Intermittent Inference: Trading a 1% Accuracy Loss for a 1.9x Throughput Speedup

Rei Barjami
Politecnico di Milano
Italy

Antonio Miele
Politecnico di Milano
Italy

Luca Mottola
Politecnico di Milano, RI.SE, and
Uppsala University
Italy and Sweden

ABSTRACT

We present INTERCEPT, a compile-time toolchain enabling *manifold throughput improvements* when running intermittent DNN inference on IoT devices, in exchange of a *maximum 1% accuracy loss*. Intermittently-computing IoT devices rely on ambient energy harvesting and compute opportunistically, as energy is available. They use NVM to persist intermediate results in anticipation of energy failures. Without requiring changes to existing models and by exploiting the features of STT-MRAM as NVM, INTERCEPT optimizes the placement and configuration of state persistence operations when executing the inference process. This happens *off-line* with *no user intervention*, while enforcing a maximum 1% accuracy loss. Our results, obtained across three platforms and six diverse neural networks, indicate that INTERCEPT provides a 40% energy gain in a single inference process, on average. With the same energy budget, this yields a 1.9x throughput speedup.

CCS CONCEPTS

• **Computer systems organization** → **Embedded systems**.

KEYWORDS

Intermittent computing; deep neural network (DNN) inference; energy efficiency

ACM Reference Format:
Rei Barjami, Antonio Miele, and Luca Mottola. 2024. Intermittent Inference: Trading a 1% Accuracy Loss for a 1.9x Throughput Speedup. In *The 22nd ACM Conference on Embedded Networked Sensor Systems (SENSYS '24), November 4–7, 2024, Hangzhou, China.* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3666025.3699364

1 INTRODUCTION

Energy harvesting allows Internet of Things (IoT) devices to eliminate their dependency on regular batteries [10]. This reduces maintenance costs and enables multi-year unattended deployments [2, 19, 32]. However, energy from the environment is generally erratic, causing frequent and unanticipated energy failures.

Intermittent inference. Because of erratic energy patterns, executions become *intermittent*: intervals of active operation are interleaved by periods of recharging energy buffers, such as capacitors [5]. Energy failures normally cause a device to lose system state, as applications run on bare hardware without operating system support. To ensure forward progress across energy failures, systems employ *persistent state* on Non-Volatile Memory (NVM). The energy overhead of these operations is, however, often *enormous* [5].

Notwithstanding resource constraints, works exist that support the intermittent execution of Deep Neural Networks (DNNs) [24], as we discuss in Sec. 2. These provide the ability to locally execute the inference process, which allows systems to only transmit the response instead of raw data, improving energy consumption especially when using established wireless technologies, such as Bluetooth and 802.15.4 [5]. Most of these works apply techniques that either *i)* alter the model; for example, using compression [24], *ii)* operate run-time decisions to reduce the energy requirements; for example, using early exists [35], or *iii)* employ programming techniques to reduce processing demands [24].

INTERCEPT. We take a different stand that provides efficient system support for intermittent inference without requiring changes to existing DNNs. We use Spin-Transfer Torque Magnetic Random-Access Memory (STT-MRAM) in place of regular NVM technology. STT-MRAM has a key feature: one can tune the current used for write operations to *save energy*, but accepting that *write errors* may occur with increasing probability as current settings reduce [20].

The trade-off between energy gains and the Write Error Rate (WER) in STT-MRAM is the stepping stone for us to develop INTERCEPT (*INTERmittent inferenCE – Persist & Tune*): a compile-time toolchain that provides support for intermittent inference without user intervention. Its functioning is based on multiple stages, as shown in Fig. 1 and illustrated in Sec. 3. Given the DNN model, we first Profile its energy consumption using existing tools [4] or based on real hardware executions. This information is input to a Prepare stage that creates an initial configuration including placement of state persistence operations and corresponding STT-MRAM write current settings. Because we use a multi-capacitor architecture [17, 29, 66], based on the output of Prepare, a Configure stage determines the capacitor array that ensures eventual completion of the inference process.

Following the Configure stage, the Persist algorithm processes the initial configuration to determine an efficient placement of state persistence operations along the inference chain. The output of Persist is fed as input to a further optimization step, called Tune, that configures the STT-MRAM chip at each state persistence operation, determining the most efficient current setting. Due to *stochastic switching*, however, depending on this setting, a memory

SENSYS '24, November 4–7, 2024, Hangzhou, China

ACM ISBN 979-8-4007-0697-4/24/11.
https://doi.org/10.1145/3666025.3699364

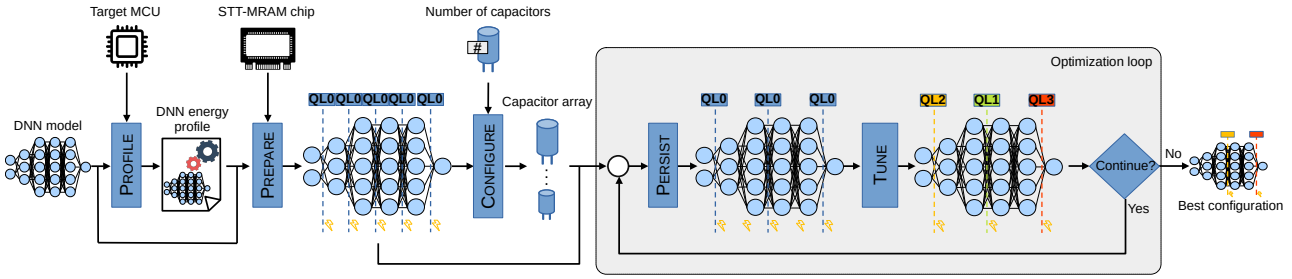


Figure 1: Overview of INTERCEPT. *PROFILE determines the energy demands of layer execution. PREPARE outputs an initial placement of state persistence operations and STT-MRAM write current settings. CONFIGURE determines capacitor sizes based on energy demands and target MCU. PERSIST determines an efficient placement of state persistence operations while favoring the use of smaller capacitors. TUNE determines the most efficient STT-MRAM current setting for each state persistence operation while keeping the accuracy loss within a given bound.*

cell may fail to commute to the new value, causing a write error. Since these errors are stochastic in nature, they appear randomly in written data. We instead embrace these errors and carefully control the current settings to reduce energy consumption, subject to a constraint that accuracy losses *do not exceed* a given bound.

The output of TUNE may potentially change the relative energy patterns during the execution of the inference process. This means, for example, that state persistence operations that consume a large fraction of available energy before applying TUNE may become much more energy-savvy, compared to other state persistence operations that now grow to be dominating. Because of this, we feed the output of TUNE back to PERSIST to re-evaluate the number and positioning of state persistence operations. This effectively closes an optimization loop that continues until we obtain a (possibly local) optimal configuration or for a predetermined number of repetitions. The code is then accordingly instrumented.

Benefits and insights. We use accurate simulations to measure the performance of INTERCEPT. This is common in intermittent computing because there is no network component, which is where simulation models often fail miserably [42], and because of the challenges in ensuring repeatability and fair conditions across the system under test and the baselines [27]. At the core of our setup are energy profiles measured on real hardware, similar to multiple existing works in the area [33, 34, 38, 48]. We examine the performance of INTERCEPT across three different platforms and six diverse neural networks, compared with the original unmodified DNN in a configuration that ensures completion of the workload and our own PERSIST stage applied in isolation.

The experimental results we collect, reported in Sec. 4, demonstrate that INTERCEPT enables great energy gains, and corresponding throughput improvements, in exchange for a maximum accuracy loss of just 1%. Compared to the baseline, energy gains range from 21% to 64.4%, with throughput speedups of 1.36x to 2.98x. Nonetheless, the optimization loop improves the energy performance of a single round of PERSIST by 29.65% on overage, yielding an average 1.44x throughput speedup. The 1% bound on accuracy loss is arguably immaterial for most applications and is usually "lost in noise" [52]. It is also smaller compared to many other techniques that trade accuracy for energy efficiency in DNN inference [26, 64].

The *fundamental insight* we gain is that the inherent data resilience of DNNs is an asset to mitigate data errors. We leverage this by tuning STT-MRAM write currents to optimize energy gains while limiting accuracy loss. We end the paper by offering a few key observations, also on the choice of the 1% accuracy bound, in Sec. 5, and with brief concluding remarks in Sec. 6.

2 RELATED WORK AND BACKGROUND

We briefly survey existing work in intermittent inference and provide background information on STT-MRAM technology and on error resilience of DNNs.

2.1 Intermittent Inference

Several works investigate the execution of DNN models on low-power microcontroller units (MCUs) [56] and in intermittent systems [46]. Our work is *orthogonal* to most of these techniques. We do not require or apply changes to the network or inference process, and rather execute completely *unchanged* models, thus benefiting from the existing training. We carefully optimize both the location and the configuration of state persistence operation when using STT-MRAM as NVM, which is arguably under-explored.

Works exist that apply a variety of compression techniques and design multi-exist network architectures. As an example, Wu et al. [64] design a network compression algorithm working with multi-exit DNNs that selects exits based on energy predictions. Given an existing multi-exit network, the notion of approximate intermittent computing [9] allows the processing to step out of the inference process before state persistence operations are necessary.

Model augmentation and pruning are often employed to run intermittent inference. Kang et al. [40, 41] and Yen et al. [65], append components to existing models to allow progress tracking information to be piggybacked onto output features. Without affecting accuracy, this allows the system to efficiently recover the inference process after an energy failure. iPrune [44] embeds an ad-hoc pruning strategy that produces compact models for intermittent systems, whereas RAD [36] employs block circulant matrices and structured pruning to exploit vector operation accelerators.

Network architecture search is also investigated for intermittent inference. HarvNet [39] includes two complementary techniques, one enabling architecture search that optimizes multi-exit features based on memory and energy constraints, the other returning efficient inference policies by taking into account energy constraints. iNas [50] seeks to strike a trade-off between data reuse and energy overhead of state persistence operations while ensuring forward progress and eventual completion of the inference process. EVE [37]

uses custom network search algorithms to produce different models, enabling run-time selection based on energy constraints.

In other works, special-purpose execution and scheduling techniques enable efficient inference. To reduce the energy overhead of state persistence, Lv et al. [47] slice the network horizontally and execute each slice in a depth-first manner, only saving a fraction of the state on NVM. Zygarde [35] models the energy patterns together with the relation between accuracy and processing requirements.

