or 32-bit microcontroller units (MCUs) with no operating system [8], and applications run on bare-metal. Consequently, volatile memory is lost on every shutdown. To guarantee forward progress, systems persist critical state, such as intermediate layer outputs, into Non-Volatile Memory (NVM) after each phase. Crucially, the cost for this operation is *massive*: persisting state can consume orders of magnitude more energy than computation itself [9]. Even so, device-local intermittent inference most often remains preferable to even more energy-expensive alternatives, such as wireless data transmission to a back-end [7], [5].

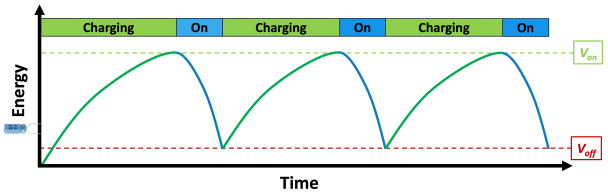


Fig. 1: Example of intermittent execution: computation alternates between recharging periods through ambient energy harvesting (green) and active operation (blue).

Opportunity. We observe that existing intermittent systems tend to assume that state persistence operations must be perfectly correct at all times. Yet, many DNN workloads are *inherently resilient* to small errors.

This feature offers a fundamental opportunity: modern NVMs, particularly Spin-Transfer Torque Magnetic Random-Access Memory (STT-MRAM) [10], allow *dynamic adjustment* of write current. Lowering write current reduces energy consumption but increases the probability of stochastic switching errors [11], [10]. By tuning the write current, we may thus deliberately trade a small increase of errors in NVM write operations for potential reductions in energy consumption. Since DNNs may tolerate moderate inaccuracies without substantial quality loss, the key question is:

> *How much energy can we save by introducing controller write errors during state persistence, and what is the impact on inference accuracy?*

Methodology and contribution. We systematically investigate this energy-accuracy trade-off across a broad design space: multiple MCU platforms, diverse neural network models, varied input datasets, and a wide range of memory fault rates. Our methodology combines precise energy modeling, hardware emulation, and real-device profiling.

Our findings are striking: across over 2.2 trillion data points we collect, *consistent sweet spots exist* where up to *40% energy savings* are attainable with *negligible impact* on inference accuracy. We also observe that *i)* the optimal trade-off point depends on the program, platform, and network characteristics—highlighting the need for whole-system investigation; and *ii)* quantized networks are generally more resilient to memory errors compared to non-quantized ones.

The rest of the paper unfolds as follows. Sec. II contrasts our work with the state of the art. Sec. III defines the problem, and Sec. IV outlines or methodology and toolset. We report on our

findings in Sec. V. We release our full framework, toolset, and datasets [12], enabling reproducibility and laying a foundation for future work in this area.

II. Related Work

We briefly survey the existing relevant literature and contrast that with our work.

Intermittent inference. Energy-harvesting or battery-less IoT devices face a fundamental challenge: frequent energy failures lead to loss of volatile memory. Long-running computations like DNN inference must be carefully managed to ensure forward progress and correctness across energy failures.

Prior work adapts rollback-recovery techniques to this domain. Hardware-assisted schemes [13], [14], [15] use voltage monitoring to trigger checkpoints. For instance, Hibernus defines a "guard band" threshold and hibernates the system once the capacitor voltage falls below the minimum needed for checkpoint completion [15], guaranteeing forward progress at the cost of idle periods for recharging. Software-based approaches like DINO [16] decompose programs into atomic tasks and use versioning to recover state after energy failures.

Works exist that modify the DNN itself to better accommodate intermittent executions. Multi-exit networks allow early inference termination under energy constraints, with limited accuracy loss [17], [18]. Model compression, pruning, and architecture search have also been used to fit DNNs within tight energy budgets [19], [20], [21], [22]. Some systems [6], [23] combine workload partitioning with energy prediction to better schedule inference across energy cycles.

Our work is *orthogonal* to these efforts. We execute *unchanged*, pre-trained networks without requiring model retraining, augmentation, or specialized scheduling. Instead, we focus on the critical yet under-explored challenge of *efficient persistence*: minimizing the energy cost of state saving when using STT-MRAM as NVM. By reducing the energy cost of state-persistent operations, we extend the effective execution window without altering the models.

Target NVM. STT-MRAM is a promising NVM technology offering fast reads and writes, high endurance, and virtually no standby leakage. Unlike flash or EEPROM, STT-MRAM provides SRAM-like performance while maintaining persistency, making it apt to intermittent systems [24]. However, STT-MRAM writes are subject to *stochastic switching*: reducing the write current to save energy increases the probability of bit errors [10], [11]. Conventional designs ensure correctness by using conservatively high write currents or by employing error correction codes [25], both of which increase energy consumption. Instead, we embrace controlled imprecision: DNN inference can tolerate occasional bit errors because the data pipelines are naturally robust to noisy inputs [26]. Adjusting the STT-MRAM write current thus offers a knob to tune the trade-off between energy and output quality.

Several works [10], [27], [28], [29] expose software interfaces to control write currents. For example, CAST [10] defines *Quality Levels (QLs)* tied to specific write current settings and associated bit error rates. These efforts primarily focus on architecture design and present a limited evaluation of application-level impact. Similarly, in prior work applying STT-MRAM to intermittent systems [5] we demonstrate write current tuning for DNN inference but within a restricted parameter space, and without systematically quantifying the relationship between write current reduction, error rates, accuracy loss, and energy savings. This work explicitly and exhaustively characterizes this trade-off across a wide range of system parameters, providing a thorough experimental basis for using energy-tuned STT-MRAM in intermittent inference.

