Resolving Energy Storage for Intermittent Inference

Rei Barjami
Politecnico di Milano
Italy
rei.barjami@polimi.it

Antonio Miele
Politecnico di Milano
Italy
antonio.miele@polimi.it

Luca Mottola
Politecnico di Milano, RISE,
and Uppsala University
Italy and Sweden
luca.mottola@polimi.it

Abstract

We reveal two fundamental insights when deploying inference workloads on intermittent systems that use capacitors as energy buffers. Because of varying structural characteristics and data-independent execution of Deep Neural Network (DNN) models, different capacitor configurations can swing inference performance from the good to the bad. Second, capacitor leakage, which is inevitable in real deployments and yet vastly overlooked in existing literature, makes a whole difference when determining the most efficient capacitor configuration. Both insights hold independently of specific techniques that enable intermittent inference. They form the basis to design LEACS: an off-line technique to determine an efficient energy storage configuration for intermittent inference workloads. LEACS determines the number and size of capacitors based on the nature of the energy source, while accounting for capacitor leakage. We test LEACS together with two state-of-the-art intermittent inference execution techniques and compare its performance with four different baselines across three hardware platforms, seven DNNs models, and three real-world energy sources. Our results indicate that LEACS improves inference throughput by up to 3.4×, with an average 1.59× across the settings we test.

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. 2026. Resolving Energy Storage for Intermittent Inference. In *ACM/IEEE International Conference on Embedded Artificial Intelligence and Sensing Systems (SenSys '26), May 11–14, 2026, Saint Malo, France.* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3774906.3800500

1 Introduction

Ambient energy harvesting allows embedded sensing devices to eliminate their dependency on batteries [9, 39]. This reduces maintenance costs and enables multi-year zero-maintenance deployments [1, 15, 25]. However, energy from the environment is generally erratic, causing frequent and unanticipated energy failures.

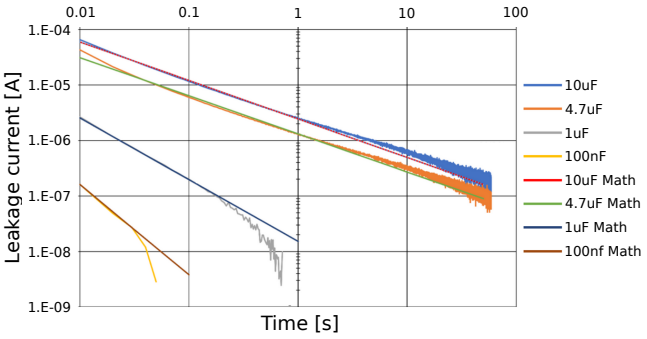


Figure 1: Leakage measurements on various capacitor sizes. *Image from Silicon Labs [48]. Leakage is proportional to capacitor size, the relationship being almost linear. Note how the* 10μF *capacitor suffers more than* 10 *times the leakage of the* 1μF *capacitor. The* Math *curves represent the linear trend of leakage current over time.*

Intermittent inference. Because of erratic energy patterns, executions become *intermittent*: active cycles are interleaved by periods of recharging energy buffers, such as capacitors [3]. 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 persist intermediate application states on Non-Volatile Memory (NVM), which usually imposes substantial energy overhead [29].

Several works exist that support the intermittent execution of Deep Neural Networks (DNNs) [19], as we discuss in Sec. 2. These techniques enable local executions of the inference process, which allows systems to only transmit the response instead of raw data. This feature reduces energy consumption due to communication, especially when using wireless technologies such as Bluetooth and 802.15.4 [3]. Existing works apply a variety of techniques that reduce processing demand and, consequently, energy consumption. These include, for example, compression techniques such as separation, quantization, and pruning [19]; operating run-time decisions such as early-exits [27]; or employing custom programming techniques [19] and taking advantage of specific NVM technology [8].

A house on sand. Despite the current state of the art, one deceptively simple, yet fundamental aspect remains under-investigated: the energy storage configuration. As we experimentally demonstrate in Sec. 3, capacitor selection is not merely a low-level implementation detail. It is a primary factor in determining the energy efficiency of the inference process and thus overall throughput, even when all other system parameters remain fixed.

Capacitors vary in size, charge–discharge dynamics, and leakage rate [23]. Large capacitors may store enough energy to execute large workloads in a single active cycle, hence abating the overhead

This work is licensed under a Creative Commons Attribution 4.0 International License.
SenSys '26, Saint Malo, France
© 2026 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-2309-4/2026/05
https://doi.org/10.1145/3774906.3800500

of state persistence operations. As we show in Fig. 1, however, large capacitors exhibit high leakage currents that dissipate stored charge over time. Moreover, they require longer recharge times before entering a new active cycle. In contrast, small capacitors recharge quickly and suffer reduced leakage but can store only limited energy, often insufficient to complete the execution of even a single inference iteration. This leads to frequent energy failures, increased overhead from state persistence operations, and larger energy waste due to partial executions. Fig. 1 also crucially provides two indications with general validity [24, 33, 44]. First, capacitor leakage occurs regardless of environment conditions: the curves in Fig. 1 are valid in a [-55°C,+125°C] interval. Second, leakage currents are largely linear against time, given the capacitor size.

Striking a trade-off between these many conflicting aspects is challenging. We argue that, as a result, the impact of energy storage configuration on intermittent inference is vastly overlooked. As a paradigmatic example that we further articulate in Sec. 3, with all other system parameters fixed, a sub-optimal choice of capacitor sizes may reduce system throughput by up to *one order of magnitude*. This happens mainly due to the inability to store excess energy or because of capacitor leakage. Most existing literature [4, 14, 19, 23] explores multiple capacitor configurations in system evaluations mainly to increase the diversity and coverage of performance results, without a critical analysis to identify the related trade-offs.

A few works exist that determine capacitor sizes to ensure eventual completion of the workload or to batch together multiple execution units [13, 14]. These techniques are oblivious to the nature of the energy source, and therefore cannot account for situations where excess energy may arise that the chosen capacitor configuration cannot store, or for energy losses due to capacitor leakage. Their output is often overly optimistic and does not necessarily translate to real-world performance [18].

LEACS. We seek to bridge this gap by crucially noting that the energy consumption of inference processes is *predictable and data-independent*. Each neural layer or subgraph executes a fixed sequence of operations whose energy cost can be statically profiled at compile time. This feature enables fine-grained energy budgeting, especially when devices are equipped with multiple capacitors of varying sizes [14], each with distinct leakage patterns.

Processing jobs with limited energy demands, such as executing activation functions or pooling layers, may be assigned to small capacitors that recharge quickly and reduce energy losses due to leakage. Processing jobs with large energy demands, such as large convolutions, may be assigned to larger capacitors only whenever necessary. The potential benefits, however, come at the cost of additional design complexity. Developers must determine the most efficient selection of capacitors to provision from a possibly large catalog of candidates, subject to constraints on cost and board space.

LEACS is a compile-time technique that automatically determines the most efficient energy storage configuration for intermittent inference workloads. It does so by tailoring the capacitor selection to the dynamics of a given ambient energy source and by taking capacitor leakage into account. It takes as input a DNN model, a catalog of available capacitors, and a cap on the number of capacitors that may be provisioned. It outputs a capacitor configuration and a mapping of execution units to capacitor(s) that optimizes throughput for a given ambient source. We compute the output of LEACS by solving a problem we formulate as a Mixed-integer Linear Program (MILP). We combine this with a custom technique to replay existing energy traces that accounts for capacitor leakage and allows the objective function to be computed quickly, and yet without introducing non-linearities in the problem formulation . Sec. 4 describes the design of LEACS.

Performance. To measure the performance impact, we integrate LEACS within two existing execution techniques supporting efficient intermittent inference [8, 19]. They employ different granularities to drive the intermittent inference process and use fixed, often manually determined energy storage configurations in either single- or multi-capacitor setups. These features are instrumental to gauge LEACS's general applicability. After integration with LEACS, these systems employ carefully optimized energy storage configurations output as a result of the optimization process.

We experimentally measure the system throughput as the number of inference processes completed within the unit of time for both the original configurations and after the integration with LEACS. We do so across seven heterogeneous DNN workloads, three different hardware platforms, and three real-world energy sources [17] with distinct energy content and instantaneous power dynamics.

Our results, discussed in Sec. 5, indicate that the throughput improvements enabled by LEACS peak at 3.4× factor, with an 1.59× average. We also observe that models with greater diversity in the energy demands of execution units benefit the most, as LEACS can tailor capacitor selection to this variety. Crucially, LEACS shows to be most effective in energy-scarce environments, where the effect of leakage is most pronounced, providing an 1.85× improvement on average. Finally, our results provide evidence that the performance improvements brought by LEACS apply to both the existing execution techniques we consider and are valid for single- and multi-capacitor setups, demonstrating general applicability.

We end the paper in Sec. 6 with additional considerations on our design and with brief concluding remarks in Sec. 7.

2 Background and Related Work

We first provide key concepts of intermittent computing, with a focus on intermittent inference. Next, we survey related work on energy storage in energy-harvesting embedded sensing systems.

2.1 Intermittent Inference

Intermittent computing devices consist of a Microcontroller Unit (MCU) equipped with some form of NVM and powered by ambient energy sources like solar radiation, motion, or thermal gradients [9]. Every energy source exhibits unique characteristics, for example, in overall energy content, time dynamics, and efficiency of the corresponding harvesting technology [9, 17].