Closer to our efforts are works that combine software and hardware support for intermittent inference. As an example, Gobieski et al. [24] present a concept of loop continuation that reduces the overhead of frequent state persistence operations during inference. Neuro.ZERO [43] uses a co-processor architecture to improve the energy performance of the inference process running on a main MCU; the co-processor runs special-purpose models that account for intermittent executions already during training. A few works [6, 54, 55] also employ in-memory computing to enable parallelized executions or to improve the efficiency of state persistence operations on special-purpose accelerators [14].

2.2 STT-MRAM

Manufacturers including Avalanche, Everspin, and Renesas produce 2MiB STT-MRAMs chips for under $10. The literature extensively considers the problem of stochastic switching with this memory technology and the resulting write errors. Besides ensuring correctness using conservatively high currents, works exist that seek to ensure data integrity through error correction codes [8], at the cost of increased energy consumption. In contrast, we embrace write errors in exchange for energy gains, leveraging the inherent error resilience of DNNs to limit their impact on inference accuracy.

This resonates with existing techniques [51] that introduce controlled data errors to save energy. For example, several works [1, 53, 58, 59] define architectures for caches or scratchpad memories supporting current tuning. For example, CAST [1] exposes a driver that defines a set of Quality Levels (QLs), each corresponding to a write current setting and annotated with a pre-characterized write error probability. Most of these works focus on the definition of architectural aspects, presenting a limited experimental investigation of the corresponding performance improvements.

We borrow the concept of QLs from CAST. We define five different QL settings, from `QL0` to `QL4`, corresponding to different write error probabilities and energy consumption. The `QL0` setting is the highest current setting and ensures that write errors are (probabilistically) guaranteed *not* to happen. This also corresponds to the highest energy consumption. At the opposite extreme, the `QL4` setting yields the highest write error probability, yet with the highest energy saving. The number of specific settings for each QL is up to developers to determine anyway. A higher number of QLs provides finer granularity in the optimization process and therefore better final performance, at the cost of longer processing times at compile-time because the solution space grows. Our specific choice strikes a trade-off between the two requirements.

2.3 Error Resilience in DNNs

Extensive literature [3] studies hardware faults in DNN execution and, in particular, the related data errors. The general observation

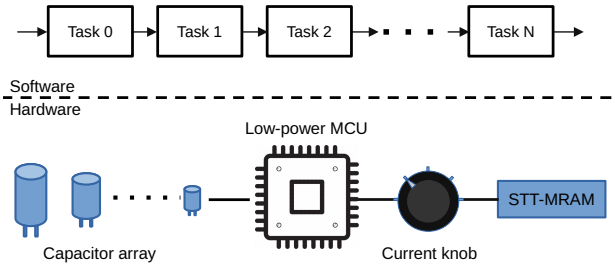


Figure 2: Target system architecture. *DNNs are encoded as a sequence of tasks. State persistence operations are interleaved with task executions. The MCU is powered by a multi-capacitor architecture [17]. The STT-MRAM chip provides a knob to set the write current.*

is that the large information redundancy in the model and weights give DNNs high error resilience, that is, the DNN can produce the correct outcome even if data errors corrupt the processing. This capability is widely exploited [7]; systems are modified at hardware or software level to elaborate data in a slightly inexact way, aiming at reducing resource consumption at the cost of a limited accuracy loss. In exchange for energy gains, in this work, we let write errors happen when persisting state on STT-MRAM.

Our specific design choices rest on a two-pronged basis. DNN targeting resource-constrained devices are normally quantized to meet memory, processing, and energy requirements. Nonetheless, quantization makes the networks more robust. Hoang et al. [30] demonstrate that the narrower the range that a value in the intermediate layer can assume, the more this layer is robust to errors. With 8-bit quantization, as in intermittent inference [64], intermediate values assume a smaller range compared to their non-quantized counterpart. The limited range acts as a hardening mechanism [15].

Ibrahim et al. [31] also show that when injected with errors, different layers of a DNN bear a different impact on accuracy. They identify the most critical layers and harden their values, increasing the network's error resilience. Tune is based on the same insight: we can afford higher QLs for layers that are robust to errors, saving energy, whereas layers that are more susceptible to errors should use lower QLs, not to severely impact the inference accuracy.

3 INTERCEPT

This section describes INTERCEPT. We overview the target architecture and the optimization process first; then we detail the single steps. We conclude with a quantitative example.

3.1 Overview

Target system. Fig. 2 shows the system architecture we target. We consider a resource-constrained MCU and use STT-MRAM to persist the system state. We map a single DNN layer to a task in a task-based programming system [45]: state persistence operations are placed in the code at task boundaries to dump their output on NVM, ensuring forward progress. In our case, the output of a task is the output tensor of a layer.

We base our design on existing multi-capacitor architectures [17]. These ensure better energy efficiency than single-capacitor configurations while ensuring eventual completion of the workload [17, 29, 66]. When used in combination with task-based programming systems, the key benefit of a multi-capacitor architecture is the ability

to accommodate different energy demands for different tasks. This is found across layers within a DNN too, as we show experimentally in Sec. 4. This variability depends on the workload and prompts for customized capacitor sizing.

For a layer x, we define $E_{compute}(x)$ as the energy consumption required for processing. Note that $E_{compute}(x)$ is largely input-independent, as the processing time of deep learning operators does not generally depend on input data. It is, instead, dependent on both the specific DNN layer and the underlying MCU. Similarly, we define $E_{persist}(x, ql)$ as the energy required to persist the output tensor of a layer x at a given quality level ql. This quantity varies based on the selected QL. The energy consumption to compute a sequence $1..n$ of layers and persist the final output tensor at a specific QL level is therefore

$$E_{exec}(n) = \sum_{i=1}^{n} E_{compute}(x_i) + E_{persist}(x_n, ql_n) \quad (1)$$

We describe next how to quantify $E_{compute}(x)$ and $E_{persist}(x, ql)$.

Preprocessing. Fig. 1 shows the INTERCEPT compile-time operation. It takes as input the DNN source code. We feed this as input to a PROFILE step that quantifies $E_{compute}(x)$ for all layers in the DNN. This quantity primarily depends on the target MCU and may be quantified using existing tools [4] or by hardware profiling of un-modified DNNs, as we do in Sec. 4.

In a PREPARE step, we build an internal representation of the optimization problem. For each layer x and quality level ql, we append $E_{compute}(x)$ from the previous step and $E_{persist}(x, ql)$. The latter depends on the specific STT-MRAM chip and may be obtained based on a combination of datasheet information, hardware profiling, or NVM simulators. We finally create an initial configuration where we persist each tensor on STT-MRAM using a `QL0` setting. This setting ensures that write operations are (probabilistically) correct. When the inference process eventually completes, it offers the same accuracy as a non-intermittent execution.

Given the number of capacitors the target architecture accommodates [17, 28, 29], the CONFIGURE step determines the size of each of the given capacitors *before* the optimization loop. This process is not different compared with capacitor sizing for other workloads [16]. One of the available capacitors must necessarily be large enough to store the energy required by the most energy-demanding task (layer) of the *original DNN* and to persist its output tensor. We size the other capacitors based on the expected energy demands of the remaining tasks (layers), favoring smaller capacitors to reduce charging times and leakage [2, 17, 29], while ensuring sufficient energy for eventual completion of the inference process. Capacitor sizing remains *unchanged* throughout the optimization loop.

Note that our design is independent of the number of capacitors. INTERCEPT may work with an arbitrary number of capacitors or even just one. Determining the number of capacitors of the target architecture is an orthogonal problem whose solution must consider not just the workload but also fabrication challenges and physical footprint [16, 17, 28, 29]. The results in Sec. 4 are based on a three capacitor setup as it is most common in available platforms and deployed systems [2, 17, 28].

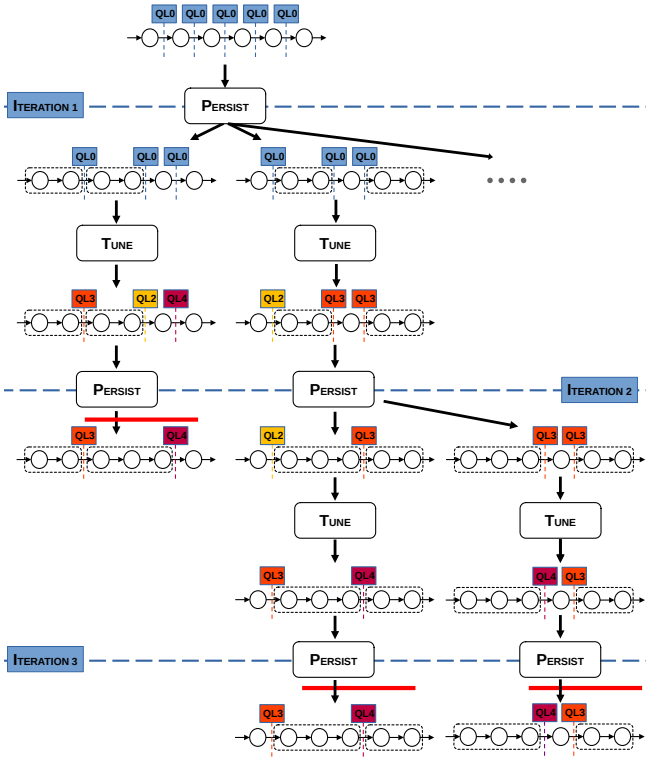


Figure 3: Example optimization loop. *The interplay between* PERSIST *and* TUNE *reduces the energy overhead due to state persistence operations while keeping the accuracy loss within the specified bound.*

Optimization loop. The right-hand side of Fig. 1 shows the core of the optimization loop. Fig. 3 depicts an example of how the optimization process unfolds. PERSIST uses a heuristic algorithm to modify state persistence operations, aiming to *i)* reduce energy consumption and *ii)* favor the use of smaller capacitors among those available, while retaining the guarantee of eventual completion. Smaller capacitors, indeed, recharge faster and experience reduced leakage. PERSIST produces multiple configurations as output, which are equivalent "by construction" in consumed energy but with different placements of state persistence operations. In Fig. 3, PERSIST produces two configurations at ITERATION 1. The execution branches out for each such configuration, using separate copies of the internal problem representation.

Every new configuration generated by PERSIST is separately taken as input by TUNE, whose goal is to determine an efficient QL setting for each state persistence operation. The application of TUNE affects the accuracy of the DNN. Varying QL setting yields different probabilities of write errors in STT-MRAM that, in turn, may differently impact the output accuracy. We let TUNE push the QL setting as much as possible but without violating a given accuracy bound, which we set in our experiments to a mere 1% [52].