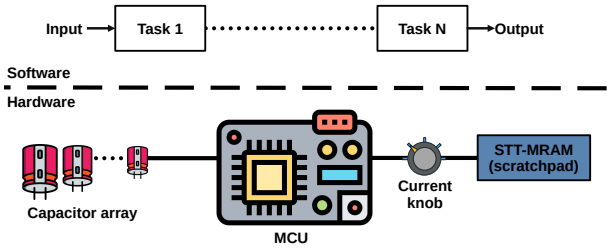


Fig. 2: Abstract representation of the system architecture. *Programs are structured as sequences of tasks. State restore and persistence operations are interleaved with task executions. The MCU is powered by a multi-capacitor architecture [30]. A knob allows setting the write current on STT-MRAM accesses.*

III. Problem and Objectives

We articulate the benefits we obtain from controlling the write currents in STT-MRAM and weigh them against the potential degradation in inference accuracy. We conclude by formulating the problem we tackle and the related objectives.

A. Platforms and Configurations

We target resource-constrained MCUs equipped with an STT-MRAM for non-volatile state persistence, as depicted in Fig. 2. Our platform mirrors existing battery-less IoT systems based on Cortex-M microcontrollers [2], [3], [4].

The STT-MRAM offers an interface to tune the write current, as the one in CAST [10]. The software controls this setting via a set of discrete QLs, each one characterized in terms of energy cost and probability of write error, namely Write Error Rate (WER). Note this is purely a software abstraction; our findings are valid regardless of the specific memory interface. Since not all data can tolerate write-induced errors, we partition the STT-MRAM address space into two regions: one storing critical variables (error-intolerant) and one storing error-resilient large data structures, such as DNN intermediate outputs. We set a QL guaranteeing no write error may occur in the former memory region, while in the latter, we tune QL at the granularity of individual variables.

In line with state-of-the-art intermittent computing, we consider task-based programming and a multi-capacitor setup [30]. Capacitors are dimensioned to match task energy requirements: small capacitors for lightweight tasks, and larger ones for compute-intensive operations. This design reduces recharge latency and leakage overhead [31].

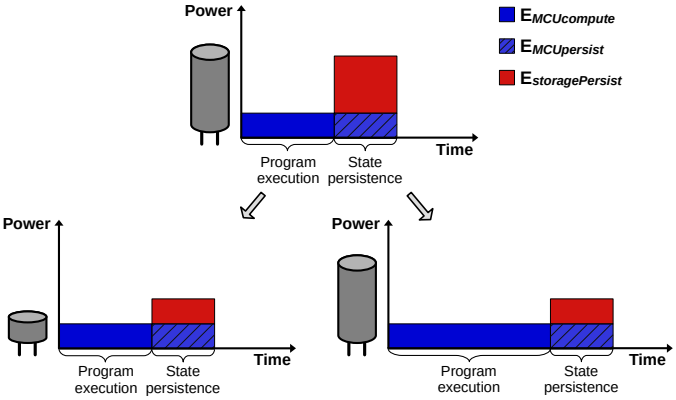


Fig. 3: Benefits of using current scaling for STT-MRAM accesses. *An example of task execution (top) and two ways to invest energy savings (bottom): either by reducing capacitor size (bottom left) or by extending computation (bottom right).*

Neural network inference is partitioned at *layer granularity*: each task executes exactly one layer. When a layer completes the execution, the output tensor is persisted to STT-MRAM to guarantee forward progress despite energy failures. This design enables effective fault tolerance but creates significant persistence overhead, since output tensors can be large. Reducing the energy cost of persisting tensors directly impacts the ability to process more layers within a single energy cycle, boosting overall system throughput.

B. Benefits and Drawbacks

The top of Fig. 3 qualitatively illustrates the energy profile of a task execution, including computation, state restore, and persistency operation. During state persistence, the MCU must actively transfer data to STT-MRAM, consuming energy both for memory writes (red area) and for its own operation during the write process (shaded blue area). This persistence overhead reduces the fraction of available energy that can be invested in useful computations.

By scaling down the write current, we reduce the energy consumption of the memory without affecting the duration of the memory transaction, since timing remains unchanged. As a result, we can improve system throughput by either

1) executing the same layer using a smaller capacitor, as shown in the bottom left of Fig. 3, reducing recharge time and leakage;
2) keeping the same capacitor and investing saved energy into additional work—potentially executing multiple layers per cycle—as shown in the bottom right of Fig. 3.

However, lower write currents increase the probability of bit errors, potentially corrupting the persisted output tensors. As illustrated in Fig. 4, three key factors influence the impact of such errors:

1) A single bit error can propagate to multiple downstream computations, as the output of a neuron is typically used as input by many neurons in the subsequent layer.

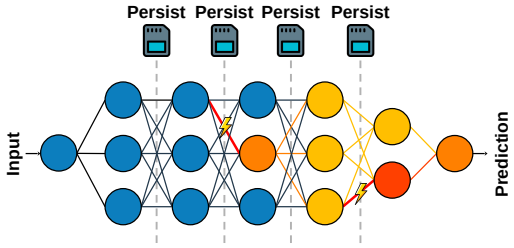


Fig. 4: Error propagation across layers. *Errors in persisted tensors propagate through downstream layers, compounding their impact and potentially causing significant degradation in final inference accuracy.*

2) Not all bit errors are equally harmful; flips in the most significant bits generally have a more severe effect than those in the least significant bits.
3) DNN error resiliency significantly varies model by model, as a function of structure, complexity, functionality, and intrinsic information redundancy.