Executing intermittently. Common to most energy sources, however, is erratic behavior [3]. Embedded devices powered by energy harvesting employ energy buffers, usually in the form of one or more capacitors, to tame fluctuations of ambient energy. Devices slowly charge the capacitor(s) until reaching a specific threshold voltage, when the MCU powers on and an active cycle begins that may include sensing, computing, and communication. MCU and peripherals rapidly drain the available energy during an active cycle until the voltage drops below a minimum threshold, causing an

energy failure where the device shuts down. The charging phase restarts until the device reaches the activation threshold. This results in an *intermittent execution* where periods of active operation are interspersed with periods of recharging energy buffers.

Several techniques ensure forward progress across energy failures by using NVM to persist intermediate states [10] and resume computation at the start of an active cycle. Task-based programming systems require programmers to split applications into distinct slices with transactional semantics [38]. For example, Alpaca [41] uses static compiler analysis to enforce uninterrupted task execution, achieving orders of magnitude higher performance than checkpoint-rollback schemes. Intermittent inference systems, illustrated next, often adapt task-based programming to DNN structure.

Enabling intermittent inference. Several works investigate the execution of DNNs in intermittent systems [9, 39]. Works exist that apply a variety of model compression techniques or rely on multi-exist network architectures. As an example, Wu et al. [55] design a network compression algorithm based on model pruning and quantization techniques that works with multi-exit DNNs to select exits based on energy predictions. Given an existing multi-exit model, the notion of approximate intermittent computing [7] allows the processing to step out of the inference process before state persistence operations are necessary.

Model augmentation and pruning are also often employed to run intermittent inference. Kang et al. [31, 32] and Yen et al. [57], 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 [37] embeds an ad-hoc pruning strategy that produces compact models for intermittent systems, whereas RAD [28] employs block circulant matrices and structured pruning to exploit vector operation accelerators.

In other works, special-purpose execution techniques and fine-grained tuning of hardware parameters enable efficient inference. To reduce the energy overhead of state persistence, Lv et al. [40] slice the network horizontally and execute each slice in a depth-first manner, only saving a fraction of the state on NVM. Zygarde [27] models the energy patterns together with the relation between accuracy and processing requirements. Sonic [19] presents a concept of loop continuation that reduces the overhead of frequent state persistence operations during inference.

Intercept [8] employs Spin-Transfer Torque Magnetic Random-Access Memory (STT-MRAM) for persisting intermediate states, tuning write currents for saving energy while keeping accuracy losses under control. Neuro.ZERO [36] 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 during training.

Mapping the inference process to tasks is natural. For example, Intercept and Sonic map the notion of task completion to the execution of a single layer in DNN inference, or a slice thereof. As a result, the energy available in a single active cycle determines how many layers, if any, can execute between energy failures. Say for example the largest capacitor is insufficient to complete the execution of the most energy-demanding layer in a network. In that case, the execution would eventually enter a livelock as the inference process would systematically restart from the same layer: as the energy storage on board is insufficient to progress any further.

2.2 Energy Storage

Despite existing literature mostly fails to recognize the impact of energy storage configurations on intermittent inference, works exist that investigate different aspects related to energy storage in the general area of intermittent computing.

Capacitor models. Intermittently computing devices determine the energy available in terms of the voltage reading at the capacitor's ends. Between the lowest voltage reading V_{min} that ensures device operation and the voltage reading V_{max} that triggers an active cycle, the energy stored in a capacitor of capacitance C is

$$E_{\text{stored}} = \frac{1}{2}C\left(V_{\text{max}}^2 - V_{\text{min}}^2\right). \tag{1}$$

The value of C determines the energy stored and thus the maximum workload achievable in an active cycle. Larger values of C yield more E_{stored}, potentially allowing completion of larger workloads before an energy failure and hence amortizing the overhead of state persistence operations. However, larger values of C also slow down the charging process. For energy sources such as motion or thermal gradients [9], a large capacitor may take minutes, if not hours [1], to charge up to V_{max}, reducing throughput.

To complicate matters, capacitors naturally experience *leakage*. While charging or during idle periods, the capacitor incurs in losses that drain energy without performing useful work. These losses are usually proportional to the capacitance value C and the capacitor voltage reading V. As an example, for electrolytic capacitors [24] and especially aluminum types, the leakage current is

$$I_{leak} = \begin{cases} 0.01CV & \text{if } I_{leak} > 3\mu A \\ 3\mu A & \text{otherwise.} \end{cases} \tag{2}$$

These expressions may differ for different capacitor technologies [24, 44]. Ceramic capacitors usually enjoy extremely low leakage. Film capacitors, on the other hand, experience leakage currents that depend on the construction process. These information are typically found in datasheets or obtained experimentally [44].

Losses due to capacitor leakage effectively reduce the usable energy and thus system throughput: energy leaked during recharge or idle periods is *wasted energy*. This is a core part of our work; we return to this in Sec. 3 with quantitative evidence.

Capacitor configurations. Using modern battery technology in place of regular capacitors, for example, Jackson et al. [30] recognize that energy-rich ambient sources may execute in a non-intermittent manner, sparing the overhead of state persistence on NVM. They study how the design space changes in this case, considering solar radiation. In contrast, our work applies regardless of the nature of the energy source and provides the greatest benefits precisely in the most challenging energy-poor scenarios, such as kinetic sources.

Hester et al. [23] first recognize the trade-offs between large and small capacitors. They note the increased leakage of the former but do not qualitatively account for this aspect in design decisions. Capybara [14] takes a step forward by presenting a hardware/software system that includes several capacitor banks and switchable connections. Applications declare different energy modes for tasks, and the runtime reconfigures the capacitor banks to

match those needs. Similarly, Stash [4] uses a multi-capacitor front end that "performs like a small capacitor when small capacitors excel and like a large capacitor when large capacitors excel".

These designs dynamically handle a *given* energy storage configuration based on application demands, yet provide no indication of what is the most efficient energy storage configuration, as the number and size of individual capacitors, for the application at hand. This information is normally up to developers to determine, which forces deployments of intermittent systems to a painful process of trial-and-error [1, 3, 12]. We seek to fill precisely this gap by taking advantage of the distinctive features of intermittent inference.

The only existing tool that indirectly hints at an effective capacitor configuration is CleanCut [13]. From compile-time worst-case task energy estimates, it checks whether chosen capacitors can ensure eventual workload completion. When livelock may occur, CleanCut suggests splitting tasks into smaller units, but offers no guidance on changing capacitor configurations. The developer must instead decide whether to equip the system with larger capacitors and then re-run CleanCut [13]. CleanCut, however, ignores capacitor leakage and its interaction with the source.

3 Motivation

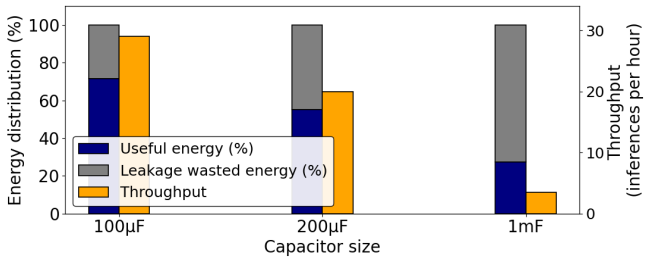
We offer *quantitative experimental* evidence of how capacitor configurations impact inference performance, and of the role of energy sources and capacitor leakage. We run experiments using MOBILENET_96 with SONIC state-of-the-art technique on a Cortex M33 MCU, using a single capacitor of three different sizes: $1mF$ and $100\mu F$ as reported in existing literature [19], plus an intermediate configuration of $200\mu F$. Note that even the smallest capacitor we consider ensures eventual completion of the workload, hence we rule out unfeasible configurations.

To ensure repeatability across energy storage configurations, we use energy traces from Bonito [17]. One trace we consider is gathered by deploying a piezoelectric harvester at the ankles of a person performing a JOGGING routine. The other trace is obtained from a SOLAR cell embedded within the surface of an outdoor stair in front of a lecture hall, with numerous students passing by, which leads to temporary shadowing effects. We consider Panasonic FR-A series electrolytic capacitors and estimate capacitor leakage using the equations provided in the corresponding datasheet [49]. The rest of the experimental setting is the same we use in Sec. 5.2.

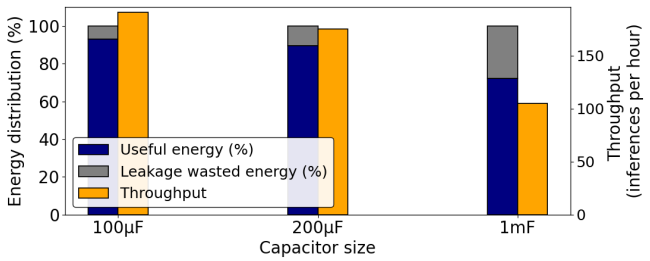
The following discussion is based on a specific setup exclusively for illustration purposes. The observations are largely applicable across different DNN models, hardware platforms, energy traces, and energy storage configurations, as we further discuss in Sec. 5.