By setting different QLs for each state persistence operation, TUNE changes the energy patterns for that specific configuration. As the energy demands reduce because write operations occur with higher QLs, we obtain two effects: *i)* some state persistence operations may be skipped because the execution has sufficient energy to continue processing the next layer and persist the state

later, and *ii)* an earlier choice of what capacitor to use for what state persistence operation may no longer be optimal; as energy demands reduce, smaller capacitors may be used. To account for the new energy patterns, we feed the output of TUNE back to PERSIST, effectively closing the loop.

Note that PERSIST may reduce the accuracy loss determined by TUNE in a previous iteration. This happens because PERSIST may eliminate a subset of state persistence operations, which are also the points where write errors may occur. As a result, those errors no longer have an impact, providing further room for energy improvements when the same configuration is fed as input to TUNE again. Therefore, the process repeats, as shown in Fig. 3, for every branch that PERSIST possibly creates every time it runs, as it happens for example in ITERATION 2 of Fig. 3.

The process terminates along a branch whenever PERSIST yields the same configuration taken as input, indicating that no more gains are attainable. The whole process ends when all branches terminate. Among all candidate configurations eventually produced when every branch of the execution completes, we select the one that maximizes the energy gains.

As the energy patterns change after applying INTERCEPT, we may revise capacitor sizing as output by CONFIGURE and possibly further improve performance by employing smaller capacitors. We may even try and embed the CONFIGURE step within the optimization loop and revise capacitor sizing at every iteration, possibly amplifying the gains. However, doing so might prolong processing times. We include exploring this opportunity in our immediate research agenda. We do not consider changing the number of capacitors instead, as that might have repercussions on the target architecture that exceed the scope of our work.

3.2 PERSIST

PERSIST places state persistence operations based on a two-pronged rationale: *i)* execution of a layer occurs only if the residual energy at the start of the execution is sufficient to process the layer and persist the output tensor on STT-MRAM, and *ii)* in a multi-capacitor setting, smaller capacitors should be favored instead of larger ones. Similar to existing work [2, 11, 17], step *i)* ensures both transactional semantics for layer execution and avoids wasting energy in processing a layer without the guarantee to eventually persist the output tensor. Differently and although PERSIST is applicable also in single-capacitor systems, the use of smaller capacitors improves overall energy efficiency [29], as discussed earlier.

The goal of PERSIST is: *i)* to *group* together the execution of multiple subsequent layers as a single task to reduce the overall number of state persistence operations, and *ii)* select the smallest capacitor sufficient to support the entire task execution, including state persistence operations at the end. Grouping layers is possible whenever the residual energy at the end of a given layer's execution is sufficient to skip the state persistence operation for that layer and execute both the next layer and persist its output tensor. The energy required to do so is computed as in Eq. 1.

PERSIST operates in a greedy manner, as illustrated in Fig. 4. The existing layer groups, included in a list G_{list}, are examined in topological order. In the initial configuration generated by PREPARE, each layer represents a separate group. We start by computing the energy

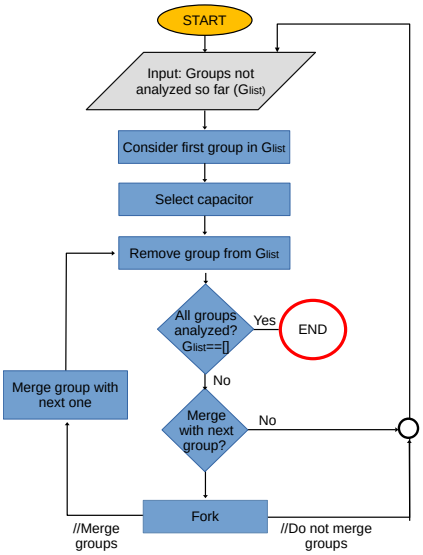


Figure 4: Operation of PERSIST. *The loop tries to extend the currently analyzed group, while forking branches to analyze both the case where the group is extended and where it is not.*

consumption $E_{exec}(g_1)$ for the execution of the first group g_1 using Eq. 1 and select for g_1 the smallest capacitor with an energy budget $E_{minCap}(g_1) \geq E_{exec}(g_1)$. Then, we check if it is possible to merge g_1 with the next group g_2; we compute the new energy consumption $E_{exec}(g_1 + g_2)$ and check whether $E_{minCap}(g_1) \geq E_{exec}(g_1 + g_2)$.

The latter condition determines whether merging g_1 with the next group g_2 is feasible. If so, PERSIST forks in two, and the internal problem representation is accordingly duplicated. In one branch, we merge g_1 with g_2 and proceed to analyze the next group g_3 in the same way. In the other branch, we keep group g_1 intact and restart the process from g_2, attempting the merge with the subsequent group g_3. If merging is not feasible, only the latter branch proceeds.

The process continues through the entire DNN until all groups in G_{list} are analyzed. When all branches created out of forking conclude, we only keep the configurations with minimum total energy consumption. There may be multiple such configurations. By construction, these configurations include the same number of state persistence operations and use the same capacitor the same number of times. Only the placement of such operations along the DNN may differ. Each such configuration yields a different branch in the execution of the optimization loop, as seen in Fig. 3.

3.3 TUNE

Given the placement of state persistence operations returned by PERSIST, the goal of TUNE is to determine the most efficient QL setting for each such operation. By doing so, we seek energy gains at the cost of lowering the output accuracy, due to write errors when persisting the tensors on STT-MRAM. TUNE limits the accuracy loss by staying within a bound provided by system designers. Its goal is thus to find a solution to a specific optimization problem.

Problem formulation. Given layers $x_1..x_m, m \leq n$ whose output tensors must be persisted according to the output of PERSIST, the

goal is to identify a set of quality levels $ql_1..ql_m$ such that

$$\begin{aligned} &\min \sum_{i}^{m} E_{persist}(x_\text{i}, ql_\text{i}) \\ &\text{while } accuracyLoss\% \leq k \end{aligned} \tag{2}$$

where

$$accuracyLoss\% = \frac{accuracy(ql0) - accuracy(ql_1..ql_n)}{accuracy(ql0)} \cdot 100 \tag{3}$$

Note the objective function in (2) only considers state persistence operations since the compute part remains unchanged. Instead, k is the maximum tolerated accuracy loss, provided by system designers. In (3), we indicate with $accuracy(ql0)$ the accuracy of the DNN using the configuration at Iteration 0 of Fig. 1, which provides the same accuracy as a non-intermittent execution.

We use genetic search to find a solution to the problem. The optimization problem here is similar to many other problems dealing with the identification of best operating points in embedded systems, such as task mapping, scheduling, or CPU voltage/frequency tuning. Existing literature shows that genetic algorithms are efficient and accurate in finding optimal solutions [13, 25, 63].

Genetic search. We encode the chromosome of each individual as a string where each gene is associated with a state persistence operation and indicates the corresponding QL setting. We select a classical single-point crossover operator for mating, a uniform random mutation operator to perturb chromosomes after mating, and a steady-state approach to pick parents for mating. In preliminary experiments, this demonstrates fast convergence with small population sizes; this is fundamental in our case since the evaluation of the fitness function is rather expensive, as discussed next.

We formulate a fitness function that embeds both the objective function and the constraint of Eq. 2 as follows

$$R = \begin{cases} E_{gain} & \text{if } accuracyLoss\% < k \\ E_{gain} - accuracyLoss\% \cdot E_{ql0} & \text{otherwise} \end{cases} \tag{4}$$

with E_{gain} is computed as

$$E_{gain} = \sum_{i}^{m} E_{persist}(x_\text{i}, ql0) - \sum_{i}^{m} E_{persist}(x_\text{i}, ql_\text{i}) \tag{5}$$

the left-hand term sum is the energy consumption for state persistence operations to guarantee the same accuracy as a non-intermittent execution, whereas the right-hand sum is the same energy figure but for the specific individual.

Note that the upper part of Eq. 4 returns the reward value for individuals not violating the accuracy constraint; the larger the energy saving, the higher the reward. We use the bottom part of Eq. 4 to rank individuals violating the constraint; in this case, the larger the violation, the more that specific individual is penalized.

As a result of the specific formulation of the reward, QL settings that reduce the accuracy by more than k are heavily and proportionally penalized, yet not completely excluded. This allows the genetic search to explore slices of the solution space with a limited, yet greater than k reduction in accuracy. This is useful to avoid getting stuck in local minima and generally improves the effectiveness of the genetic search by allowing it to explore a broader range of potential solutions and to maintain diversity in the solution space.

Computing $accuracy(ql_1..ql_n)$ requires careful consideration of write errors, which are stochastic in nature. For a single execution of the inference process with given inputs, we must consider multiple different occurrences of write errors to eventually obtain a statistically significant estimate of the accuracy we may obtain with a given QL setting. This is computationally heavy, which motivates our choice of a limited population size illustrated earlier.

Nonetheless, in the specific case of $k = 1$, the algorithm is guaranteed never to output a configuration that reduces accuracy more than *one* percentage point. In the extreme case where no QL setting achieves any positive energy gain by limiting the accuracy loss by *one* percentage point, the algorithm eventually outputs the configuration with a `QL0` setting. In this case, in fact, for any possible QL setting, the upper part of R becomes negative, as $accuracyLoss\% > 1$. The search then converges to a `QL0` setting, where $E_{gain} = 0$, which is to be preferred over any negative quantity.

3.4 Quantitative Example

We complete the discussion with a quantitative example of the optimization loop, shown in Fig. 5 and taken from a real execution of INTERCEPT with Mobilenet_96 and targeting an M33 MCU. The input configuration is the one produced by the Configure step for a three-capacitor setup. The capacitors are sized as 13 uF, 6 uF, and 3 uF. The system incurs in a total of 33 state persistence operations using a `QL0` setting, ensuring the same accuracy as a non-intermittent execution. Fig. 5 shows the execution unfolding along the branch that eventually leads to the configuration achieving the highest energy gain among the nine branches created during the execution. For a layer x, the blue color indicates $E_{compute}(x)$ and the yellow color indicates $E_{persist}(x)$.

At the start of Iteration 1, Persist processes the initial configuration and, based on the processing in Sec. 3.2, reduces the number of state persistence operations to just 10, with a 44% energy gain. This is achieved solely by removing state persistence operations, yet the energy consumption of those that remain stays the same. Similarly, inference accuracy is unchanged because the remaining state persistence operations still operate at a `QL0`.