C. Problem

The problem we address is therefore to identify a "sweet spot" in write current settings that offer meaningful energy savings, in exchange for a limited degradation in inference accuracy. This is a multi-faceted problem because the factors determining the energy savings and how errors impact the inference accuracy are many.

Given sufficient energy, we model the energy necessary to complete a task as

$$E_{total} = E_{MCUcompute} + E_{MCUpersist} + E_{storagePersist} \quad (1)$$

where $E_{MCUcompute}$ is the energy spent by the MCU in useful computations, shown in blue in Fig. 3, $E_{MCUpersist}$ is the energy spent by the MCU while the STT-MRAM writes data, which is shown in green, and $E_{storagePersist}$ is the energy spent by the STT-MRAM when writing data, shown in red.

Each term in Eq. 1 may be rewritten, piece by piece, depending on the hardware characteristics:

$$\begin{aligned} E_{\text{total}} &= E_{\text{cycle}} \cdot \#\text{cycles} \\ &\quad + E_{\text{cycle}} \cdot \#\text{memWriteAccesses} \\ &\quad + E_{\text{memWrite}} \cdot \#\text{memWriteAccesses} \\ &= E_{\text{cycle}} \cdot \#\text{cycles} \\ &\quad + (E_{\text{cycle}} + E_{\text{memWrite}}) \cdot \#\text{memWriteAccesses} \end{aligned} \quad (2)$$

where E_{cycle} is the MCU energy for a single compute cycle, $\#cycles$ is the number of MCU cycles to complete the tasks chosen for execution, $\#memWriteAccesses$ is the number of memory accesses during state persistence operations. Finally, $E_{memWrite}$ is the write energy for a single memory operation.

Given a task T in a program, the quantities of Eq. 2 depend on multiple software and hardware factors as:

$$\begin{aligned} E_{cycle} &= f_1(MCU_{activePower}, MCU_{freq}) \\ \#cycles &= f_2(MCU_{ISA}, T) \\ \#memWriteAccesses &= f_3(T) \\ E_{memWrite} &= f_4(QL) \end{aligned} \tag{3}$$

where $MCU_{activePower}$ and MCU_{ISA} capture the power figures and instruction set architecture of the chosen MCU, MCU_{freq} is the MCU operating frequency, and QL is the Quality Level for STT-MRAM writes corresponding to specific current settings. We derive functions f_1, f_4, and f_6 from an analysis of the datasheets and encode them as lookup tables, whereas we determine f_2, f_3, and f_5 through profiling executions on real hardware or emulators, as shown next.

IV. Methodology and Toolset

We illustrate the methodology and experimental framework that enable us to systematically explore the trade-offs between energy savings and inference accuracy across different configurations. We first introduce the benchmarks and hardware platforms in Sec. IV-A, followed by the description of our experimental framework in Sec. IV-B. We present the setup used for evaluation in Sec. IV-C.

A. Benchmarks, Metrics, and MCUs

Our evaluation spans eight different DNNs, detailed in Tab. I and deployed on intermittent systems [6], [23]. We use implementations from the STM ModelZoo suite [32], which cover a wide range of model sizes, architectures, and training datasets. Such heterogeneity is key for a comprehensive evaluation, as it allows us to investigate which characteristics make a DNN more resilient to memory access errors, and thus better suited to exploit aggressive energy-saving techniques.

We select models focusing on two main application domains: image classification and human activity recognition. The models are 8-bit quantized, in line with the constraints of edge deployments. The naming convention for the models reflects their input dimensions: for example, `FDMobileNet_96` and `MobileNetV1_224` process RGB images of sizes 96×96 and 224×224 pixels, respectively. In human activity recognition benchmarks, such as `IGN_wl_24` and `IGN_wl_48`, the inputs are matrices of accelerometer readings with dimensions 24×3×1 and 48×3×1, respectively.

Each DNN is structured as a sequence of layers, where the intermediate tensors generated by each layer must be stored into STT-MRAM before they are consumed by the next computation stage. To assess the impact of memory errors, we use classification accuracy as the quality metric across all models, following established evaluation practices [32], [33]. This allows us to consistently quantify the deviation in the output caused by introducing write errors.

We conduct experiments across a set of MCUs listed in Tab. II, comprising Cortex-M0, Cortex-M4, Cortex-M33, and Cortex-M7 cores [34]. We select these MCUs to cover a broad spectrum of hardware capabilities in terms of clock frequency, available main memory, and active power consumption. This diversity enables us to capture how microarchitectural features and power profiles influence the energy-accuracy trade-off. The active power figures reported correspond to the MCU operating in compute mode, with only the main memory and compute core powered on, ensuring measurements reflect purely computational workloads without I/O.

TABLE I: Benchmarks for experimental evaluation. *We use a diverse set of benchmarks from different domains, to enable a comprehensive evaluation of how the current scaling impacts the tradeoff between energy gains and accuracy.*

Benchmark	Domain	MACs	Max state size
`MobileNetV1_0.25_96`	Image classification	7.5M	39KiB
`MobileNetV1_0.25_224`	Image classification	41M	202KiB
`FDMobileNet_128`	Image classification	3.9M	49KiB
`FDMobileNet_224`	Image classification	12M	152KiB
`SqueezeNetV1.0`	Image classification	175M	450KiB
`STMnist`	Image classification	1M	12KiB
`IGN_wl_24`	Activity recognition	13K	1.63KiB
`IGN_wl_48`	Activity recognition	50K	5.2KiB

TABLE II: Selected MCUs. *Each MCU is characterized by different instruction sets and energy figures, allowing us to perform a comprehensive experimental evaluation.*

MCU	Clock speed (Mhz)	Main memory (KiB)	$\mathbf{MCU_{activePower}}$ (uW/Mhz)
Cortex M0	40	32	12.5
Cortex M33	160	768	12.0
Cortex M4	80	128	32.82
Cortex M7	480	1024	58.5

B. Framework

Ensuring reproducible experimental settings in intermittent computing is extremely difficult [7], due to the multitude of system variables [35], resource constraints of target hardware that limit visibility into the system operation [36], and the stochastic behavior of energy sources [1].