Observation 1: *The throughput performance changes with capacitor size and in a way* not *proportional to it.*

Fig. 2a shows SONIC's performance on the JOGGING trace for three capacitor setups. The $100\mu F$ setup is the most efficient, completing the most inference processes over the energy trace. By contrast, the $1mF$ setup completes an order of magnitude fewer, mainly because of longer recharge times and leakage, discussed next. The $200\mu F$ setup stores twice the energy of the $100\mu F$ one, yet incurs only a 35% performance penalty. These results are difficult to anticipate; they indicate that the energy budget available to the system bears no clear relation to the attainable performance.



(a) JOGGING trace.



(b) SOLAR trace.

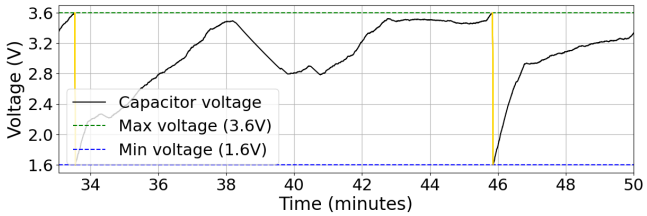
Figure 2: SONIC performance in an example setting. *We consider a Cortex M33 MCU running MOBILENET_96. We note that* i) *the lower the capacitor size, the higher the throughput;* ii) *the energy trace considerably affects the performance,* iii) *the larger the capacitor, the higher the leakage, and in turn the lower the throughput.*

Observation 2: *The same energy storage configuration performs differently with different energy sources.*

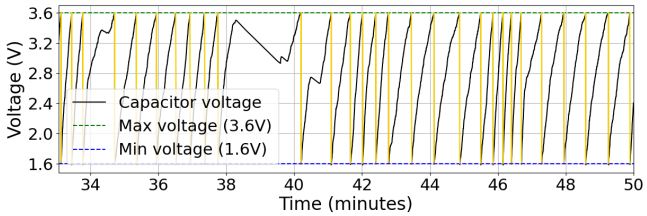
Fig. 2b mirrors Fig. 2a for the SOLAR trace, which is richer in energy and exhibits higher instantaneous power delivery than JOGGING. The *same* capacitors we use with the JOGGING trace now perform very differently individually and relative to each other; most importantly, some earlier observations no longer apply. The absolute number of inference processes completed per hour is one order of magnitude higher than with the JOGGING trace. The performance of the $200\mu F$ is now much closer to the best performing $100\mu F$ configuration. The $1mF$ capacitor now provides about half the performance of the latter, compared to the one order of magnitude difference in Fig. 2a. With the SOLAR trace, recharge times are generally shorter even for the largest capacitor, and the effect of leakage currents is mitigated by the higher instantaneous power delivery.

Observation 3: *Capacitor size regulates the interplay between recharge times and leakage, which determines performance together with the dynamics of the energy source.*

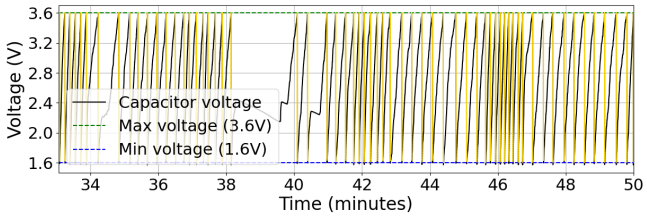
The blue and grey bars in Fig. 2a and Fig. 2b show the percentage of energy stored in the capacitor used to achieve useful processing or lost through capacitor leakage, respectively. The larger the capacitor, the greater is the latter. The performance shown in the yellow bars of both charts are an effect of losing an increasing amount of energy through leakage and longer recharge times. Accordingly to the earlier observations, the energy wasted through leakage is reduced in the SOLAR trace compared to the JOGGING one, as the former somehow compensates for these losses. Existing literature does not report performance numbers to quantify these effects, neither takes these aspects explicitly into account in the design process.



(a) $1mF$ **capacitor.**



(b) $200\mu F$ **capacitor.**



(c) $100\mu F$ **capacitor.**

Figure 3: Sonic execution with the Jogging trace over time. *Light grey segments represent the charge period while yellow ones represent the active cycles. Larger capacitors take more time to recharge and therefore reduce the number of active cycles.*

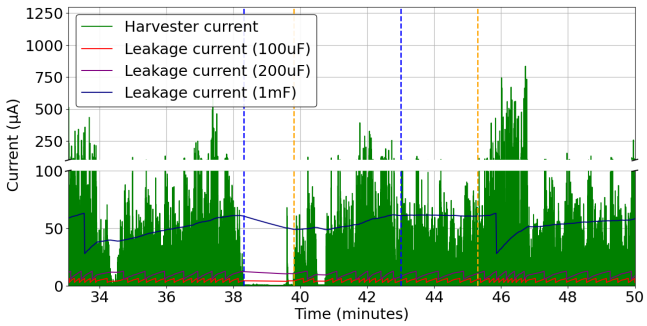


Figure 4: Harvester and leakage currents with the Jogging trace over time. *The green line indicates the harvester current over time, while the other three lines represent leakage currents depending on capacitor size; the larger the capacitor, the larger the leakage.*

Observation 4: *Depending on the energy source, leakage current may hamper active cycles at unpredictable times.*

Fig. 3 details the execution of Sonic in a specific time frame with the Jogging trace in either of the three capacitor configurations, while Fig. 4 reports with the harvester and leakage currents in the same period. Fig. 3a shows that at around 38 minutes into the execution, the $1mF$ capacitor is nearly at the activation threshold. At that time, the current coming from the harvester is almost down to zero, as shown in Fig. 4. Leakage current, shown in red in the same plot, now effectively discharges the capacitor, as shown by the decreasing trend in Fig. 3a between 38 and 40 minutes, pushing the system away from a new active cycle. The execution unfolds in yet a different way between 43 and 46 minutes for the same $1mF$ capacitor. Fig. 4 shows that within this time frame, the average current from the harvester nearly equals the leakage current. The charge in the $1mF$ capacitor stays almost constant and close to the activation threshold, without ever surpassing it. This happens only at minute 46, where an active cycle finally begins.

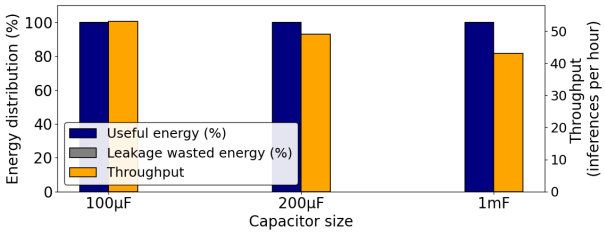


Figure 5: Sonic performance with the Jogging trace without accounting for capacitor leakage. *Compared to results in Fig. 2a, a simplified capacitor model yields efficient performance, which is however not expected in a real deployment.*

These issues are much less severe with the other capacitor configurations in Fig. 3b and Fig. 3c, because of shorter recharge times and lower leakage currents, shown by the blue and purple lines in Fig. 4. Between 38 and 40 minutes, both configurations lack of input current from the harvester, but manage to start an active cycle at around minute 40 anyway. Between 43 and 46 minutes into the experiment, both configurations activate the device multiple times, with the $100\mu F$ configuration enjoying shorter recharge times and hence more frequent activations.

We see no similar behaviors with the Solar trace, whose trends we do not show for brevity, because of the higher power.

Observation 5: *Not considering leakage currents distorts performance evaluations.*

Fig. 5 shows the performance of Sonic with the Jogging trace without considering the energy loss due to capacitor leakage. The difference compared to Fig. 2a is striking. The $100\mu F$ configuration provides slightly higher performance, yet the $1mF$ configuration shows a 600+% increase in performance, eventually producing a mere 20% penalty compared to the best-performing setting.

Two factors contribute to this spurious result. Most of the energy that would go wasted in a real deployment because of capacitor leakage is now fictitiously employed to push forward the inference process. In reality, this would simply not happen. Further, recharge times are vastly underestimated because situations akin to Fig. 3a are now masked by the idealized, zero-leakage model. Not accounting for these factors may thus result in misleading performance measurements and steer design processes in incoherent directions.

4 LEACS

Based on the insights in Sec. 3, we design LEACS: an off-line technique to determine the most efficient multi-capacitor configuration

to maximize system throughput for an intermittent inference workload. LEACS selects the number and size of capacitors to place on board, as well as the mapping between capacitors and the execution units within the inference process. These decisions account for the nature of the energy source, capacitor characteristics, in particular leakage current and recharge/discharge time, and energy constraints that determine eventual completion of the workload.

4.1 Design Features

Our key objective is to maximize throughput against a truthful energy setting, which thus must account for capacitor leakage and recharge times. We build on two key observations:

1) the energy demands of embedded inference are predictable and data-independent; given a DNN model, it is possible to estimate its energy consumption independent of the inputs;
2) systems supporting intermittent inference feature a well-defined granularity of execution; therefore we can partition the single inference in software execution units, for example, corresponding to single layers [8], chains of layers [8], or loops [19].

These observations are instrumental in affording the following fundamental design features.

C1: Account for capacitor leakage. In LEACS, we shift the design emphasis *from raw capacitance to a balance between usable energy and unavoidable loss*, which ultimately determines the performance. We do so by incorporating accurate leakage models derived from datasheets or direct empirical measurements into both the selection of capacitors and the mapping to execution units, ensuring to employ capacitors whose leakage profiles are acceptable under realistic operating conditions. We thus avoid configurations where harvested energy is squandered before executions even begin, as experimentally shown in Fig. 3a, thereby guaranteeing that more of the harvested energy translates into useful inference work.