The output of Persist at Iteration 1 is fed to Tune, which optimizes the QL settings within a $k = 1\%$ accuracy bound. In this case, Tune yields another boost in energy gain, as it shaves another 17.5% of the initial energy at the cost of a marginal accuracy loss that moves from 0.88 to 0.873. We show the energy gain due to Tune at Iteration 1 with a reduction in the height of the yellow bars. This gain actually depends on the specific layer, as some layers experience higher energy gains than others.

The gains obtained by Tune at the end of Iteration 1 allow Persist to eliminate additional state persistence operations in Iteration 2, for example, because a reduction in the energy for persisting output tensors enables new layer groups. Persist specifically removes the state persistence operations for layers 1 and 11, reducing their total number from 10 to 8. This shaves another 1.1% of the initial energy consumption and slightly increases the accuracy, as some STT-MRAM write operations, which are potential sources of data errors, are now completely removed. The new configuration is now fed again back to Tune. The changes are minimal now, with an increase to the QL setting for layer 16, which changes from `QL2` to

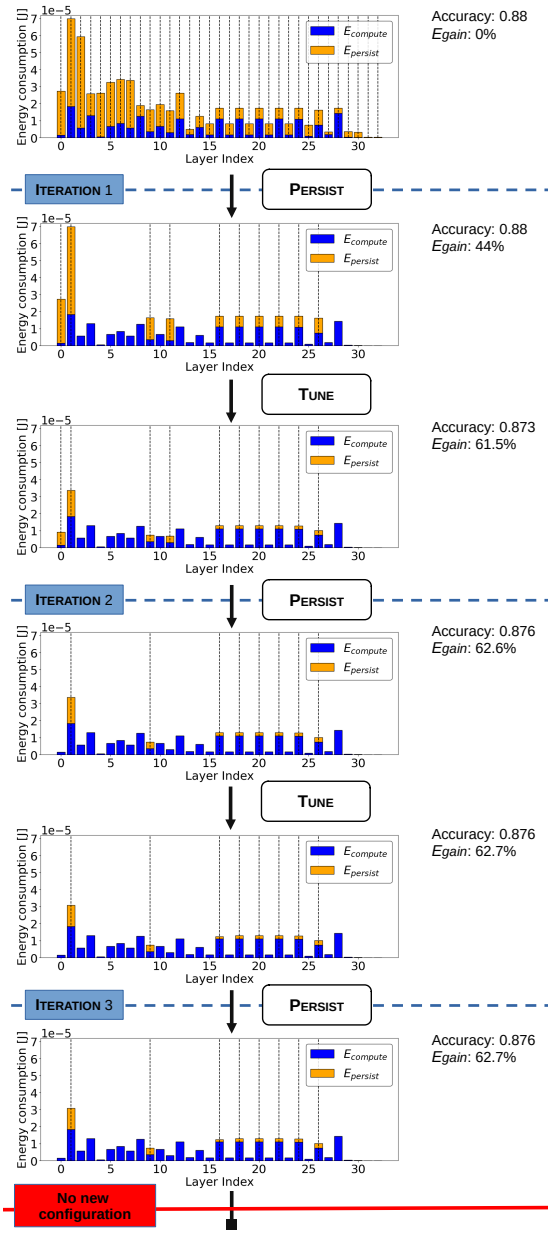


Figure 5: INTERCEPT execution on Mobilenet_96. *We note a gradual reduction of state persistence operations and a subsequent tuning of corresponding QLs.*

QL3. This leaves the output accuracy unchanged and gains another 0.1% in energy consumption compared to the initial configuration.

At the start of ITERATION 3, we feed the configuration back to PERSIST, which outputs the same configuration again in that it finds no opportunities for further energy gains. We reach the termination condition that concludes with an energy gain of 62.7% compared to the initial configuration, with just a 0.4% accuracy loss. Besides this example, in Sec. 4 we dig deeper into the relative contribution of PERSIST and TUNE to the total energy gains.

Table 1: DNN models. *We employ a diverse set of DNNs as benchmarks, to provide evidence of general applicability.*

Model	**Workload** [M Multiply and Accumulates (MACCs)]	**Total tensor size** [KiB]
Mobilenet_96	7.511	396
Mobilenet_224	41.092	2134
FDMobilenet_128	3.953	126
FDMobilenet_224	12.099	385
SqueezeNet	81.244	1122
IGN_wl_48	0.052	3.19

4 EVALUATION

We report on the energy gains we obtain by limiting the accuracy loss to a 1% factor, and on the corresponding throughput improvements. We end with a microscope analysis of the individual contributions of PERSIST and TUNE to the energy improvements and with a summary discussion on the processing times of INTERCEPT. Compared with the original model in a necessary configuration that ensures completion of the inference process, further described next, our results indicate that:

1) within the 1% bound on accuracy loss, INTERCEPT yields an average 40% energy gain in a single inference process;
2) notwithstanding the 1% bound, the actual accuracy loss of the INTERCEPT-optimized DNNs is often much lower;
3) the energy gains enabled by INTERCEPT enable a 1.9x throughput increase;
4) depending on the DNN architecture, PERSIST or TUNE contribute differently to the overall energy gain.

Before illustrating the experimental results, we describe the corresponding setup.

4.1 Experimental Setup

Benchmarks and platforms. Similar to existing works [18], we pick DNN models from the STM Model Zoo suite [62]. Tab. 1 summarizes their features, mainly defined by their computational requirements. The models also vary in the dimension of the output tensors, which impacts both computational requirements and memory usage. This diversity ensures our findings are robust across different DNNs architectures.

We consider several Cortex M MCUs on top of STM boards, as summarized in Tab. 2. The MCUs we consider are characterized by different instruction sets and energy figures, adding to the generality of the results. Due to memory limitations, FDMOBILENET_224, MOBILENET_224 and SQUEEZENET cannot run on the M4 MCU.

Capacitor selection. We use a three-capacitor configuration as it is most common in available platforms and deployed systems [2, 17, 28]. As for their sizing, each combination of DNN and MCU exhibits different energy patterns for different layers. In our prototype, the CONFIGURE step sizes each of the three available capacitors for each combination of DNN and MCU by applying the following rationale:

- The first capacitor C_1 is sized to support the most energy-intensive layer; its capacity is set to provide the energy required to execute its processing and persist its output tensor.
- The second capacitor C_2 is sized to support the least energy-intensive layers; we find that 57.5% of the layers in the DNNs

Table 2: MCUs. *The choice of MCUs is representative of different instruction sets and energy figures.*

MCU	Board	Active Power (uW/MHz)	Main Memory [KiB]
Cortex M33	NUCLEO-U575ZI-Q	12	786
Cortex M4	B-L475E-IOT01A2	32.82	128
Cortex M7	NUCLEO-H743ZI	58.5	1024

Table 3: Characterization of QLs for a 2MB 32nm scratchpad STT-MRAM. *Write energy decreases linearly; the corresponding WER increases by orders of magnitude.*

QL	WER	Set current (μA)	Write energy/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

we consider consume at least 4 times less energy than the most energy-demanding layer, thus, we set C_2 to 25% of C_1.

- The third capacitor C_3 strikes a trade-off between the two extremes; we find that 30.2% of the layers in the DNNs we consider consume less than half of what C_1 can store but more than what C_2 can store, thus, we set C_3 to 50% of C_1.

As mentioned in Sec. 3, capacitor sizing remains unchanged throughout the optimization loop.

Comparison. We compare INTERCEPT with a baseline configuration where *all output tensors* are persisted on STT-MRAM using a `QL0` setting. As write operations are probabilistically correct, the accuracy is the same as a non-intermittent execution. This baseline reflects the case where developers must ensure that the accuracy of the original model is retained regardless of energy concerns.

We also compare INTERCEPT with a single run of our own Persist stage, called PersistOnce, using the same inputs as INTERCEPT. With this, we obtain a configuration where the output tensors are persisted at `QL0`, thus not affecting the DNN accuracy. PersistOnce is useful to understand the performance gains obtained by looping through Persist and Tune, and independent of NVM technology as it may operate on any such kind of memory.

Metrics. We primarily measure two quantities: *energy consumption* and *throughput improvement*. Energy consumption is the net amount of energy required to complete an inference process, including both processing and STT-MRAM operations. We measure the energy invested in processing by looking at the execution trace given as input to the optimization loop.

We compute the throughput improvement by considering that recharge times vastly dominate active times in intermittent systems [10]. Although absolute throughput values are inherently a function of the specific energy patterns [10], we can compute the corresponding improvements by normalizing this figure to the throughput of the baseline. The improvement is thus a function of energy consumption and capacitor usage. We consider the smallest capacitor used by the baseline as a unit of charge and normalize the use of all other capacitors in the INTERCEPT-optimized version to that. Note this is, instead, *independent* of the specific energy patterns; the corresponding conclusions, therefore, apply regardless and across different energy sources.

Table 4: Tune genetic search parameters. *We empirically tune the algorithm to allow reasonably fast convergence.*

Parameter	Value
Population size	10
Crossover probability	60%
Mutation probability	10%
Stopping condition	No improvement on 5 gens.

A different reasoning applies to examining the *accuracy* of the inference process. An upper-bound on this quantity, which we call k and set to 1 hereafter, is given as input to the optimization process described in Sec. 3 and is at the core of the functioning of Tune. The solution eventually returned to the user is guaranteed to stay within this bound. However, this does *not necessarily* mean that the solution loses *exactly* k percentage points of accuracy. The set of QLs we consider is finite and limited, for example. This means that a solution may use a set of QLs that makes it lose less than k percentage points of accuracy, provided no other solution is found that offers higher energy efficiency within the same bound. We examine this aspect next to energy consumption.