Design. To ensure accurate measurements and reproducible executions, we combine NVM estimation tools and profiling on real hardware, as shown in Fig. 5. For each DNN model and target MCU, we measure energy consumption and corresponding accuracy while applying the different QLs to STT-MRAM writes. As we articulate next, the performance figures for the MCU come from widely used emulators or real hardware executions, while STT-MRAM performance data comes from datasheets and state-of-the-art NVM simulators. These design choices provide a sound basis to maintain that our quantitative results are on par with real executions.

We measure the system's energy consumption as the sum of MCU and STT-MRAM energy. The MCUs we consider feature very simple architectures with no dynamic voltage/frequency scaling; we can reliably estimate their energy consumption as a simple function of power consumption and the number of cycles to execute the workload. We retrieve power consumption from the MCU datasheet ①. To compute the number of cycles, we profile executions to extract a trace of machine

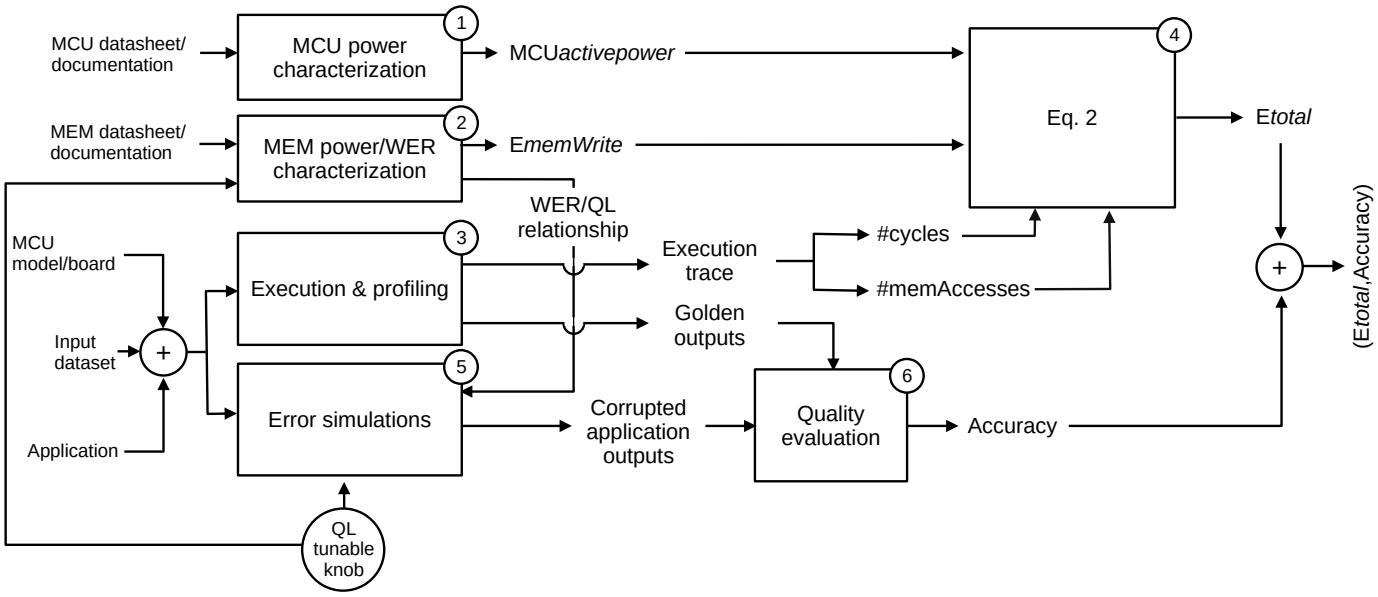


Fig. 5: Experimental framework and toolset. *Our framework processes input data derived from both datasheets and on-hardware measurements. It outputs the trade-off between energy consumption and accuracy for each QL.*

instructions and memory accesses ③. We perform multiple runs to average out the effects of inputs.

For write operations on STT-MRAM operations, energy consumption depends on the QL ②; we explain next how we obtain this value and the corresponding single-bit error probability. Finally, we extract the number of write operations from the execution trace. We then combine all these energy contributions in Eq. 2 ④.

Measuring the accuracy of a benchmark at a given QL requires memory access errors to be simulated. Since the errors caused by current scaling on STT-MRAM are stochastic, as common practice in reliability analysis of digital systems [37], we perform a large number of error simulation runs on every target architecture ⑤. We compare the faulty outputs against the golden counterparts to compute the accuracy loss ⑥. For each benchmark and MCU, we repeat experiments for all QLs and we collect energy consumption and accuracy loss.

Implementation. We use NVSIM [38] to obtain STT-MRAM energy figures. NVSIM is a widely used NVM simulator [10] that allows system designers to characterize STT-MRAM devices, and therefore to estimate read/write currents and energy consumption for a given chip configuration. We borrow the mapping between the energy invested for read and write operations at a given QL and the corresponding single-bit WER from existing literature [10]. We use hardware development kits from ST Microelectronics [39] to profile the benchmarks with Cortex M MCUs. We use hardware counters and on-board debugging facilities to count the number of executed cycles.