C2: Mapping capacitors and execution units. LEACS matches *each execution unit in intermittent inference to the smallest capacitor that can efficiently ensure its completion.* Oversized capacitors incur higher leakage currents and require longer recharge times, reducing throughput. Conversely, undersized capacitors may cause execution units to fail to complete, leading to energy waste. By precisely sizing the mapping between capacitors and execution units, we tame the effects of leakage and reduce recharge times. This yields higher effective duty cycles for each capacitor: smaller capacitors serve execution units with low energy demands, while larger capacitors are reserved for energy-hungry execution units.

C3: Compile-time operation. Run-time energy management in intermittent systems introduces overhead. Any controller that makes decisions on the fly based on residual energy, capacitor voltage, or energy forecasts burns computing cycles. We spare the added complexity and run-time overhead *by determining energy storage configurations at compile time.* This is possible partly because of the key observations above, partly because we compensate for the lack of run-time information by capturing the nature of the energy source through existing traces, which are increasingly available [1, 2, 9, 17, 21, 39] as we articulate in Sec. 6. As a result, we shift the computational burden off the deployed device and onto the compile-time toolchain, where time and resources abound.

4.2 Determining Energy Storage

To determine an efficient energy storage configuration, we uniquely combine the optimal solution to a custom optimization problem, trace-driven throughput estimations, and offline profiling.

Inputs and outputs. As shown in Fig. 6, LEACS takes four inputs:

1) the set of executing units $\mathcal{U} = \{u_1, \ldots, u_N\}$ required to be executed in a sequence to complete an inference, like a layer or a portion of it or a chain of multiple subsequent layers, each annotated with its energy consumption E_i, $1 \le i \le N$.
2) an energy trace $H(t)$, represented as a discrete time series specifying how much power the device harvests at each time step, capturing the nature of the energy source.
3) a capacitor catalog $C = \{C_1, \ldots, C_K\}$, represented as a discrete catalog of capacitors to choose from, each with a known leakage curve and voltage range $[\mathtt{V_{off}}, \mathtt{V_{on}}]_j$, $1 \le j \le K$.
4) an upper bound M on the number of capacitors a device supports, as determined by area, component cost, and the complexity of switching logic.

Given these inputs, LEACS produces as output:

1) a subset $C' \subseteq C$ of at most M capacitors, representing the capacitor array to place on board;
2) a mapping from each execution unit $u_i \in \mathcal{U}$ to a capacitor $C_j \in C'$ that can power the execution of u_i under $H(t)$.

Problem formulation. To determine the outputs, we formulate capacitor selection and the mapping to execution units as a MILP. This formulation captures the discrete nature of the outputs while allowing us to carefully express the necessary constraints.

We define a binary variable x_j for each capacitor C_j in the catalog. Setting $x_j = 1$ indicates that capacitor C_j is included in the design, and $x_j = 0$ means it is omitted. For every execution unit–capacitor pair u_i, C_j, we define a binary variable y_{ij}: if $y_{ij} = 1$, execution unit u_i is assigned capacitor C_j; $y_{ij} = 0$ otherwise. Together, these variables encode every possible energy storage configuration and the mapping of chosen capacitors to execution units.

The objective of LEACS is to maximize throughput, that is, the number of completed inference processes, given a specific energy source. The latter is captured in the MILP formulation by the energy trace $H(t)$. Thus, the objective function is

$$\max_{\{x_j\},\{y_{ij}\}} \mathit{Throughput}(X, Y) \text{ over } H(t). \tag{3}$$

Since throughput depends on non-linear and time-varying phenomena, like capacitor leakage and discharge, as well as energy harvesting, we cannot express it as a linear function. We describe next how we use an analytical approach to accurately estimate $\mathit{Throughput}(X, Y)$, given X and Y.

To aid formulating the necessary problem constraints, we introduce a binary variable $f_{i.j} \in \{0, 1\}$ for each pair $\langle u_i, C_j \rangle$, which models whether an execution unit u_i can possibly be completed with the energy stored in capacitor C_j. Given the inputs described earlier and especially the energy consumption of each execution unit E_i, $1 \le i \le N$, we can precompute the values of all $f_{i,j}$, even before attempting to solve the problem. By relying on these variables, we spare the need to embed discharge equations directly into the MILP, keeping the formulation linear.

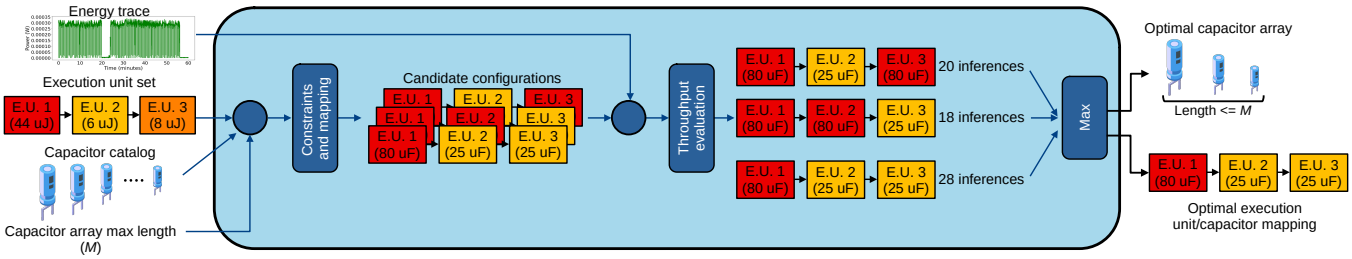


Figure 6: Overview of LEACS. *We take as input an energy trace representative of a given source, the catalog of available capacitors, the energy profile of the execution units in the DNN, and the maximum number M of capacitors on the board. We output the number and sizes of the capacitors, and a mapping of the execution units and capacitors to use during the inference execution.*

We impose four key constraints. First, each execution unit must be assigned to exactly one capacitor:

$$\sum_{j=1}^{K} y_{ij} = 1, \quad \forall\, i \in \{1, \ldots, N\}. \tag{4}$$

Configurations where an execution unit is mapped to multiple capacitors would require deciding at run-time, based on some criteria, which capacitor to employ. The added complexity and overhead, compared with unclear advantages [14], would not pay off.

Second, we must impose that an execution unit is mapped to a capacitor that is actually selected to be part of the solution, which we express as

$$y_{ij} \leq x_j, \quad \forall\, i \in \{1, \ldots, N\},\ j \in \{1, \ldots, K\}. \tag{5}$$

Indeed, if $x_j = 0$, meaning capacitor C_j is not selected, then y_{ij} must be zero for all execution units. These constraints avoid producing meaningless configurations where an execution unit is mapped to a capacitor that is not found onboard the device.

Third, we ensure that the capacitor selected to support the execution unit is sufficient to complete its execution, which entails

$$y_{ij} \leq f_{ij}, \quad \forall\, i \in \{1, \ldots, N\},\ j \in \{1, \ldots, K\}. \tag{6}$$

Without this, we may obtain solutions that are unfeasible in reality and may cause the system to enter a livelock, in that an execution unit may never complete and impede any forward progress [13]. Finally, we impose an upper bound on the number of capacitors as

$$\sum_{j=1}^{K} x_j \leq M, \tag{7}$$

where M is the maximum number of capacitors a device can support, as provided in input and reflecting the physical and cost constraints of the target platform.

Throughput evaluation. When executions are predictable and data-independent, the energy consumption of each execution unit E_i, $1 \leq i \leq N$, depends only on the target MCU and can be accurately quantified using existing tools [2, 13] or by profiling sample executions on real hardware, as we do in Sec. 5 and as found in existing literature [2, 8, 10, 13, 22, 42, 47]. Moreover, leakage currents are reported by capacitor vendors on datasheets, as in Fig. 1, both empirically and analytically. Since energy consumption is additive, components such as execution-unit energy and leakage losses can be summed to estimate total energy consumption.

To evaluate the objective function in Eq. 3, we integrate a specialized replay of the inference process for an energy trace $H(t)$ into the optimization loop. For a candidate assignment (X, Y), we unroll the trace while tracking each selected capacitor's voltage, applying leakage at each time step, and subtracting the energy consumed by the corresponding execution unit. This process returns a scalar throughput value $N_{\text{comp}}(X, Y)$, the total number of inferences performed during the trace. During optimization, the MILP solver uses $N_{\text{comp}}(X, Y)$ to compare candidate solutions. To break ties between configurations with identical throughput, we prefer those using fewer capacitors and, among them, those with lower total leakage.

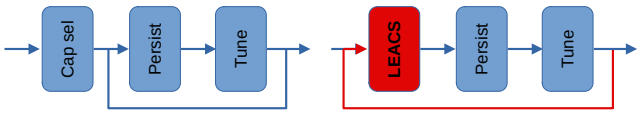
By offloading nonlinear effects such as energy estimation, discharge curves, leakage, and time-varying harvesting to offline profiling, we keep the MILP formulation linear in its decision variables, avoiding nonlinearities. Thus, the optimizer solves a tractable problem with binary variables x_j and y_{ij}, ensuring efficient processing while accurately capturing key energy dynamics.