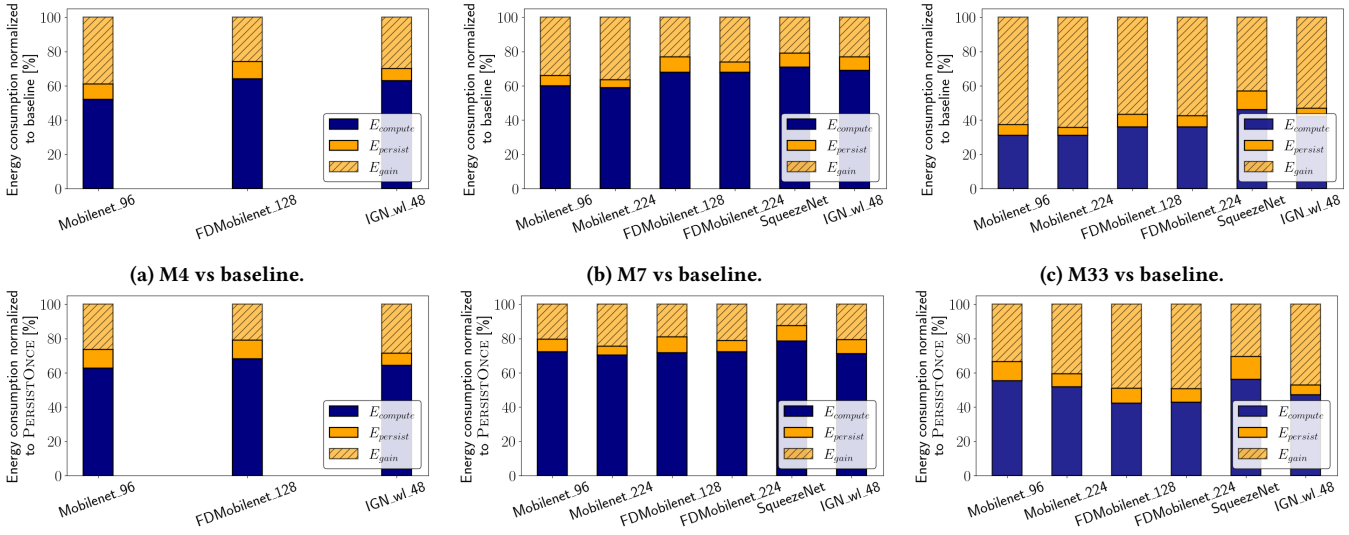
Simulation design. Our setup enables measuring the system performance in practical time across the many combinations of MCU, DNN, and systems under test, while taming potential sources of inaccuracy. The energy consumption for the execution of each layer of the DNN is taken from real hardware using the CYCCNT register available on the Cortex M MCUs to count the number of cycles required for each layer execution and relating this to energy consumption as in existing works [33, 34, 38, 48].

We use NVSim [21] to obtain the energy consumption figures for STT-MRAM and the corresponding WER. NVSim is a widely employed [1, 57] simulator that allows system designers to characterize the operation of NVM chips. Using NVSim, we profile a 32nm 2MB STT-MRAM chip powered at 0.9V. Its energy consumption figures and corresponding WER are in Tab. 3. Write energy decreases linearly, whereas the corresponding WER increases by orders of magnitude. This is a peculiar feature of STT-MRAM technology and requires careful consideration of the related trade-offs [1].

Auxiliary circuitry generally mounted on a board, such as low-dropout regulators necessary to adapt capacitor voltage to the various chips, consume at least three orders of magnitude less power than major components, such MCU and NVM [28, 60, 61]. These components impact the systems under test in the same way, or would even limit their impact in INTERCEPT compared with the baseline [61]. We choose not to account for these factors.

The fundamental threat to validity in our setup is that the energy harvested from the environment during the very short computing phases (up to a few ms) is assumed not to affect the duration of the computing phase. Based on existing work and available power traces [23], we argue that the energy harvested during these short periods is indeed largely insufficient to extend their duration and/or drastically change the execution pattern.

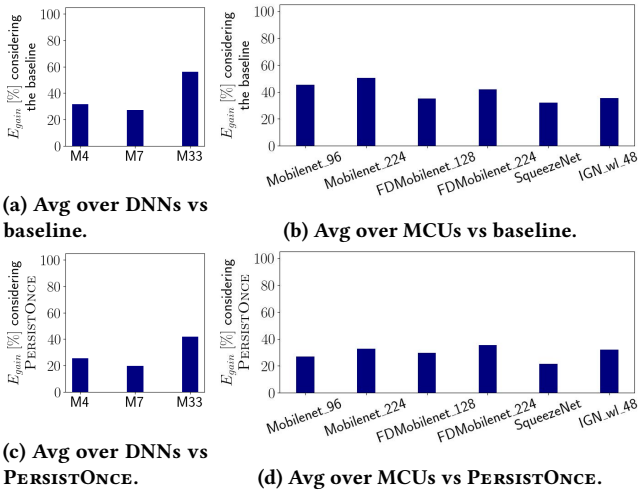
Prototype. INTERCEPT is written in Python, processing TensorFlow DNN models, based on STM Model Zoo implementations. We



(a) M4 vs baseline. (b) M7 vs baseline. (c) M33 vs baseline.

(d) M4 vs PersistOnce. (e) M7 vs PersistOnce. (f) M33 vs PersistOnce.

Figure 6: Energy gains compared with the baseline and PersistOnce with a 1% bound on accuracy loss, depending on MCU. *The gains vary from 64.4% to 12.7%. The gains are due to reducing the cost of state persistence operations.*



(a) Avg over DNNs vs baseline. (b) Avg over MCUs vs baseline.

(c) Avg over DNNs vs PersistOnce. (d) Avg over MCUs vs PersistOnce.

Figure 7: Energy gains of INTERCEPT-optimized DNNs with a 1% bound on accuracy loss, averaged over MCUs and DNNs. *INTERCEPT is beneficial independent of the target MCU and DNN.*

implement the genetic search for Tune by means of the PyGAD library [22] and use the parameters in Tab. 4.

To compute the reward defined in Sec. 3.3, we need to evaluate the accuracy of a specific QL configuration. To do so, running inference predictions on large datasets is necessary. It is impractical to do so on embedded hardware. We adopt an application-level error injection method. As in existing work [12], we develop a custom TensorFlow DNN layer that enables us to simulate errors exactly like those caused by STT-MRAM writes at different QLs.

The custom layer takes as input a tensor and introduces bit-switch errors in the tensor. The probability of introducing these errors can be adjusted as needed. We do so according to the WER of the QL we want to test. After applying these changes to the data, the layer outputs the corrupted tensor. We insert this custom layer right after the layers whose output are persisted in STT-MRAM. Being an experimental artifact, this entire processing happens at compile-time and does not contribute to any energy estimation.

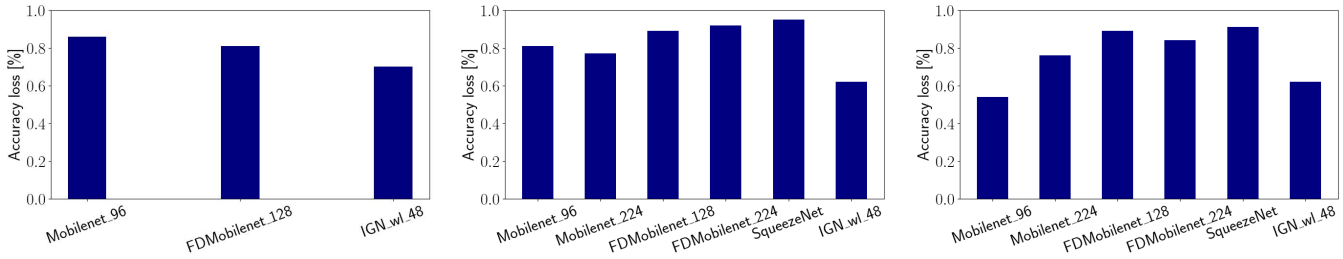
4.2 Results → Energy Consumption

Fig. 6 depicts the energy gain, corresponding to the yellow shaded area, after applying INTERCEPT, based on the target MCU and compared with the baseline and PersistOnce respectively.

We demonstrate that INTERCEPT enables improvements across all configurations we test. When compared with the baseline, the greatest gains are with Mobilenet_224 running on the M33 MCU, where INTERCEPT shaves 64.4% of the energy consumption. The smallest gain in energy is instead with SqueezeNet running on the M7 MCU, where this figure amounts to a 21% gain compared with the baseline. When compared with PersistOnce, we obtain the most gains with FDMobilenet_224 running on the M33 MCU, saving 49.3% of the energy consumption. The smallest gains are again obtained with SqueezeNet running on the M7 MCU, where the energy saved is 12.7% of the energy consumption of PersistOnce.

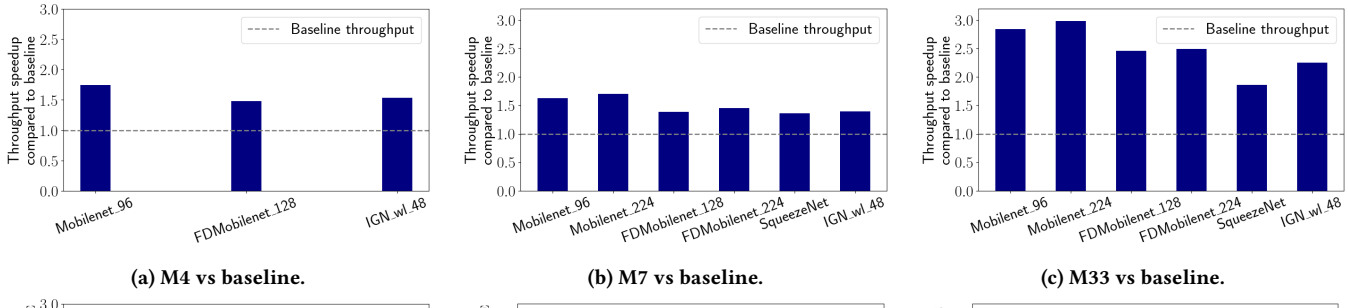
We return to these extremes in Sec. 4.4, where we gain deeper insights that explain how different network architectures and MCUs impact performance with INTERCEPT. In all cases, these gains are obtained by limiting the accuracy loss to just 1%, which is arguably immaterial for most applications [52] and smaller than that of many other existing techniques that trade accuracy for the energy efficiency in DNN inference [26, 64].

The energy gains in Fig. 6 are due to the ability to reduce, by means of Tune, or to eliminate, using Persist, the energy cost $E_{persist}(x, ql)$ for a layer x using ql. *Looping through their repeated execution* amplifies their individual contributions to the final performance, enabling an overall 40% and 29.65% average energy gain considering the baseline and PersistOnce, respectively. What remains of state persistence operations, that is, the yellow solid part

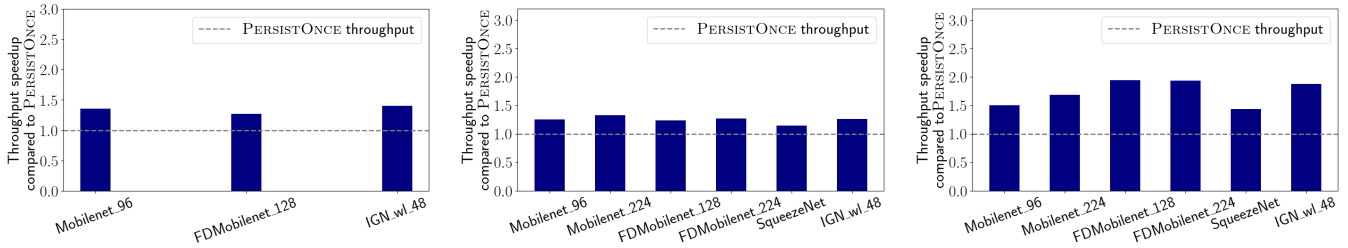


(a) M4. (b) M7. (c) M33.