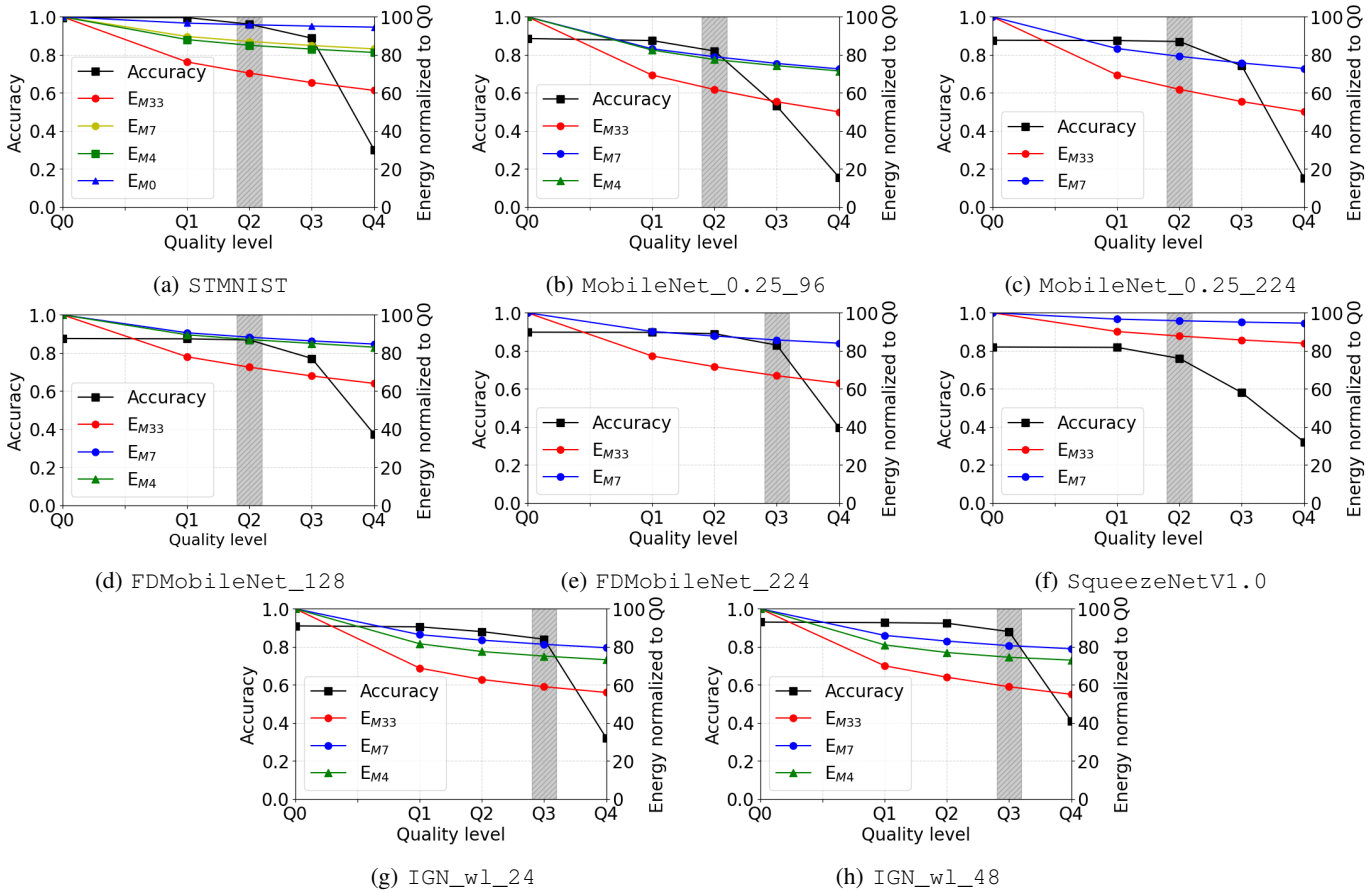
To simulate NVM errors, since the STT-MRAM is used as a scratch-pad memory, STT-MRAM operations are directly mapped to variable accesses in the application source code; as a consequence we can instrument the benchmark code in Tensorflow with an error injection functionality corrupting the bits during each variable write or read with the given WER probability as in existing literature [37]. As this strategy is *independent* of target hardware, as long as the data encoding is the same, it is possible to execute error simulations on a high-end workstation and yet obtain the same results as the target MCU, yet with a multi-fold reduction of execution times.

C. Setup

To systematically explore the energy-accuracy trade-off, we define five distinct QLs, denoted as `Q0` through `Q4`. The former `Q0` corresponds to standard operating conditions with virtually error-free memory writes, serving as the golden baseline. Progressively higher QLs reduce the memory write and read currents, resulting in lower energy consumption at the cost of increasing write error rates, as reported in Tab. III.

We derive the parameters associated with each QL, including set current and energy per bit, from simulations targeting a 32nm STT-MRAM process technology. Notably, even moderate current reductions yield substantial energy savings; for instance, setting `Q1` results in write operations that consume only about 56.3% of the energy compared to a `Q0` setting. However, these savings come with a significant increase in error probability, and the relationship between energy reduction and error rate is non-linear, with diminishing returns at higher QLs.