5 Evaluation

We build a software prototype that emulates DNN execution and capacitor charge/discharge patterns. The emulation is based on energy measurements from concrete hardware and real-world harvested-energy traces, used to extract performance measures across many settings and analyze improvements over the state of the art. We obtain 252 billion data points, leading to the following conclusions:

1) overall, LEACS provides throughput improvements up to a 3.4× factor, with an 1.59× average;
2) models with greater diversity in the energy demands of execution units benefit the most, as LEACS can tailor capacitor selection to this variety;
3) LEACS is most effective in energy-scarce environments, where the effect of leakage is most pronounced, providing an 1.85× improvement on average;
4) the performance improvements brought by LEACS apply to both the adopted execution techniques, namely Intercept and Sonic, and are valid for single- and multi-capacitor setups, providing evidence of general applicability;
5) the benefits enabled by more capacitors eventually flatten out at an 1.69× improvement factor, as the diversity of energy demands of the models we test is fulfilled with $M = 5$.

The rest of the section unfolds as follows. Sec. 5.1 describes the prototype implementation. Sec. 5.2 illustrates the experimental setting, whereas Sec. 5.3 discusses the results we obtain.



(a) Original Intercept. (b) Intercept with LEACS.

Figure 7: Integration of LEACS in Intercept toolchain. *We replace the capacitor selection in the original Intercept by placing LEACS within the optimization loop.*

5.1 Software Prototype

System support for intermittent inference represents DNNs as subsequent execution units with transactional semantics. This structure is given as input to LEACS, as described in Sec. 4. To gauge LEACS's impact on system performance, we integrate it with two intermittent inference techniques: Sonic [19] and Intercept [8]. Each implements a specific way to structure inference models and uses a different set of knobs to control their execution.

Intercept integration. Intercept consists of an initial *Capacitor Selection* step followed by an optimization loop consisting of two further steps, namely *Persist* and *Tune*, as shown on the left side of Fig. 7. Execution units in Intercept correspond to layers in a model. State persistence operations dump the output tensors on NVM. Intercept determines first what capacitor size is sufficient to meet the demand of the most energy-expensive execution unit (layer). Based on that, it selects two additional fixed-size capacitors as a predetermined fraction of the largest capacitor, for example, as one-half and one-quarter of the former. This yields a fixed triplet including a large, a medium, and a small capacitor.

The right side of Fig. 7 shows how we integrate LEACS into Intercept. We replace the capacitor selection with LEACS, embedding LEACS within the optimization loop. This design is based on a simple rationale: *Tune* may alter the energy requirements of the execution units (layers) by adjusting the NVM settings [8]. Placing LEACS within the loop allows it to capture these changing energy figures and possibly update capacitor configurations accordingly.

Sonic integration. Sonic partitions the inference model at fine granularity, typically down to single-channel convolutions in DNNs. We integrate LEACS downstream of Sonic. Unlike Intercept, Sonic's execution units and profiling are independent of capacitor configurations. As a result, embedding LEACS into Sonic's loop offers no benefit. Thus, we perform capacitor selection only at the beginning of the design flow of Sonic after the profiling step.

This integration presents two caveats. First, Sonic is not designed to support multi-capacitor configurations. To maintain a fair comparison, we force LEACS to operate with a single capacitor by setting $M = 1$. This ensures compatibility with Sonic's original assumptions and avoids introducing asymmetries in evaluation. Further, Sonic conditionally skips state persistence operations and supports partial execution when residual energy is sufficient. To emulate this behavior, we extend our throughput evaluation technique to track partial executions and conditional skips.

MILP formulation and throughput evaluation. To derive the inputs for LEACS, we measure on *real hardware* the number of clock cycles required to execute each computational unit of the inference process. These values are combined with the average energy-per-cycle information from the MCU datasheets to estimate the energy consumption of each execution unit. Capacitor parameters and leakage values are taken from datasheets, while energy traces are obtained from real-world measurements, as detailed next. These traces are replayed synthetically while accounting for recharge dynamics, leakage, and energy costs of each execution unit. The resulting performance data are used to compute the throughput values employed during both optimization and evaluation.

MCU	Clock Speed (MHz)	Power Efficiency (μW/MHz)
Cortex M33	160	12.0
Cortex M4	80	32.8
Cortex M7	480	58.5

Table 1: MCUs used for evaluation [5].

The replay of the inference process unfolds by first evaluating a candidate mapping of capacitors and executing units. With this, we compute the corresponding throughput, that is, the number of completed inferences, over a given energy trace. To do so, we integrate a voltage-based energy model accounting for input power, capacitor leakage, and MCU load. Capacitor characteristics and voltage thresholds are input parameters, allowing straightforward adaptation to different hardware targets. In a multi-capacitor setup, we consider switching energy and delays to be negligible relative to the duration of capacitor recharges and active cycles. If not, they can be straightforwardly incorporated into the process that follows.

During replay of the inference process, we track capacitor voltage over time using the harvested energy trace $H(t)$ as input. The process starts with capacitor voltage at 0 V and the MCU powered off. At each time step, we compute currents from energy harvesting, leakage, and, when applicable, MCU operation. The latter accounts for the MCU's power state in determining net current and includes the energy for NVM load/store operations. Combining these terms with relevant boundary parameters, such as the equivalent resistance of the MCU and supporting circuitry, we update capacitor voltage every time unit, normally set to 1 ms. When this value reaches $V_{\max}$, an active cycle begins and processing of an execution unit starts. Since each execution unit U_i has a pre-profiled energy cost E_i, execution continues until E_i is consumed. The MILP constraints discussed earlier ensure that the capacitor stores enough energy to complete the unit.

If residual energy remains post-execution, it is retained. This process continues to emulate the charge–execute cycle in the same way for all the subsequent execution units, each time considering the assigned capacitor, for the overall duration of the energy trace $H(t)$. While progressing, we count the number of executed inferences $N_{\text{comp}}(X, Y)$, which is eventually returned to the MILP engine as the value of the objective function for a specific (X, Y).

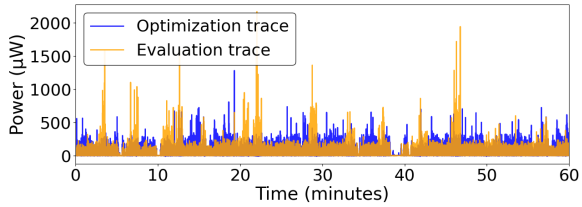
We use Python to implement both MILP solving, using the `pulp` library, and the replay of the inference process described in Sec. 4.

5.2 Setting

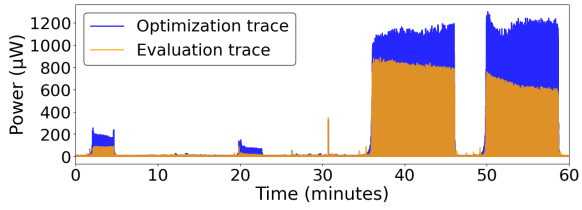
Setup. Similar to existing literature [8, 19, 55], we consider three Cortex-M MCUs with different micro-architectures, instruction sets, and clock speed, to capture how results apply across different platforms. Their key features are illustrated in Tab. 1. As NVM, in Intercept we adopt STT-MRAM as in the original evaluation [8]. For Sonic, we employ Flash memory according to the specifications of the STMicroelectronics datasheets [50–52].

DNN model	MACCs	Avg cost (std) INTERCEPT	Avg cost (std) SONIC
Mobilenet_96	7.5M	35 µJ (±28)	1.8 µJ (±3.8)
Mobilenet_224	41M	137 µJ (±357)	7.33 µJ (±6.5)
FDMobilenet_128	3.9M	13 µJ (±15)	1.51 µJ (±3.7)
FDMobilenet_224	12M	54 µJ (±81)	7.1 µJ (±15)
IGN_24	25K	1.7 µJ (±1.6)	0.6 µJ (±0.8)
IGN_48	68K	4.5 µJ (±3.2)	1.3 µJ (±1.6)
Squeezenet	175M	280 µJ (±246)	11 µJ (±22)

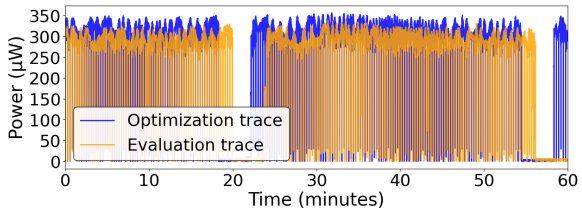
Table 2: Energy cost comparison of INTERCEPT and SONIC execution units for the DNNs models we test.



(a) Jogging.



(b) Washing machine.



(c) Office.

Figure 8: Energy sources. *We use three energy sources with distinct patterns and power dynamics. JOGGING provides the least energy, OFFICE provides the most energy, and WASHING lies in between. Blue traces are used for off-line optimization, while orange ones are used for evaluation. The evaluation traces consistently have lower energy content than the optimization traces: −15.11% for JOGGING, −21.56% for WASHING, and −6.74% for OFFICE.*

We select workloads from the STM32 Model Zoo, covering diverse DNN models. Their main characteristics are summarized in Tab. 2, including architecture, parameter count, activation size, and energy profile. This set is representative of typical embedded-inference workloads and ensures generality of the results.

