Special Session - Intermittent TinyML: Powering Sustainable Deep Intelligence Without Batteries

Hashan Roshantha Mendis
Academia Sinica, Taiwan
rosh.mendis@citi.sinica.edu.tw

Kasım Sinan Yıldırım
University of Trento, Italy
kasimsinan.yildirim@unitn.it

Marco Zimmerling
TU Darmstadt, Germany
marco.zimmerling@tu-darmstadt.de

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

Pi-Cheng Hsiu
Academia Sinica, Taiwan
National Taiwan University, Taiwan
pchsiu@citi.sinica.edu.tw

Abstract

Tiny battery-free devices running deep neural networks (DNNs) embody *intermittent TinyML*, a paradigm at the intersection of intermittent computing and deep learning, bringing sustainable intelligence to the extreme edge. This paper, as an overview of a special session at Embedded Systems Week (ESWEEK) 2025, presents four tales from diverse research backgrounds, sharing experiences in addressing unique challenges of efficient and reliable DNN inference despite the intermittent nature of ambient power. The first explores enhancing inference engines for efficient progress accumulation in hardware-accelerated intermittent inference and designing networks tailored for such execution. The second investigates computationally light, adaptive algorithms for faster, energy-efficient inference, and emerging computing-in-memory architectures for power failure resiliency. The third addresses battery-free networking, focusing on timely neighbor discovery and maintaining synchronization despite spatio-temporal energy dynamics across nodes. The fourth leverages modern nonvolatile memory fault behavior and DNN robustness to save energy without significant accuracy loss, with applicability to intermittent inference on nano-satellites. Collectively, these early efforts advance intermittent TinyML research and promote future cross-domain collaboration to tackle open challenges.

Keywords

Intermittent systems, TinyML, runtime and design-time methodologies, hardware architectures, wireless networking, fault tolerance

1 Introduction

By relying entirely on energy harvesting, battery-free devices enable a low-cost, greener future. However, the weak and unstable nature of ambient energy causes frequent shutdowns to recharge the capacitor, with the device powering on when sufficient energy is harvested. Consequently, applications run intermittently and use nonvolatile memory (NVM) to accumulate progress across power cycles, as volatile data is lost upon power failure. Nevertheless, intermittent deep learning is essential for intelligent edge applications reshaping industries and society, motivating a multi-pillar research approach to improve overall system efficiency. These efforts are presented through four tales of intermittent TinyML, spanning runtime engines for intermittent inference and design-time tools for intermittent-friendly DNNs, lightweight algorithms and emerging hardware for fast and energy-efficient inference, coordinated communication in battery-free networks, and embracing faults for reliable intermittent inference in extreme environments. The advancements extend beyond embedded systems to sensor networking, operating systems, and computer architecture, offering perspectives that enrich broader embedded software interests.

Our exploration begins in Section 2, where we discover how simply adapting conventional intermittent execution methods for resource-demanding, complex applications like DNN inference leads to critical performance inefficiencies that may offset the benefits of accumulative execution. This motivated us to progressively develop runtime optimizations for hardware-accelerated intermittent inference engines that minimize progress accumulation overhead, while accounting for the dynamic nature of both the power supply and the neural network. While these engines execute the deployed models efficiently, their effectiveness is tightly coupled with the DNN architecture. However, frameworks commonly used to design DNNs for tiny devices often yield architectures inherently unsuitable for intermittent systems, leading to energy-wasteful, slow deployments that could even fail to complete inference. Therefore, we developed several design-time tools, adapting neural architecture search and neural network pruning to derive architectures optimized for intermittent execution.

Complementing prior efforts, Section 3 examines how the intermittent execution of DNNs on ultra-low-power microcontrollers remains highly inefficient, rooted in the computational demands of DNN inference and the architectural limitations of conventional Von Neumann processors. Firstly, traditional DNNs rely heavily on multiply-and-accumulate (MAC) operations, all of which are executed nonselectively during inference, leading to significant energy consumption. Moreover, Von Neumann-based cores are inherently inefficient, as they waste most of their energy on control operations and suffer from the data transfer bottleneck between the processor and the memory units. These inefficiencies have motivated us to explore alternative inference algorithms that can activate only the most relevant components of the network for each input, significantly enhancing energy efficiency while maintaining

EMSOFT '25, Taipei, Taiwan

ACM ISBN 979-8-4007-1993-6/2025/09
https://doi.org/10.1145/3742874.3757084

accuracy. In parallel, to address the architectural constraints of Von Neumann cores, we leverage the emerging computing-in-memory (CIM) paradigm, which enables computation directly within memory, removing frequent data transfer between the processor and memory and providing inherent resilience to power failures.