Each test configuration is defined as a combination of *i)* benchmark, *ii)* target MCU, and *iii)* QL for STT-MRAM operations. For each benchmark, we use between 15,000 and 70,000 distinct input samples, ensuring a diverse workload. We execute every configuration multiple times with different inputs, totaling between 750,000 and 3,750,000 runs per benchmark. This extensive sampling is necessary to accurately



(a) STMNIST (b) MobileNet_0.25_96 (c) MobileNet_0.25_224

(d) FDMobileNet_128 (e) FDMobileNet_224 (f) SqueezeNetV1.0

(g) IGN_wl_24 (h) IGN_wl_48

Fig. 6: Trade-off between energy savings and accuracy loss for Cortex M benchmarks with write current scaling. *The energy gain at the sweet spots, ranges from 5% to 40%, depending on MCU and benchmark.*

TABLE III: Characterization of QLs. *Write and read energy decrease linearly; while the corresponding WERs increase by orders of magnitude.*

QL	WER	Set current (μA)	Write energy per bit (pJ)
Q0	10^{-8}	1153	167
Q1	10^{-6}	865	94
Q2	10^{-5}	769	74
Q3	10^{-4}	673	57
Q4	10^{-3}	577	43

capture the stochastic behavior of errors and to provide statistically meaningful estimates of inference accuracy.

Resource constraints in the target MCUs dictate the mapping of benchmarks to platforms. For example, the Cortex-M0, with its limited memory capacity, can only execute the lightweight STMNIST benchmark, whereas the Cortex-M4 can accommodate the smaller image classification models but not the larger variants such as MobileNetV1_224. The Cortex-M33 and Cortex-M7, offering more generous memory budgets, are capable of running all benchmarks.

System capacitors, necessary for intermittent execution, are sized based on a hybrid analytical and empirical approach [40], [3], [41]. In particular, we adopt a tiered approach, sizing the largest capacitor according to the energy demand of the most computation-intensive layer, and supplement it with medium and small capacitors to execute layers of varying energy requirements [40], [30]. Additional details regarding benchmark inputs, number of repetitions, platform mappings, and capacitor sizing strategies are available elsewhere [12].

V. Results

Based on more than 2.2 trillion experimental data points, we provide quantitative evidence of where and to what extent it is possible to identify sweet spots that enable substantial energy savings in exchange for a limited degradation in accuracy, by applying current scaling to write operations. We also discuss key factors governing these trade-offs.

A. Sweet Spots

Fig. 6 illustrates the trends in energy savings and accuracy loss across benchmarks. The plots consistently reveal non-linear patterns in accuracy loss. Importantly, beyond Q0, there is almost always at least one QL where the accuracy is comparable to Q0 or deteriorates only slightly. The critical insight is to evaluate the energy gain at this alternative QL.

Across all DNNs tested on Cortex M-class MCUs, Fig. 6 shows that operating at Q2 often achieves an intangible or minimal impact on classification accuracy, typically only a few

percentage points at most. Yet, energy consumption improves by up to 40% compared to Q0. For specific benchmarks like FDMobileNet_224, similar improvements persist even at Q3. Among the platforms tested, the Cortex M33 achieves the best energy gains, and along with the Cortex M7, it is the only MCU capable of executing all considered benchmarks.

It is important to note that the best energy savings occur when the energy needed to persist data outweighs the energy required to compute them. Since we selectively reduce the energy of state persistence without affecting compute energy, applications where E_{storage} dominates $E_{\text{MCUcompute}}$ benefit the most. Thus, the full potential of the approach emerges when the output tensor is large enough, or persistence is costly enough, that optimizing only storage-related operations leads to visible system-level gains.

These findings highlight the necessity of empirical tuning to discover optimal operating points, which vary across hardware and workload characteristics. The framework we release [12] streamlines this process, enabling developers to explore and exploit these trade-offs.

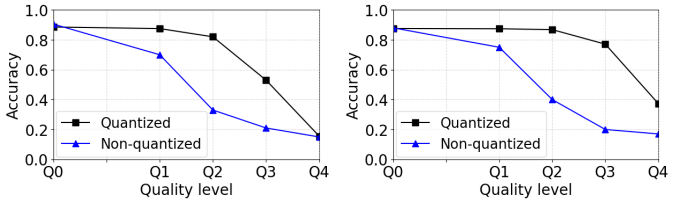
B. Key Factors

Several factors critically influence whether a sweet spot exists and what its characteristics are.

Quantization and input size. We employ 8-bit quantization, a standard practice when deploying DNNs on resource-constrained platforms [6], [23]. Beyond footprint benefits, quantization inherently enhances error resilience by restricting the numerical dynamic range, thus reducing the impact of bit flips [42]. In Fig. 7 we illustrate this effect. While non-quantized DNNs initially have higher accuracy than their quantized counterparts, they exhibit sharp accuracy degradation as early as Q1. In contrast, quantized models maintain stable performance across a wider range of quality levels, exposing error-tolerant operating points that non-quantized models cannot sustain. For instance, at QL1, the non-quantized FDMobileNet_128 suffers nearly a 30% drop in accuracy, whereas the quantized version experiences a negligible accuracy loss of less than 1%.

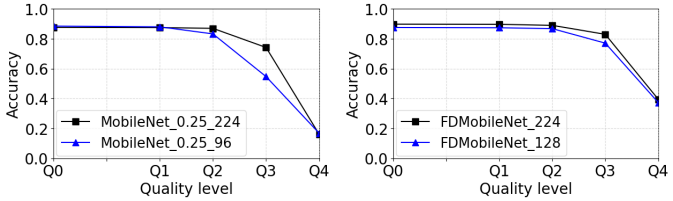
Similarly, in Fig. 8 we show that input size also affects how well DNNs handle write-induced errors. At lower quality levels like Q1, the accuracy drop is small and does not depend much on input size. However, as the error rate increases at Q2 and Q3, DNNs with smaller inputs tend to lose more accuracy. This happens because larger inputs carry more information and inherently offer greater data redundancy. This allows the network to tolerate partial corruption since even when some bits are flipped, the abundance of correct information in the intermediate outputs compensates for the errors, finally mitigating the accuracy loss. Still, under extreme degradation at Q4, all models fail, regardless of input size.