Figure 8: Accuracy of INTERCEPT-optimized DNNs within the 1% bound on accuracy loss, depending on MCU. *The net accuracy loss is often much lower than the 1% bound.*



(a) M4 vs baseline. (b) M7 vs baseline. (c) M33 vs baseline.



(d) M4 vs PersistOnce. (e) M7 vs PersistOnce. (f) M33 vs PersistOnce.

Figure 9: Throughput speedup with a 1% bound on accuracy loss considering both the baseline and PersistOnce, depending on MCU. *The energy gains enable many-fold improvements in throughput, sometimes more than doubling that of the baseline.*

in Fig. 6, is limited to 7.22% (8.56%) of the overall energy cost compared with the baseline (PersistOnce). This is notable compared with the overhead of state persistence in existing literature, which is reported to reach up to 350% [38].

In contrast, we do not affect $E_{compute}(x)$, which depends on several factors: the layer's dimensions, the active energy of the MCU while computing, and the number of cycles required to compute a layer. The relative proportion of the overall energy consumed to persist all the layers of the DNN, namely $E_{persist}$, compared with the compute counterpart $E_{compute}$ is a function of MCU and DNN.

To investigate these aspects, Fig. 7 provides a different view of the results. Less energy-efficient MCUs might benefit less from INTERCEPT, compared with highly energy-efficient ones, since their $E_{compute}$ tends to dominate the energy figure and we do not alter that. Fig. 7a and Fig. 7c show the overall performance across all DNNs we test, depending on the MCU. The M33 MCU is the one reaping the most benefits, with an average gain of 56.24% and 41.72% compared with the baseline and PersistOnce, respectively. Still, the gains are only slightly less pronounced with the M4 and M7 MCUs but still significant. The chart ultimately provides evidence that INTERCEPT is beneficial independent of the target MCU.

In contrast, Fig. 7b and Fig. 7d show the energy gain for each DNN model we test, averaged across different MCUs. DNNs whose layers generate large tensors that need to be persisted on NVM within fewer computing cycles, like Mobilenet_224, show higher energy gains compared with DNNs where longer computations generate smaller tensors, like SqueezeNet. In the latter case, however, the energy gain is still 32% compared with the baseline and 21.5% compared with PersistOnce. The benefits provided by INTERCEPT also apply across diverse neural network architectures.

Fig. 8 examines the accuracy of the INTERCEPT-optimized DNNs. As expected, the accuracy stays within the $k = 1$ bound. Interestingly, nonetheless, the net accuracy loss is often much lower than that. For example, the INTERCEPT-optimized Mobilenet_224 gains a 64.4% of energy compared with the baseline at the cost of a tiny 0.56% accuracy loss when running on the M33 MCU. This result is due to the inherent error resilience of DNNs, discussed in Sec. 2.3. The fundamental insight we gain is that exerting careful control on the trade-off between energy gain and write errors on

STT-MRAM unlocks great energy benefits, at the expense of almost immaterial losses in accuracy.

4.3 Results → Throughput

Fig. 9 shows the increase in system throughput enabled by INTERCEPT. Compared with the baseline, the largest throughput increase amounts to a 2.98x factor for MOBILENET_224 running on the M33 MCU, whereas the smallest one is a 1.36x factor with SQUEEZENET running on the M7 MCU. Compared with PERSISTONCE, the largest throughput speedup amounts to a 1.94x for FDMOBILENET_224 running on the M33MCU, while the smallest one is again SQUEEZENET running on the M7 MCU, with a speedup of 1.15x.

Comparing Fig. 6 with Fig. 9 provides evidence that the increase in throughput comes from the energy gains shown earlier. The MCU manifesting the highest throughput increase is indeed the M33 MCU for both the baseline and PERSISTONCE, with an average 2.48x speedup considering the baseline and a 1.73x speedup considering PERSISTONCE, while the M4 and M7 MCUs measure respectively a 1.59x and 1.49x speedup compared with the baseline, and a speedup of 1.34x and 1.25x compared with PERSISTONCE.

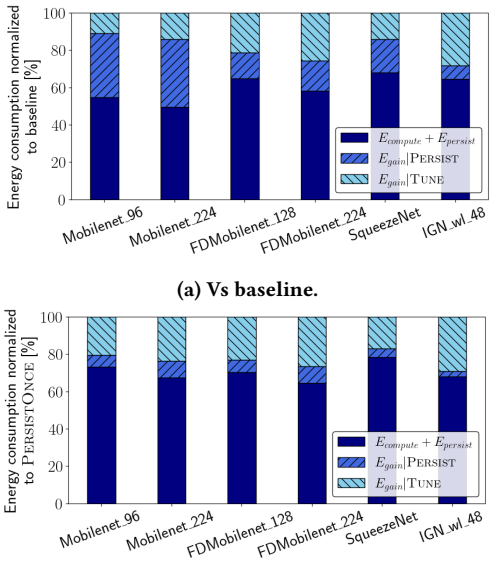
Recharge times dominate active times in an intermittent system [10]. Reducing their number and duration is one way to increase throughput. This is precisely what INTERCEPT enables by lowering the energy demands for running the inference process. Lower energy demands allow the inference process to progress further with the same energy budget, reducing the number of times state persistence is required and capacitors must be recharged.

Lower energy demands, in addition, also allow one to use smaller capacitors, which recharge faster and further reduce recharge times. This is one of the knobs that PERSIST exploits. Smaller capacitors also experience reduced leakage [10]. We do *not* account for capacitor leakage in our throughput measures, while the capacitor selection output by the CONFIGURE stage remains unchanged through the INTERCEPT processing, as explained in Sec. 3, thus missing the concrete opportunity to shorten the recharge times. This means that the throughput improvements we discuss here are, in fact, a lower bound and performance is likely even better.

4.4 Microscope Analysis

Fig. 10 shows the amount of energy we shave because of PERSIST or TUNE. Unlike existing techniques [49], PERSIST works based on the static task structure. Differently, TUNE reduces the energy consumption of individual state persistence operations. Worth observing is that the energy gains they enable are due not just to their individual contribution, but especially to their repeated operation, enabled by the INTERCEPT optimization loop. Simply applying PERSIST, as demonstrated earlier through the comparison with PERSISTONCE, or applying PERSIST followed by TUNE just once would only enable a fraction of the improvements.

Compared with the baseline, as shown in Fig. 10, we observe that the contribution of either step to the overall energy gain depends on the network. For both MOBILENET_96 and MOBILENET_224, PERSIST is responsible for most gains; conversely, TUNE generates the greatest gains for FDMOBILENET_128, FDMOBILENET_224, and IGN_WL_48. In SQUEEZENET, both contribute similarly.



(a) Vs baseline.

(b) Vs PERSISTONCE.

Figure 10: Contribution of PERSIST and TUNE to the overall energy gain. *The specific DNN architecture impacts the relative contribution of PERSIST and TUNE.*

The relative impact of PERSIST or TUNE appears to be tied to the specific DNN architecture, that is, the type of layers and the way they are organized. Indeed, the relative contributions are similar in networks with same architectures but different tensor and input dimensions, such as MOBILENET_96 and MOBILENET_224, where PERSIST bears the greater impact, as opposed to FDMOBILENET_128, FDMOBILENET_224, where TUNE is most effective.

A closer look reveals that networks with frequent alternations of layers where the energy required for computation $E_{compute}$ is relatively small but the energy for state persistence operation $E_{persist}$ is large, with layers where both $E_{compute}$ and $E_{persist}$ are small, benefit from PERSIST. In these cases, PERSIST effectively skips state persistence operations for layers with large output tensors, performing persistence only for those with smaller tensors. This pattern is indeed found in MOBILENET_96 and MOBILENET_224.

In contrast, networks with a more even distribution of energy across compute and state persistence operations offer fewer opportunities for PERSIST to skip state persistence operations. In this case, the operation of TUNE becomes more fundamental to lower the energy consumption of those state persistence operations that are difficult to remove. This is achieved by adjusting the QL settings.

Differently, comparing INTERCEPT with PERSISTONCE from this perspective essentially means looking at the optimization process one step ahead, as PERSIST already executed once. Fig. 10b shows that the reactive contribution of PERSIST is smaller, as the gains it enables *in the first* iteration are not accounted for in the chart anymore. Still, the plot demonstrates that the contribution of PERSIST remains significant also in later iterations.

4.5 Processing Times

The time taken for a full execution of INTERCEPT depends on two factors: *i)* the number of possible state persistence configurations

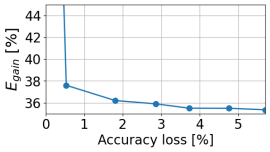


Figure 11: Energy gains with different bounds on accuracy loss, considering MOBILENET_96 running on the M33 MCU. *Note how E_{gain} drops and quickly flattens when k increases.*

generated by PERSIST, that is, the number of times the optimization loop branches out, and *ii)* the time required to complete the genetic search at each step along every branch.

The first aspect only depends on the depth of the DNN. As an example, in the case of a shallow neural network like IGN_WL_48, the number of different state persistence configurations we generate through the whole execution of INTERCEPT is just two. A different DNN, like FDMOBILENET_224, only generates seven different state persistence configurations. The average fan-out seen during the optimization process is generally limited.

With the experimental setup described earlier, the average processing time for the genetic search in a Google Colab environment equipped with an NVIDIA T4 GPU is roughly 1.5 hours. Factors impacting this time are the number of state persistence operations, as determined by PERSIST, and the DNN size. The former determines the combinatorial complexity of the optimization problem, affecting how many generations the genetic algorithm needs to explore. The DNN size impacts the effort to compute the reward for each population element. Evaluating the accuracy of a larger network on a given dataset is more computationally demanding than evaluating a smaller one. For example, a search with six state persistence operations for a small DNN like IGN_WL_48 takes about 0.25 h, while a search with twenty state persistence operations on a larger network like FDMOBILENET_224 takes about 3 h.