As input to LEACS, we choose three diverse real-world energy sources from Bonito [17], each with different energy content and power dynamics. For each of these sources, Bonito provides multiple traces, so-called "nodes" in the Bonito open dataset. In our experiments, LEACS performs off-line optimization using one specific trace from a given source, as illustrated in Fig. 6, and we then evaluate its performance on a *different* trace from the same source.

Panasonic FM series specs	
Rated voltage	6.3 V
Capacitor catalog range	$[4.7\,\mu F - 400\,\mu F]$
Leakage current	$0.01CV$

Table 3: Specifications of the capacitor catalog.

This setup ensures that the evaluation conditions differ from those used during optimization, thereby highlighting LEACS's ability to generalize to previously unseen energy dynamics.

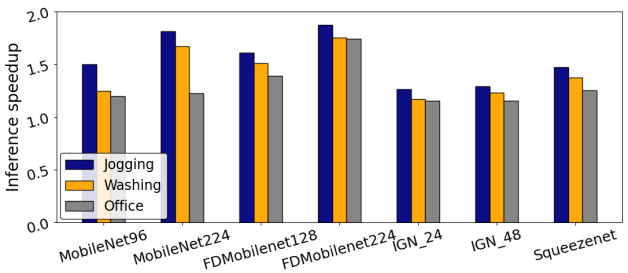
Fig. 8 shows an excerpt: the traces used for off-line optimization are shown in blue, whereas those used for performance evaluation are shown in orange. Note how the traces used for evaluation consistently provide lower energy than those used for optimization: −15.11% for JOGGING, −21.56% for WASHING, and −6.74% for OFFICE. The JOGGING traces include the one we use in Sec. 3. The WASHING traces track the output of a piezoelectric harvester mounted on an industrial washing machine, providing bursts of energy when it operates. The OFFICE traces report the power from solar panels embedded within a floor.

The sources we consider cover a sizable slice of the spectrum of possible energy dynamics [9]. The JOGGING source is scarce in overall energy content and provides quasi-periodic bursts of power, as shown in Fig. 8a. The WASHING source provides more total energy than JOGGING, but with a distinct pattern of long high-power bursts corresponding to the times the machine is operating, as in Fig. 8b. The OFFICE source provides the most energy overall, but with occasional periods of lack of power due to occlusions of the panel, which may stretch for minutes, as in Fig. 8c.

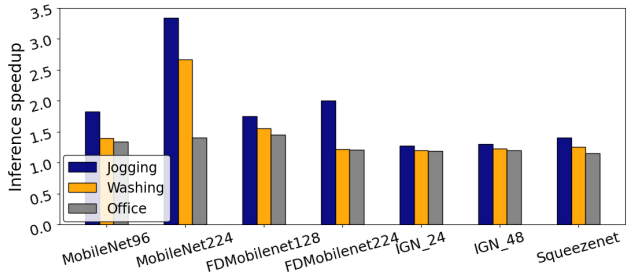
Metrics and baselines. The primary metric we measure is the overall *system throughput*, defined as the number of inference processes completed within the unit of time. This represents how efficiently the system employs available energy and directly links to the perceived quality of service provided to end users. Because of the off-line operation of LEACS, we bear no run-time overhead. We consider four baselines representative of the state of the art.

1) CHECKPOINT: the system uses multiple capacitors [14] and execution units map to single layers in the model. This configuration represents the established system design in the absence of specific support for intermittent inference.
2) INTERCEPTDEFAULT [8]: the system uses multiple capacitors and an off-line toolchain that groups layers in a single execution unit to reduce the number of state persistence operations. The largest capacitor is selected to ensure completion of the most energy-demanding layer, other capacitors are selected as a fixed fraction of the largest one.
3) SONICDEFAULT [19]: the system uses a single capacitor and a programming technique to split model execution into units smaller than a layer. Of those used by Gobieski et al. [19], only the $100\mu F$ capacitor can be used here, as it is the only configuration that ensures completion of all DNN in Tab. 2.
4) SONICMANUAL: this is the same execution technique as SONIC. For each DNN in Tab. 2, we select the smallest capacitor from our catalog that ensures completion of the inference, adding diversity to the configuration we test.

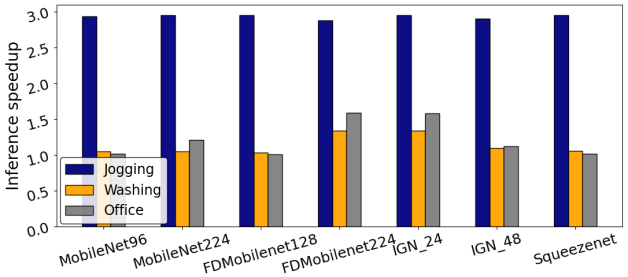
We measure run-time throughput by re-using the technique we describe in Sec. 4 to compute the objective function in LEACS, which applies to the baselines as well.



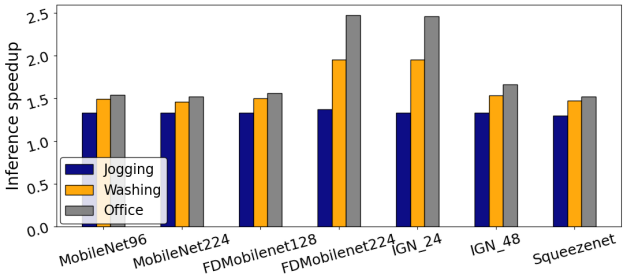
(a) Throughput speedup compared to CHECKPOINT.



(b) Throughput speedup compared to INTERCEPTDEFAULT.



(c) Throughput speedup compared to SONICDEFAULT.



(d) Throughput speedup compared to SONICMANUAL.

Figure 9: LEACS throughput speedup over the baselines, depending on the model and energy source. *Values are averaged over MCUs. Models with greater diversity in the energy costs of the single execution units enjoy the most benefits, as LEACS matches these different energy demands with the proper capacitors. Integrating LEACS with INTERCEPT boosts the gains as the capacitor selection is re-evaluated at each iteration of the INTERCEPT optimization loop; whereas LEACS redefines the SONICDEFAULT configuration because of the significant leakage.*

We summarize the capacitor catalog we consider in Tab. 3. We select components from the Panasonic FR-A series of radial aluminum electrolytic capacitors, spanning values from $4.7\mu F$ to $400\mu F$ in $1\mu F$ increments. We choose the FR-A series because it offers a practical capacitance range for the target MCUs and workloads, as well as datasheet-documented leakage values, ensuring a reproducible evaluation basis. Its leakage characteristics are also representative of typical aluminum electrolytic capacitors of similar sizes [43, 45] and comparable in magnitude to tantalum [35, 53], film [16, 46], and ceramic types [34]. FR-A therefore sits between low- and high-leakage extremes, making it a fair and representative choice for our study. Each capacitor is rated for 6.3 V, which is well above the maximum operating voltage of the target MCUs, ensuring safe operation. We model leakage as $I_{leak} = 0.01CV$, following the specification in the component datasheet [49], where C is the capacitance and V is the voltage reading across the capacitor.

5.3 Results

We observe no significant difference in the trends depending on the kind of MCU, mostly because of the interplay between power efficiency and clock speed. We discuss next the dimensions that expose interesting trends in the data we collect.

Models. Fig. 9 shows the throughput speedup obtained with LEACS, normalized to the throughput obtained by each baseline, as a function of model and energy source. The improvements peak at a 3.4× factor, with an average of 1.59×.

Comparing Fig. 9 with Tab. 2 allows one to note that models with greater diversity in the energy cost of single execution units are those that benefit more from LEACS. For instance, `FDMobilenet_224` is a model with a few high-cost convolutional layers interleaved with many low-cost operations. With `FDMobilenet_224`, LEACS achieves up to a 2.05× throughput improvement. In contrast, `IGN_48` is a lighter network with more uniform execution costs. We observe a more modest 1.20× speedup in this case.

We argue that these improvements come from the way LEACS configures the capacitor array. When layers exhibit diverse energy requirements, we assign the smallest capacitor to each execution unit that ensures completion of the workload, while compensating for capacitor leakage depending on the dynamics of the specific energy source and within the upper bound on the number of capacitors. This reduces the amount of energy wasted because of a sub-optimal energy allocation to each execution unit. If a network consists of layers with roughly equal energy cost, the design space reduces and thus LEACS enjoys fewer degrees of freedom. Most capacitor assignments yield similar performance in this case.

Baselines. Fig. 9 shows that, depending on the baseline we compare with, LEACS achieves an average 1.54× speedup over CHECKPOINT, a 1.59× speedup over INTERCEPTDEFAULT, a 1.8× speedup over SONICMANUAL, and a 1.87× speedup over SONICDEFAULT.