Although inference on individual battery-free devices is well studied, Section 4 shifts focus to the underexplored challenge of communication across multiple devices, essential for distributed intelligence. We first noticed that the ability to systematically evaluate battery-free networks and measure community progress was sorely missing. Therefore, we developed tools to accurately capture and reproduce real-world energy environments facilitating repeatable lab-based evaluation, making them publicly and remotely accessible for broad use. Due to the spatio-temporal variability in harvested energy, nodes operate asynchronously, making it challenging for networked nodes to discover each other quickly and efficiently, and to ensure reliable bidirectional communication. To address this, we design networking protocols to align device wake-ups through adaptive random delays for fast neighbor discovery and maintain synchronization by learning charging-time patterns and adapting device wake-up times accordingly.

Largely unexplored knobs exist that may drastically improve the inference performance of DNNs on resource-constrained intermittent devices. Emerging NVM technology is a key example. Spin-Transfer Torque Magnetic Random-Access Memory (STT-MRAM), for instance, offers the ability to save energy in exchange of accepting that write errors of stochastic nature possibly occur. Taking advantage of these knobs without introducing unnecessary overhead is challenging. We report in Section 5 on the design of compile-time techniques that build upon the features of STT-MRAM to markedly improve the energy efficiency of DNN inference by imposing a hard bound on the potential accuracy losses. Embracing the faults of STT-MRAM prompted us to look in the direction of novel deployment scenarios. We eventually realize that much of the same techniques may be applied to running intermittent inference workloads on nano-satellites, provided that the fault patterns are comparable. To verify this hypothesis experimentally, by deploying a CubeSat in Low-Earth Orbit (LEO) equipped with different NVM chips. Early results, also discussed in Section 5, indicate that fault patterns are indeed comparable, providing a foundation for deploying efficient intermittent DNN inference in space operations.

2 Intermittent Deep Inference: Rethinking Engines and Models

The overarching goal of advancing intelligent battery-free systems is to enable increasingly complex deep learning models to run efficiently on tiny devices powered by extremely weak ambient energy. Our research efforts over the past few years have evolved toward this goal through advances in runtime inference engines and design-time tools. We also highlight carbon footprint reduction as a central unresolved concern in TinyML sustainability.

2.1 Intermittent-aware Runtime Inference Engines

DNN inference is executed at runtime by an *inference engine*, a software-based middleware processing each layer, as shown in Figure 1. Even with DNN optimization [57] and hardware acceleration

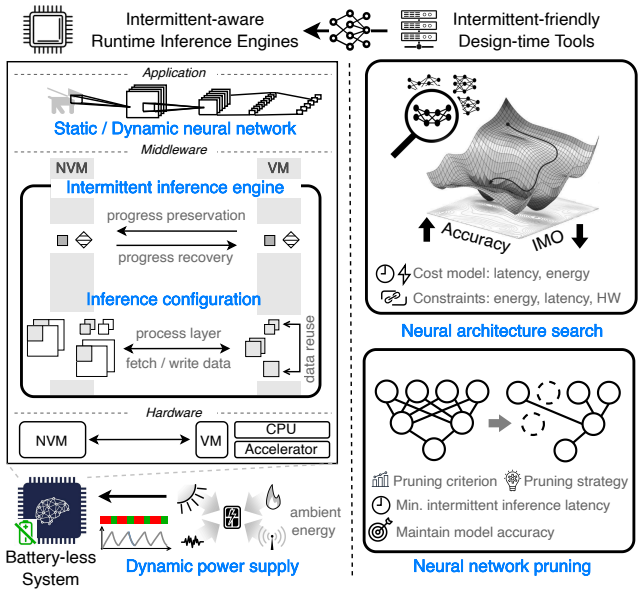

Figure 1: Overview of runtime inference engines and design-time tools for intermittent deep inference.