5 DISCUSSION

We offer a few considerations on INTERCEPT design and operation.

Selection of k and general optimality. We obtain the results of Sec. 4 by setting a 1% bound on accuracy loss. Such a value is arguably irrelevant in the operation of most applications using DNNs, which tend to have a probabilistic nature in the first place. A natural question, however, is whether it is worth pushing this further and selecting $k > 1$, hoping for even higher energy gains and corresponding throughput improvements.

Fig. 6 already provides an intuition. INTERCEPT exclusively operates on the energy invested in state persistence operations (the yellow parts), whereas it does not alter the cost of computing (the blue part). Already with $k = 1$, INTERCEPT saves most of the energy normally required for state persistence operations, corresponding to the yellow shaded area. What is left, that is, the yellow solid areas, is limited. Setting $k > 1$ could only reduce this part further, yet there is not much room for improvement.

Fig. 11 provides an example quantitative analysis, considering MOBILENET_96 running on the M33 MCU. We run this experiment by increasing k by one percentage at a time, starting from 1. The plot shows that the curve describing the energy gain quickly flattens as k increases. This is precisely because of the intuition above: with $k = 1$, INTERCEPT has limited margins to maneuver further. Similar observations apply to all other settings we explore.

In general, due to the operation of TUNE and especially to the nature of genetic search, we have no guarantee that the solution INTERCEPT outputs is optimal. Computing an optimal configuration may be computationally challenging due to the size of the solution space. The shape of the solution space is also not known. We argue, nonetheless, that the energy gains INTERCEPT provides are also worth a sub-optimal solution, which is obtained within reasonable times, as discussed in Sec. 4.5, yet entirely off-line.

Real-world operation. Environmental factors may impact the performance of embedded devices. However, the setting we consider, and especially the functioning of STT-MRAM in such a setting, is largely oblivious to such factors. The literature specifically indicates a distinct relationship with chip temperature [67], which may hurt reliability [57]. Chip temperature, however, is not due to environmental factors, but to intensive write operations causing the memory to self-heat. This may happen, for example, where STT-MRAM is used as an on-chip last-level cache, yielding frequent write operations that cause self-heating [57, 59]. We use STT-MRAM as a scratch-pad memory at the very end of a computing cycle; writes are performed sparingly and separate in time by at least a cycle of recharge. This may take minutes if not hours.

Voltage fluctuations may occur during memory write operations due to capacitor discharges. The STT-MRAM we consider features a working supply voltage that is sensibly lower than the MCUs one. During the whole computing phase, the capacitor supplies enough voltage to power the STT-MRAM without causing unwanted performance deviations. Moreover, since the amount of data to be persisted is small, that is, in the order of a few KBytes, and the memory bandwidth is some hundreds of MB/s, the time to complete memory operations is reasonably shorter than any capacitor voltage variation. Drops in supply voltage below the working tension of the STT-MRAM are extremely unlikely.

6 CONCLUSION

We presented INTERCEPT, a toolchain for efficient DNN inference on STT-MRAM-equipped intermittent systems. INTERCEPT enables throughput improvements in exchange for bounded accuracy losses, as provided by system designers. Operating entirely at compile time, it does not require any user intervention or changes to the DNN model. INTERCEPT consists of two distinct components cast in an optimization loop. PERSIST minimizes the number of state persistence operations required to ensure the eventual completion of the inference process while favoring the use of smaller capacitors. TUNE leverages STT-MRAM trade-off between energy savings and write errors to reduce the energy consumption of the state persistence operations. By setting the bound on accuracy loss to a mere 1%, we demonstrate across three different MCUs and six diverse DNN models that INTERCEPT achieves an average 40% energy gain, in turn enabling a 1.9x speedup in system throughput.

Acknowledgments. This work is partially supported by the Swedish Science Foundation (SSF) and by the National Recovery and Resilience Plan (NRRP), Mission 4 Component 2 Investment 1.3 - Call for tender No. 1561 of 11.10.2022 of Ministero dell'Università e della Ricerca (MUR); funded by the European Union - NextGenerationEU.

REFERENCES

[1] A. M. Hosseini Monazzah, A. M. Rahmani, A. Miele, and N. Dutt. 2020. CAST: Content-Aware STT-MRAM Cache Write Management for Different Levels of Approximation. *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems* 39, 12 (2020), 4385–4398.

[2] M. Afanasov, N. A. Bhatti, A. Naveed, D. Campagna, G. Caslini, F. M. Centonze, K. Dolui, A. Maioli, E. Barone, M. H. Alizai, J. H. Siddiqui, and L. Mottola. 2020. Battery-Less Zero-Maintenance Embedded Sensing at the Mithræum of Circus Maximus. In *Proc. of ACM Conf. on Embedded Networked Sensor Systems (SenSys)*. 368–381.

[3] M. H. Ahmadilivani, M. Taheri, J. Raik, M. Daneshtalab, and M. Jenihhin. 2024. A Systematic Literature Review on Hardware Reliability Assessment Methods for Deep Neural Networks. *ACM Comput. Surv.* 56, 6 (2024), 1–39.

[4] S. Ahmed, A. Bakar, N. A. Bhatti, M. H. Alizai, J. H. Siddiqui, and L. Mottola. 2019. The betrayal of constant power×time: Finding the missing joules of transiently-powered computers. In *Proc. of the 20th ACM SIGPLAN/SIGBED Intl. Conf. on Languages, Compilers, and Tools for Embedded Systems (LCTES)*. 97–109.

[5] S. Ahmed, B. Islam, K. S. Yildirim, M. Zimmerling, P. Pawełczak, M. H. Alizai, B. Lucia, L. Mottola, J. Sorber, and J. Hester. 2024. The Internet of Batteryless Things. *Commun. ACM* 67, 3 (2024), 64–73.

[6] K. Akhunov and K. S. Yıldırım. 2024. CRAM-Based Acceleration for Intermittent Computing of Parallelizable Tasks. *IEEE Trans. on Emerging Topics in Computing* 12, 1 (2024), 48–59.

[7] G. Armeniakos, G. Zervakis, D. Soudris, and J. Henkel. 2022. Hardware Approximate Techniques for Deep Neural Network Accelerators: A Survey. *ACM Comput. Surv.* 55, 4 (2022), 1–36.

[8] Z. Azad, H. Farbeh, A. M. H. Monazzah, and S. G. Miremadi. 2017. An Efficient Protection Technique for Last Level STT-RAM Caches in Multi-Core Processors. *IEEE Trans. on Parallel and Distributed Systems* 28, 6 (2017), 1564–1577.

[9] F. Bambusi, F. Cerizzi, Y. Lee, and L. Mottola. 2022. The Case for Approximate Intermittent Computing. In *Proc. of Intl. Conf. on Information Processing in Sensor Networks (IPSN)*. 463–476.

[10] N. A. Bhatti, M. H. Alizai, A. A. Syed, and L. Mottola. 2016. Energy Harvesting and Wireless Transfer in Sensor Network Applications: Concepts and Experiences. *ACM Trans. on Sensor Networks* 12, 3 (2016), 1–40.

[11] N. A. Bhatti and L. Mottola. 2017. HarvOS: Efficient Code Instrumentation for Transiently-powered Embedded Sensing. In *Proc. of ACM/IEEE Intl. Conf. on Information Processing in Sensor Networks (IPSN)*. 209–220.

[12] C. Bolchini, L. Cassano, A. Miele, and A. Toschi. 2023. Fast and Accurate Error Simulation for CNNs Against Soft Errors. *IEEE Trans. on Computers* 72, 4 (2023), 984–997.

[13] C. Bolchini and A. Miele. 2013. Reliability-Driven System-Level Synthesis for Mixed-Critical Embedded Systems. *IEEE Trans. on Computers* 62, 12 (2013), 2489–2502.

[14] L. Caronti, K. Akhunov, M. Nardello, K. S. Yıldırım, and D. Brunelli. 2023. Fine-grained hardware acceleration for efficient batteryless intermittent inference on the edge. *ACM Trans. on Embedded Computing Systems* 22, 5 (2023), 1–19.

[15] Z. Chen, G. Li, and K. Pattabiraman. 2021. A low-cost fault corrector for Deep Neural Networks through range restriction. In *Proc. Intl. Conf. Dependable Systems and Networks (DSN)*. 1–13.

[16] A. Colin et al. 2018. Termination Checking and Task Decomposition for Task-based Intermittent Programs. In *Proceedings of the 27th International Conference on Compiler Construction (CC 2018)*.

[17] A. Colin, E. Ruppel, and B. Lucia. 2018. A reconfigurable energy storage architecture for energy-harvesting devices. In *Proc. of Intl. Conf. on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*. 767–781.

[18] Fabrizio De Vita, Rawan MA Nawaiseh, Dario Bruneo, Valeria Tomaselli, Marco Lattuada, and Mirko Falchetto. 2023. μ-ff: On-device forward-forward training algorithm for microcontrollers. In *2023 IEEE International Conference on Smart Computing (SMARTCOMP)*. 49–56.

[19] B. Denby, K. Chintalapudi, R. Chandra, B. Lucia, and S. Noghabi. 2023. Kodan: Addressing the Computational Bottleneck in Space. In *Proc. of Intl. Conf. on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*. 392–403.

[20] T. Devolder, J. Hayakawa, K. Ito, H. Takahashi, S. Ikeda, P. Crozat, N. Zerounian, J.-V. Kim, C. Chappert, and H. Ohno. 2008. Single-Shot Time-Resolved Measurements of Nanosecond-Scale Spin-Transfer Induced Switching: Stochastic Versus Deterministic Aspects. *Physical Review Letters* 100 (2008), 057206. Issue 5.

[21] X. Dong, C. Xu, Y. Xie, and N. P. Jouppi. 2012. Nvsim: A circuit-level performance, energy, and area model for emerging nonvolatile memory. *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems* 31, 7 (2012), 994–1007.

[22] A. F. Gad. 2023. Pygad: An intuitive genetic algorithm python library. *Multimedia Tools and Applications* 83 (2023), 58029–58042.

[23] K. Geissdoerfer and M. Zimmerling. 2022. Learning to communicate effectively between battery-free devices. In *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*. 419–435.

[24] G. Gobieski, B. Lucia, and N. Beckmann. 2019. Intelligence beyond the edge: Inference on intermittent embedded systems. In *Proc. of Intl. Conf. on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*. 199–213.