DNN models and platforms. The trends seen in Fig. 6 can be explained by examining the relative contribution of the terms in Eq. 1. Benchmarks like SqueezeNet emphasize computation phases and have comparatively small states. Thus, $E_{\text{MCUcompute}}$ dominates E_{total}, and reducing STT-MRAM write



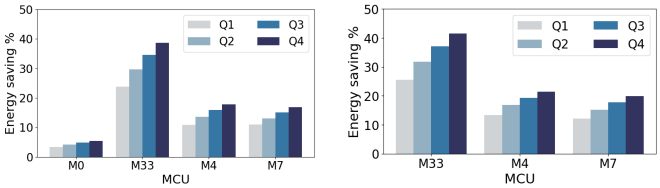
(a) MobileNet_0.25_96 (b) FDMobileNet_128

Fig. 7: Impact of quantization on neural network resilience to write errors. *Quantized DNNs experience significantly smaller accuracy drops compared to non-quantized versions under degraded write conditions, enabling the discovery of useful sweet spots.*



(a) MobileNet_0.25 (b) FDMobileNet

Fig. 8: Effect of input size on neural network robustness. *Networks operating on larger inputs show greater resilience to degraded storage conditions at higher quality levels, delaying accuracy drop.*



(a) Energy savings for STMINIST across Cortex M-class MCUs. (b) Average energy savings for Cortex M33, M4, and M7.

Fig. 9: Energy savings enabled by current scaling on state persistence operations. *Benchmarks and platforms with larger storage requirements relative to computation phases yield the highest energy gains, confirming that the most effective savings arise when $E_{storage}$ dominates $E_{MCUcompute}$.*

energy yields minimal benefit. Conversely, benchmarks such as MobileNet maintain large intermediate states and outputs, resulting in E_{storage} bearing a much greater impact on E_{total}. These benchmarks are the ones that reap the greatest rewards from optimizing storage energy.

Fig. 9 aggregates energy savings across MCUs. Fig. 9a shows the energy gains for STMINIST, the only DNN compatible across all Cortex M* cores. Despite having lower active power, the Cortex M0 achieves the least savings, as its limited instruction set prolongs execution time, dampening the effect of reduced storage energy.

Fig. 9b further shows that the Cortex M33 consistently obtains the best absolute energy savings. Cortex M4 and M7 offer similar results despite the Cortex M7 consumes ≈50%

more active power. In fact, the Cortex M7 advanced instruction set requires significantly fewer cycles per inference compared to the Cortex M4—roughly 1.4x fewer cycles—thus reducing the dominance of compute energy relative to storage energy, and maximizing the benefit of reduced persistence energy.

VI. Conclusion

We determined that tuning the write current of STT-MRAM in intermittent computing executing DNNs enables the identification of a sweet spot where up to 40% energy gains are available in exchange for a negligible degradation in accuracy. Together with this, we also studied how DNN and system settings, including quantization, input size, program, and target platform, impact these trends. The toolset we release [12] for others to use and build upon allows a similar investigation to be carried out on different programs and platforms.

Acknowledgments. This work is partially supported by the Swedish Science Foundation (SSF).

References

[1] N. A. Bhatti *et al.*, "Energy harvesting and wireless transfer in sensor network applications: Concepts and experiences," *ACM Transactions on Sensor Networks*, vol. 12, no. 3, 2016.

[2] N. Ikeda *et al.*, "Soil-Monitoring Sensor Powered by Temperature Difference between Air and Shallow Underground Soil," *Proc. of ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, vol. 4, no. 1, 2020.

[3] M. Afanasov *et al.*, "Battery-less zero-maintenance embedded sensing at the mithræum of circus maximus," in *Proc. of ACM Conf. on Embedded Networked Sensor Systems (SenSys)*, 2020.

[4] B. Denby *et al.*, "Kodan: Addressing the computational bottleneck in space," in *Proc. of Intl. Conf. on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2023.

[5] R. Barjami *et al.*, "Intermittent Inference: Trading a 1% Accuracy Loss for a 1.9x Throughput Speedup," in *Proc. of ACM Conf. on Embedded Network Sensor Systems (SenSys)*, 2024.

[6] G. Gobieski *et al.*, "Intelligence beyond the edge: Inference on intermittent embedded systems," in *Proc. of Intl. Conf. on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2019.

[7] S. Ahmed *et al.*, "The internet of batteryless things," *CACM*, 2024.

[8] K. Geissdoerfer *et al.*, "Riotee: An Open-source Hardware and Software Platform for the Battery-free Internet of Things," in *Proc. of ACM Conf. on Embedded Network Sensor Systems (SenSys)*, 2024.

[9] J. Van Der Woude *et al.*, "Intermittent computation without hardware support or programmer intervention," in *Proc. of USENIX Symp. on Operating Systems Design and Implementation (OSDI)*, 2016.

[10] A. M. Monazzah *et al.*, "Cast: Content-aware stt-mram cache write management for different levels of approximation," *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, vol. 39, no. 12, 2020.

[11] T. Devolder *et al.*, "Single-Shot Time-Resolved Measurements of Nanosecond-Scale Spin-Transfer Induced Switching: Stochastic Versus Deterministic Aspects," *Physical Review Letters*, vol. 100, 2008.