These gains stem from different trade-offs between leakage losses and the energy cost of persisting intermediate states. In CHECKPOINT, this operation occurs after every execution unit without other energy consideration. Fig. 9a shows that LEACS significantly improves performance even when limited by rigid placement of persistence operations. For INTERCEPTDEFAULT, energy efficiency already improves over CHECKPOINT by grouping execution units within the INTERCEPT optimization loop. Integrating LEACS within

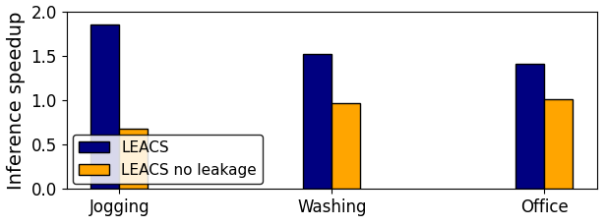


Figure 10: LEACS performance depending on energy source. *Values are averaged across models, MCUs, maximum number of capacitors M, and baselines. The gains of LEACS are more significant for energy-poor sources, where the impact of leakage is greater. Not considering leakage in LEACS produces worse configurations than the baselines, as LEACS systematically employs the largest capacitor.*

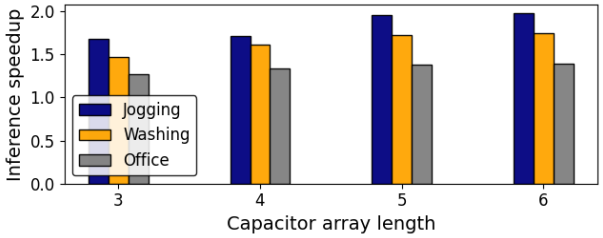


Figure 11: LEACS performance depending on the maximum number M of capacitors from the catalog. *Values are averaged across models, MCUs, and* Checkpoint *as well as* Intercept. *Throughput speedup increases with a higher upper bound on the number of capacitor, but eventually flattens after a certain value of M.*

Intercept, as shown in Fig. 9b, further refines capacitor selection following the placement of state persistence operations.

In the case of SonicDefault, Fig. 9c indicates how applying LEACS lets developers avoid inefficient design choices. On the Jogging source, the $100\mu F$ configuration performs especially poorly: this configuration bleeds away a substantial amount of energy because of leakage even before the device activates, as we illustrate in Fig. 2a. In this scenario, we identify a much smaller capacitor value, taming the effect of leakage and drastically boosting throughput. The strategy we employ in SonicManual minimizes leakage by employing the smallest capacitor that ensures eventual completion. On the other hand, this configuration causes frequent energy failures and incurs high overhead from state persistence operations. LEACS selects a moderately larger capacitor that amortizes this overhead without incurring excessive leakage, yielding again significant improvements as shown in Fig. 9d.

Integrating LEACS within different baselines does *not* compromise semantics. With Sonic, correctness is preserved since only capacitor selection changes, not the execution strategy. Similarly, for pipelines already optimizing persistence operations under a fixed capacitor, as in Intercept, LEACS adds performance by exposing energy storage as an optimization dimension.

Energy source. Fig. 10 summarizes LEACS performance depending on the energy source, averaged across models, baselines, and maximum number of capacitors. The blue bars represent the throughput speedup with the original LEACS. With the energy-poor Jogging source, we achieve the highest throughput speedup, about 1.85× across all settings we explore, while the energy-rich Office source shows a more limited 1.36× speedup. This aligns with our design rationale: when energy is limited, capacitors often dwell near empty, so leakage current represents a large fraction of energy drain. LEACS is especially designed to tame these effects and thus achieves the greatest relative performance improvements. In contrast, when energy is abundant, leakage losses are relatively less important as they are partly compensated by the energy source.

These results confirm that LEACS is most beneficial in energy-scarce environments where minimizing leakage is critical. In energy-rich settings, the advantage is smaller but still appreciable.

Impact of leakage during optimization. The yellow bars in Fig. 10 report the performance of a purposely modified LEACS that *ignores* capacitor leakage when selecting the capacitor array. Interestingly, this may produce configurations that perform even worse than the baselines. This is especially true with the Jogging source, matching our observations in Sec. 3 that energy-scarce sources are most vulnerable to capacitor leakage. In these cases, LEACS naturally tends to use the largest capacitors in the catalog, precisely because it ignores the significant leakage they may suffer, while enabling the longest activation cycles and thus the fewest energy failures. No predetermined configuration in the literature or in our baselines uses capacitors that large.

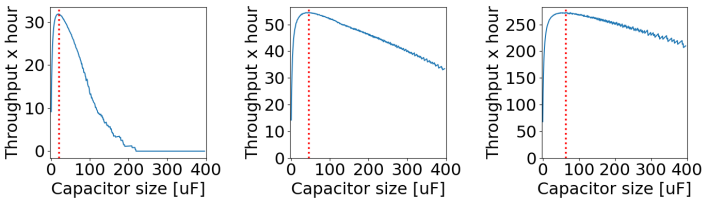
On the other hand, sources that compensate for capacitor leakage, such as Washing and Office, do not particularly suffer from the use of large capacitors. The performance of LEACS with those sources in the yellow bars of Fig. 10 is on par with the baselines.

Number and size of capacitors. Fig. 11 shows the performance of LEACS as a function of the maximum number M of capacitors. We restrict the analysis to the Checkpoint and Intercept baselines here, as they are those employing multi-capacitor configurations. The results are obtained by setting M to the exact number of capacitors employed by the baselines; thus, LEACS can only employ the same number of capacitors as the baselines, or fewer.

Fig. 11 indicates that larger values for M correspond to higher throughput speedup, up to a point. With M set to 3 capacitors, the speedup is relatively modest, corresponding to a 1.42× factor averaged over energy sources, then it increases to a 1.53× factor with M set to 4 capacitors and to a 1.65× factor with M set to 5 capacitors. According to the earlier discussion, the improvements are more pronounced for the Jogging source. Increasing M to 6 capacitors, nonetheless, offers only a tiny additional gain, resulting in a 1.66× speedup. In other words, the benefit of extra capacitors plateaus around $M = 5$ in our tests.

This trend shows that more capacitors mean more degrees of freedom to fit each execution unit with just enough energy and minimal leakage. The Checkpoint baseline chooses one large capacitor to cover the most energy-demanding layer and then selects the others as predetermined fractions of the former, which may leave some layers served by far too large capacitors. In contrast, LEACS systematically searches all configurations. With $M = 5$, essentially every layer in every model is powered by a nearly optimal capacitor, and increasing M further adds negligible benefit.

Fig. 12 shows the symmetrical analysis for SonicDefault, which employ a single capacitor. The plot graphically depicts how throughput varies with capacitor size for a given energy source. The trends at stake align with the throughput versus capacitance trade-off.



(a) Jogging. (b) Washing machine. (c) Office.

Figure 12: Capacitor selection trend with LEACS in SonicDefault. *The trends demonstrate the trade-off between throughput and capacitor size, depending on the energy source. LEACS determines the maximum throughput at a sweet spot that balances the overhead of energy failures, recharge times, and leakage. This sweet spot, indicated by red dashed lines in the figures, occurs at 19 µF for the Jogging trace, 65 µF for the Washing trace, and 78 µF for the Office trace.*

Small capacitors would cause too many energy failures and thus increase the overhead of state persistence operations, whereas large capacitors suffer from leakage. Throughput peaks at an intermediate capacitor size that represents a sweet spot between conflicting dimensions. LEACS precisely identifies this sweet spot.

6 Discussion

Input data precision and availability. The precision and availability of input data might limit the applicability of LEACS. Nonetheless, the accuracy of off-line energy profiling of inference models is vastly demonstrated in the literature, especially when combining profiling experiments on real hardware and datasheet information [2, 8, 10, 13, 22, 42, 47]. Similar considerations apply to the capacitor behaviors, including the charge-discharge dynamics in terms of timings and currents. These trends are accurately documented and characterized by means of empirical measurements and derived analytical models [24, 33, 44, 48].

A more critical aspect may be the availability of energy traces. We use those of Bonito [17], which offers a combination of publicly available datasets. Other similar datasets, nonetheless, also exist [1, 2, 20] and are used even in global competitions [26]. Collecting new traces does require a relevant effort in the implementation of the experimental testbed and in data collection. However, we maintain that *i)* as the field progresses, establishing a benchmarking framework to compare different solutions becomes a necessity, as seen in other closely-related areas [11], and hence additional push is likely to exist to make energy traces publicly available; and *ii)* as hardware platforms specific to intermittent computing emerge [18, 21], the effort to collect energy traces may likely reduce.

Throughput estimation. We use the aforementioned input data in the evaluation of system throughput. The kernel of that estimation is the integral model for computing the energy variation of the capacitor over time. As hinted before, energy consumption is additive and therefore can be decomposed into separate contributions. For those that are represented analytically, such as leakage current, our prototype performs an analytical computation of the integral over the specified time frame. For the other contributions modeled by traces, our implementation mimics the Riemann sums numerical method for a fast, approximate evaluation of the integral. This yields an accurate evaluation of the energy variation over time and, as a consequence, of the throughput achieved by the system.

Hardware variation and harvester aging. Beyond leakage, additional real-world aspects such as hardware variation and harvester aging may influence system performance. In our evaluation, we used the rated capacitance values reported in component datasheets; however, actual capacitance may vary due to manufacturing tolerances and drift over time. Such deviations can be easily accounted for by measuring the actual capacitance of the components intended for deployment and providing these measured values to the LEACS capacitor catalog prior to optimization. This procedure does not require any modification to the optimization algorithm, as it naturally operates on the supplied capacitance values.