[25] H. M. G. de A. Rocha, A. C. S. Beck, S. M. D. M. Maia, M. E. Kreutz, and M. M. Pereira. 2020. A Routing based Genetic Algorithm for Task Mapping on MPSoC. In *Proc. Brazilian Symp on Computing Systems Engineering (SBESC)*. 1–8.

[26] Soheil Hashemi, Nicholas Anthony, Hokchhay Tann, R Iris Bahar, and Sherief Reda. 2017. Understanding the impact of precision quantization on the accuracy and energy of neural networks. In *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2017*. 1474–1479.

[27] J. Hester et al. 2014. Ekho: Realistic and Repeatable Experimentation for Tiny Energy-harvesting Sensors. In *Proc. of ACM Conf. on Embedded Network Sensor Systems (SenSys)*.

[28] J. Hester et al. 2017. Flicker: Rapid Prototyping for the Batteryless Internet-of-Things. In *Proc. of ACM Conf. on Embedded Network Sensor Systems (SenSys)*.

[29] J. Hester, L. Sitanayah, and J. Sorber. 2015. Tragedy of the coulombs: Federating energy storage for tiny, intermittently-powered sensors. In *Proc. of ACM Conf. on Embedded Networked Sensor Systems (SenSys)*. 5–16.

[30] L.-H. Hoang, M. A. Hanif, and M. Shafique. 2020. Ft-clipact: Resilience analysis of deep neural networks and improving their fault tolerance using clipped activation. In *Proc. of Design, Automation & Test in Europe Conf. & Exhibition (DATE)*. 1241–1246.

[31] Y. Ibrahim, H. Wang, M. Bai, Z. Liu, J. Wang, Z. Yang, and Z. Chen. 2020. Soft error resilience of deep residual networks for object recognition. *IEEE Access* 8 (2020), 19490–19503.

[32] N. Ikeda, R. Shigeta, J. Shiomi, and Y. Kawahara. 2020. Soil-Monitoring Sensor Powered by Temperature Difference between Air and Shallow Underground Soil. *Proc. of ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies* 4, 1 (2020), 1–22.

[33] B. Islam, Y. Luo, and S. Nirjon. 2023. Amalgamated intermittent computing systems. In *Proceedings of the 8th ACM/IEEE Conference on Internet of Things Design and Implementation*. 184–196.

[34] B. Islam and S. Nirjon. 2020. Scheduling computational and energy harvesting tasks in deadline-aware intermittent systems. In *2020 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*. IEEE, 95–109.

[35] B. Islam and S. Nirjon. 2020. Zygarde: Time-Sensitive On-Device Deep Inference and Adaptation on Intermittently-Powered Systems. *Proc. of ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies* 4, 3 (2020), 1–29.

[36] S. Islam, J. Deng, S. Zhou, C. Pan, C. Ding, and M. Xie. 2022. Enabling fast deep learning on tiny energy-harvesting IoT devices. In *Proc. of Design, Automation & Test in Europe Conf. & Exhibition (DATE)*. 921–926.

[37] S. Islam, S. Zhou, R. Ran, Y.-F. Jin, W. Wen, C. Ding, and M. Xie. 2022. Eve: Environmental adaptive neural network models for low-power energy harvesting system. In *Proc. of IEEE/ACM Intl. Conf. on Computer-Aided Design (ICCAD)*. 1–9.

[38] J. Van Der Woude and M. Hicks. 2016. Intermittent Computation Without Hardware Support or Programmer Intervention. In *Proc. of USENIX Symp. on Operating Systems Design and Implementation (OSDI)*. 17–32.

[39] S. Jeon, Y. Choi, Y. Cho, and H. Cha. 2023. 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)*. 42–55.

[40] C.-K. Kang, H. R. Mendis, C.-H. Lin, M.-S. Chen, and P.-C. Hsiu. 2020. Everything leaves footprints: Hardware accelerated intermittent deep inference. *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems* 39, 11 (2020), 3479–3491.

[41] C.-K. Kang, H. R. Mendis, C.-H. Lin, M.-S. Chen, and P.-C. Hsiu. 2022. More is less: Model augmentation for intermittent deep inference. *ACM Trans. on Embedded Computing Systems* 21, 5 (2022), 1–26.

[42] K. Langendoen. 2006. Apples, oranges, and testbeds. In *2006 IEEE International Conference on Mobile Ad Hoc and Sensor Systems*. IEEE, 387–396.

[43] S. Lee and S. Nirjon. 2019. 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)*. 138–152.

[44] C.-C. Lin, C.-Y. Liu, C.-H. Yen, T.-W. Kuo, and P.-C. Hsiu. 2023. Intermittent-aware neural network pruning. In *Proc. of ACM/IEEE Design Automation Conf. (DAC)*. 1–6.

[45] B. Lucia and B. Ransford. 2015. A Simpler, Safer Programming and Execution Model for Intermittent Systems. In *Proc. of ACM SIGPLAN Conf. on Programming Language Design and Implementation (PLDI)*. 575–585.

[46] M. Lv and E. Xu. 2022. Deep Learning on Energy Harvesting IoT Devices: Survey and Future Challenges. *IEEE Access* 10 (2022), 124999–125014.

[47] Mingsong Lv and Enyu Xu. 2022. Efficient dnn execution on intermittently-powered iot devices with depth-first inference. *IEEE Access* 10 (2022), 101999–102008.

[48] A. Maioli and L. Mottola. 2021. Alfred: Virtual memory for intermittent computing. In *Proceedings of the 19th ACM Conference on Embedded Networked Sensor Systems*. 261–273.

[49] A. Y. Majid et al. 2020. Dynamic Task-Based Intermittent Execution for Energy-Harvesting Devices. *ACM Trans. on Sensor Networks* 16, 1 (2020), 1–24.

[50] H. R. Mendis, C.-K. Kang, and P.-C. Hsiu. 2021. Intermittent-aware neural architecture search. *ACM Trans. on Embedded Computing Systems* 20, 5s (2021),

1–27.
[51] S. Mittal. 2016. A survey of techniques for approximate computing. *ACM Comput. Surv.* 48, 4 (2016), 1–33.
[52] H. Noh, T. You, J. Mun, and B. Han. 2017. Regularizing deep neural networks by noise: Its interpretation and optimization. In *Conf. on Neural Information Processing Systems (NIPS)*. 5115–5124.
[53] A. Ranjan, S. Venkataramani, X. Fong, K. Roy, and A. Raghunathan. 2015. Approximate storage for energy efficient spintronic memories. In *Proc. of ACM/IEEE Design Automation Conf. (DAC)*. 1–6.
[54] S. Resch, S. K. Khatamifard, Z. I. Chowdhury, M. Zabihi, Z. Zhao, H. Cilasun, J.-P. Wang, S. S. Sapatnekar, and U. R. Karpuzcu. 2020. MOUSE: Inference in nonvolatile memory for energy harvesting applications. In *Proc. of Annual IEEE/ACM Intl. Symp. on Microarchitecture (MICRO)*. 400–414.
[55] S. Resch, S. K. Khatamifard, Z. I. Chowdhury, M. Zabihi, Z. Zhao, H. Cilasun, J.-P. Wang, S. S. Sapatnekar, and U. R. Karpuzcu. 2022. Energy-efficient and reliable inference in nonvolatile memory under extreme operating conditions. *ACM Trans. on Embedded Computing Systems* 21, 5 (2022), 1–36.
[56] S. S. Saha, S. S. Sandha, and M. Srivastava. 2022. Machine learning for microcontroller-class hardware: A review. *IEEE Sensors Journal* 22, 22 (2022), 21362–21390.
[57] Arash Salahvarzi, Mohsen Khosroanjam, Amir Mahdi Hosseini Monazzah, Hakem Beitollahi, Umit Y. Ogras, and Mahdi Fazeli. 2023. WiSE: When Learning Assists Resolving STT-MRAM Efficiency Challenges. *IEEE Trans. on Emerging Topics in Computing* 11, 1 (2023), 43–55.
[58] N. Sayed, F. Oboril, A. Shirvanian, R. Bishnoi, and M. B. Tahoori. 2017. Exploiting STT-MRAM for approximate computing. In *Proc. of European Test Symp. (ETS)*. 1–6.
[59] S. Seyedfaraji, J. T. Daryani, M. M. Sabry Aly, and S. Rehman. 2022. EXTENT: Enabling Approximation-Oriented Energy Efficient STT-RAM Write Circuit. *IEEE Access* 10 (2022), 82144–82155.
[60] Joshua R Smith, Alanson P Sample, Pauline S Powledge, Sumit Roy, and Alexander Mamishev. 2006. A wirelessly-powered platform for sensing and computation. In *International Conference on Ubiquitous Computing*. Springer, 495–506.
[61] STMicroelectronics. 2023. STLQ015 Datasheet. https://www.st.com/resource/en/datasheet/stlq015.pdf. Accessed: 2024-09-23.
[62] STMicroelectronics. 2023. STM32AI Model Zoo - Image Classification Models. https://github.com/STMicroelectronics/stm32ai-modelzoo/tree/main/image_classification/models. Accessed: January 22, 2024.
[63] S. Tosun, O. Ozturk, E. Ozkan, and M. Ozen. 2015. Application mapping algorithms for mesh-based network-on-chip architectures. *Journal of Supercomputing* 71, 3 (2015), 995–1017.
[64] Y. Wu, Z. Wang, Z. Jia, Y. Shi, and J. Hu. 2020. Intermittent inference with nonuniformly compressed multi-exit neural network for energy harvesting powered devices. In *Proc. of ACM/IEEE Design Automation Conf. (DAC)*. 1–6.
[65] C.-H. Yen, H. R. Mendis, T.-W. Kuo, and P.-C. Hsiu. 2022. Stateful neural networks for intermittent systems. *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems* 41, 11 (2022), 4229–4240.
[66] K. S. Yıldırım, A. Y. Majid, D. Patoukas, K. Schaper, P. Pawelczak, and J. Hester. 2018. Ink: Reactive kernel for tiny batteryless sensors. In *Proc. of ACM Conf. on Embedded Networked Sensor Systems (SenSys)*. 41–53.
[67] Liuyang Zhang, Yuanqing Cheng, Wang Kang, Lionel Torres, Youguang Zhang, Aida Todri-Sanial, and Weisheng Zhao. 2018. Addressing the Thermal Issues of STT-MRAM From Compact Modeling to Design Techniques. *IEEE Trans. on Nanotechnology* 17, 2 (2018), 345–352.