[12] "Stt-mram intermittent inference toolset," 2025, accessed: 2025-05-19. [Online]. Available: http://sttmram-infer.neslab.it

[13] B. Ransford *et al.*, "Mementos: System support for long-running computation on rfid-scale devices," *ACM SIGARCH Computer Architecture News*, vol. 39, no. 1, 2011.

[14] H. Jayakumar *et al.*, "Quickrecall: A hw/sw approach for computing across power cycles in transiently powered computers," *ACM Journal on Emerging Technologies in Computing Systems*, 2015.

[15] D. Balsamo *et al.*, "Hibernus: Sustaining computation during intermittent supply for energy-harvesting systems," *IEEE Embedded Systems Letters*, vol. 7, no. 1, 2015.

[16] B. Lucia *et al.*, "A simpler, safer programming and execution model for intermittent systems," in *Proc. of ACM SIGPLAN Conf. on Programming Language Design and Implementation (PLDI)*, 2015.

[17] S. Lee *et al.*, "Neuro.ZERO: a zero-energy neural network accelerator for embedded sensing and inference systems," in *Proc. of ACM Conf. on Embedded Networked Sensor Systems (SenSys)*, 2019.

[18] Y. Wu *et al.*, "Intermittent inference with nonuniformly compressed multi-exit neural network for energy harvesting powered devices," in *Proc. of ACM/IEEE Design Automation Conf. (DAC)*, 2020.

[19] C.-K. Kang *et al.*, "Everything leaves footprints: Hardware accelerated intermittent deep inference," *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, vol. 39, no. 11, 2020.

[20] C.-H. Yen *et al.*, "Stateful neural networks for intermittent systems," *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, vol. 41, no. 11, 2022.

[21] C.-C. Lin *et al.*, "Intermittent-aware neural network pruning," in *Proc. of ACM/IEEE Design Automation Conf. (DAC)*, 2023.

[22] S. Jeon *et al.*, "Harvnet: resource-optimized operation of multi-exit deep neural networks on energy harvesting devices," in *Proc. of the Annual Intl. Conf. on Mobile Systems, Applications and Services (MobiSys)*, 2023.

[23] B. Islam *et al.*, "Zygarde: Time-Sensitive On-Device Deep Inference and Adaptation on Intermittently-Powered Systems," *Proc. ACM on Interactive, Mobile, Wearable and Ubiquitous Tech.*, vol. 4, no. 3, 2020.

[24] S. Kannan *et al.*, "Energy aware persistence: Reducing energy overheads of memory-based persistence in nvms," in *Proceedings of the 2016 International Conference on Parallel Architectures and Compilation*, 2016.

[25] Z. Azad *et al.*, "An Efficient Protection Technique for Last Level STT-RAM Caches in Multi-Core Processors," *IEEE Trans. on Parallel and Distributed Systems*, vol. 28, no. 6, 2017.

[26] S. Mittal, "A survey of techniques for approximate computing," *ACM Computing Surveys (CSUR)*, vol. 48, no. 4, 2016.

[27] N. Sayed *et al.*, "Exploiting STT-MRAM for approximate computing," in *Proc. of European Test Symp. (ETS)*, 2017.

[28] A. Ranjan *et al.*, "Approximate storage for energy efficient spintronic memories," in *Proc. of Design Automation Conf. (DAC)*, 2015.

[29] S. Seyedfaraji *et al.*, "Extent: Enabling approximation-oriented energy efficient stt-ram write circuit," *IEEE Access*, vol. 10, 2022.

[30] A. Colin *et al.*, "A reconfigurable energy storage architecture for energy-harvesting devices," in *Proc. of Intl. Conf. on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2018.

[31] J. Hester *et al.*, "Flicker: Rapid prototyping for the batteryless internet-of-things," in *Proc. of ACM Conf. on Embedded Network Sensor Systems (SenSys)*, 2017.

[32] STMicroelectronics, "STM32AI Model Zoo - Image Classification Models," https://github.com/STMicroelectronics/stm32ai-modelzoo/tree/main/image_classification/models, 2023, accessed: January 22, 2024.

[33] M. Mohri *et al.*, *Foundations of machine learning*. MIT press, 2018.

[34] ARM Developer. (2024) Arm processors. Accessed: May 11, 2024. [Online]. Available: https://developer.arm.com/Processors

[35] J. Hester *et al.*, "Ekho: Realistic and repeatable experimentation for tiny energy-harvesting sensors," in *Proc. of ACM Conf. on Embedded Network Sensor Systems (SenSys)*, 2014.

[36] J. Yang *et al.*, "Clairvoyant: a comprehensive source-level debugger for wireless sensor networks," in *Proceedings of the 5th international conference on Embedded networked sensor systems*, 2007.

[37] C. Bolchini *et al.*, "Fast and accurate error simulation for cnns against soft errors," *IEEE Transactions on Computers*, vol. 72, no. 4, 2023.

[38] X. Dong *et al.*, "Nvsim: A circuit-level performance, energy, and area model for emerging nonvolatile memory," *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, vol. 31, no. 7, 2012.

[39] ST Microelectronics, "ST Microelectronics Boards," https://www.st.com/en/microcontrollers-microprocessors/, 2025, accessed: May 16, 2025.

[40] M. Thielen *et al.*, "Human body heat for powering wearable devices: From thermal energy to application," *Energy conversion and management*, vol. 131, 2017.

[41] M. Magno *et al.*, "Infinitime: Multi-sensor wearable bracelet with human body harvesting," *Sustainable Computing: Informatics and Systems*, vol. 11, 2016.

[42] Z. Chen *et al.*, "A low-cost fault corrector for deep neural networks through range restriction," in *Proc. of Intl. Conf. on Dependable Systems and Networks (DSN)*, 2021.