Harvester aging represents another source of variability. Our evaluation setup intrinsically captures this effect: the traces used for evaluation consistently provide less available energy than those used during optimization, as we show in Fig. 8. This scenario emulates a harvester whose output has degraded over time, yet LEACS continues to deliver throughput improvements under such conditions. To quantify the expected impact, consider that solar panels typically degrade by $0.6 - 1.75\%$ per year [6], while piezoelectric harvesters can experience up to 32% degradation after 10×10^6 heavy intermittent load cycles (approximately three years of road traffic) [54]. The energy traces we employ thus approximate realistic long-term use. Capacitor aging adds to long-term variation, with capacitance losses ranging from under 1% to 15% after 10,000 hours of operation [56]. This effect reduces available energy storage over time. Similarly to harvester aging, LEACS remains effective as long as the largest capacitor can sustain the most energy-demanding DNN layer; otherwise, re-profiling and re-running LEACS with updated measurements restores optimal operation.

Capacitor technology. LEACS is not tied to any specific capacitor family or technology. Leakage is treated as a parameter within the optimization process: users may supply the leakage characteristics of tantalum, ceramic, or supercapacitors, and LEACS computes the corresponding optimal configuration without requiring any modification to its design. As a result, LEACS is applicabile across a wide range of capacitor technologies.

7 Conclusion

We presented LEACS, a compile-time technique that automatically determines an efficient energy storage configuration for intermittent inference workloads. LEACS tailors the capacitor selection to the dynamics of a given energy source and by taking capacitor leakage into account. LEACS operates by solving a MILP problem coupled with a custom evaluation of system throughput during optimization that is both accurate and avoids introducing non-linearities in the problem formulation. We integrate LEACS with Sonic and Intercept and measure the performance improvements compared to the original designs. LEACS improves inference throughput by up to 3.4×, with an average 1.59× across the settings we test. The benefits are greater on models with higher diversity in the energy demands of execution units, as LEACS matches the capacitor selection to this diversity, and with energy-scarce sources, as the effect of leakage is more pronounced there.

References

[1] 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.

[2] 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.

[3] 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.

[4] A. Alsubhi, S. Babatunde, N. Tobias, and J. Sorber. 2024. Stash: Flexible Energy Storage for Intermittent Sensors. *ACM Trans. Embed. Comput. Syst.* 23, 2 (2024), 23 pages.

[5] ARM Developer. 2024. *ARM Processors*. https://developer.arm.com/Processors Accessed: May 11, 2024.

[6] Doaa M Atia, Amal A Hassan, Hanaa T El-Madany, Aref Y Eliwa, and Mohamed B Zahran. 2023. Degradation and energy performance evaluation of mono-crystalline photovoltaic modules in Egypt. *Scientific Reports* 13, 1 (2023), 13066.

[7] 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.

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

[9] 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.

[10] 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.

[11] Carlo Alberto Boano, Simon Duquennoy, Anna Förster, Omprakash Gnawali, Romain Jacob, Hyung-Sin Kim, Olaf Landsiedel, Ramona Marfievici, Luca Mottola, Gian Pietro Picco, et al. 2018. IoTBench: Towards a benchmark for low-power wireless networking. In *2018 IEEE Workshop on Benchmarking Cyber-Physical Networks and Systems (CPSBench)*. IEEE, 36–41.

[12] Q. Chen et al. 2017. Harvest Energy from the Water: A Self-Sustained Wireless Water Quality Sensing System. *ACM Trans. on Embedded Computing Systems* 17, 1 (2017).

[13] 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)*.

[14] 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.

[15] 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.

[16] Exxelia (Dearborn) Corporation. n.d.. *Film & Mica Capacitors Catalog*. https://exxelia.com/storage/exxelia-assets/medias/Catalogs%20-%20Brochures%20-%20Docs/Catalog/Capacitors/film-capacitors-dearborn-catalog.pdf Accessed: 2025-10-13.

[17] 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.

[18] Kai Geissdoerfer and Marco Zimmerling. 2024. Riotee: An Open-source Hardware and Software Platform for the Battery-free Internet of Things. In *Proceedings of the 22nd ACM Conference on Embedded Networked Sensor Systems*. 198–210.

[19] 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.

[20] 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)*.

[21] 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)*.

[22] J. Hester et al. 2017. Timely execution on intermittently powered batteryless sensors. In *Proceedings of the 15th ACM Conference on Embedded Network Sensor Systems*.

[23] 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.

[24] Innocent S Ike, Iakovos Sigalas, and Sunny Iyuke. 2016. Understanding performance limitation and suppression of leakage current or self-discharge in electrochemical capacitors: a review. *Physical chemistry chemical physics* 18, 2 (2016), 661–680.

[25] 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.

[26] International Conference on Embedded Wireless Systems and Networks (EWSN). [n. d.]. Sustainability Competition—Being Boss at Intermittent Computing. Retrieved July 8th, 2020 from https://ewsn24.tii.ae/call-for-competitors.html

[27] 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.

[28] 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.

[29] 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.

[30] Neal Jackson, Joshua Adkins, and Prabal Dutta. 2019. Capacity over capacitance for reliable energy harvesting sensors. In *Proceedings of the 18th International Conference on Information Processing in Sensor Networks*. 193–204.

[31] 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.

[32] 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.

[33] KEMET Electronics Corporation. 2023. ESK, +85° C: Radial Aluminum Electrolytic Capacitors - Datasheet. KEMET Electronics Corporation. https://content.kemet.com/datasheets/kem_a4004_esk.pdf Datasheet number A4004_ESK; published August 29, 2023.

[34] KEMET Electronics Corporation. n.d.. *Open Mode Design (FO-CAP), X7R Dielectric, 16–200 VDC Surface Mount Multilayer Ceramic Chip Capacitors*. https://www.yageogroup.com/content/datasheet/asset/file/KEM_C1012_X7R_OPENMODE_SMD Accessed: 2025-10-13.

[35] Kyocera-AVX Corporation. n.d.. *High Reliability Tantalum Capacitors — Product Catalog*. https://catalogs.kyocera-avx.com/HighReliabilityTantalum.pdf Accessed: 2025-10-13.

[36] 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.

[37] 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.

[38] 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.

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

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

[41] K. Maeng, A. Colin, and B. Lucia. 2017. Alpaca: Intermittent Execution Without Checkpoints. *Proc. of ACM SIGPLAN Intl. Conf. on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA)* 1 (2017), 1–30.

[42] 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.

[43] Nichicon Corporation. n.d.. *Chip-Type Aluminum Electrolytic Capacitors E1 Series — Product Specifications & Application Notes*. https://www.nichicon.co.jp/english/products/pdf_x/Chip_Type_Aluminum_Electrolytic_Capacitors_E1.pdf Accessed: 2025-10-13.

[44] Atsushi Nishino. 1996. Capacitors: operating principles, current market and technical trends. *Journal of power sources* 60, 2 (1996), 137–147.

[45] Panasonic Corporation. n.d.. *Electrolytic Capacitor AS(T) Series — Product Specification & Application Notes*. https://industrial.panasonic.com/cdbs/www-data/pdf/RDF0000/ast-ind-152837.pdf Accessed: 2025-10-13.

[46] Panasonic Corporation. n.d.. *S_Film Capacitor Catalog*. https://industrial.panasonic.com/content/data/CP/PDF/S_Film_cap_catalog_e.pdf Accessed: 2025-10-13.

[47] B. Ransford et al. 2011. Mementos: System Support for Long-running Computation on RFID-scale Devices. *ACM SIGARCH Computer Architecture News* 39, 1 (2011).

[48] Silicon Labs. 2023. Current Leakage through Ceramic Capacitors - Application Notes. https://www.silabs.com/documents/public/application-notes/an1415-current_leakage_through_ceramic_capacitors.pdf

[49] STMicroelectronics. 2020. Aluminium Electrolytic Capacitors datasheet. https://industrial.panasonic.com/cdbs/www-data/pdf/RDF0000/ABA0000C1259.pdf. Accessed: July 1, 2025.

[50] STMicroelectronics. 2025. *STM32H742 – High-performance Arm Cortex-M7 MCU.* https://www.st.com/en/microcontrollers-microprocessors/stm32h742.html Accessed: 2025-07-01.

[51] STMicroelectronics. 2025. *STM32L4 Series Documentation.* https://www.st.com/en/microcontrollers-microprocessors/stm32l4-series/documentation.html Accessed: 2025-07-01.

[52] STMicroelectronics. 2025. *STM32U5 Series Documentation.* https://www.st.com/en/microcontrollers-microprocessors/stm32u5-series/documentation.html Accessed: 2025-07-01.

[53] Vishay Intertechnology, Inc. n.d.. *T95 — Tantalum Chip Capacitors.* https://www.vishay.com/docs/40081/t95.pdf Accessed: 2025-10-13.

[54] Jun Wang, Xiangzhen Qin, Zhiming Liu, Guangya Ding, and Guojun Cai. 2021. Experimental study on fatigue degradation of piezoelectric energy harvesters under equivalent traffic load conditions. *International Journal of Fatigue* 150 (2021), 106320.

[55] 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.

[56] Jiale Xu, Lei Gu, and Juan Rivas-Davila. 2019. Effect of class 2 ceramic capacitor variations on switched-capacitor and resonant switched-capacitor converters. *IEEE Journal of Emerging and Selected Topics in Power Electronics* 8, 3 (2019), 2268–2275.

[57] 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.