[40, 53], completing a single inference under intermittent power may still require multiple power cycles. Therefore, *intermittent inference engines* additionally perform progress preservation during inference and progress recovery upon power resumption to accumulate progress across power cycles [27, 30]. Progress preservation involves frequently backing up a progress indicator together with intermediate computation outputs (e.g., activations) from VM to NVM. Progress recovery uses the preserved indicator to correctly identify the interrupted computation, refetch the lost data back into VM from NVM, and resume execution without starting from scratch. Progress preservation and recovery introduce significant memory traffic and latency, with even simple DNNs often requiring several seconds to minutes for completion [13, 27]. Intermittent inference approaches differ in the type and granularity of data they preserve, impacting the progress preservation overhead, and the amount of progress re-executed during recovery.

To understand how such overheads impact inference performance, we applied general intermittent execution approaches, such as checkpoint-based and task-based ones, to DNN inference. Checkpoint-based execution backs up volatile data to NVM, and resumes from the last successful checkpoint. We found that the high memory usage of DNN inference leads to large checkpoints with significant runtime overhead. By contrast, task-based execution decomposes applications into multiple tasks that can each complete within the available energy in a power cycle, with progress preserved after each task completion and execution resumed from the interrupted task. As DNN inference contains loop-heavy computations, each loop iteration is typically treated as a task [27]. Loop indices are preserved as progress indicators, and the entire interrupted task is re-executed during recovery. Since only full accelerator operations can be invoked, tasks cannot be finer than an operation and usually group multiple operations to reduce preservation overhead. This causes task-based inference to inherently demonstrate high re-execution overhead. Therefore, the high overhead of these general approaches demands more energy, which presents a key challenge,

as it raises the risk of non-termination (i.e., repeated re-execution) when energy is scarce.

2.1.1 Hardware Accelerated Inference. To overcome the high runtime overhead of general approaches, we proposed *inference footprinting* [54] specifically for hardware accelerated intermittent inference. Inference footprinting preserves each intermediate computation output, (e.g., as fine-grained as a partial sum) along with a progress indicator during inference, and uses the latest indicator to identify and resume the interrupted computation upon power resumption. This fine-grained progress preservation enables parallel inference computation and progress preservation, which reduces the overhead of progress preservation. Moreover, it relaxes the accelerator operation's atomicity constraint, requiring re-execution of only the interrupted sub-operation instead of the entire operation, greatly reducing the overhead of progress recovery. With reduced runtime overhead, footprinting-based inference is less prone to non-termination under weak power. Realized as HAWAII [54], an intermittent inference engine, footprinting-based inference substantially improves inference throughput over both checkpoint-based and task-based inference, and is especially effective for heavily accelerated DNNs executed under small energy budgets.

Later on, we found that separately preserving progress indicators and output features introduces additional data transfer overhead, potentially offsetting the accumulated progress. To alleviate this issue, we proposed *model augmentation* [32], allowing indicator preservation to be piggybacked onto output feature preservation. By appending additional network components with assigned values at specific positions in the deployed model, progress information is intrinsically integrated into inference, trading extra computations for reduced data transfer overhead, without affecting accuracy. Implemented as the JAPARI inference engine [32], this approach achieved notable latency reductions over HAWAII, particularly for highly accelerated networks under limited energy budgets. We advanced this idea with *stateful neural networks* [60], which embeds progress indicators into specific network components, allowing a DNN to indicate progress itself. A stateful DNN can contain progress information in its output features without their corruption. Progress indicators are intrinsically preserved with output features, completely eliminating the additional data transfers needed for indicator preservation. Realizing this approach, the Stateful inference engine [60] significantly speeds up inference over JAPARI, notably for modern convolutional networks under weak power.

2.1.2 Dynamically Reconfigurable Inference. Inference engines typically use a fixed *inference configuration* with parameters such as tile size and loop order to maximize data reuse, thereby reducing costly NVM accesses [33]. However, when using a fixed configuration under intermittent power, we found that this strategy became inefficient, as retaining more data in VM increases the amount that must be refetched during progress recovery. This issue is intensified as recovery overhead now dominates, driven by our prior efforts to minimize preservation overhead. Thus, minimizing intermittent inference latency requires balancing data reuse during inference and data refetch during recovery. Nevertheless, we observed that the balance point fluctuates depending on the dynamicity of the power supply driven by ambient energy, posing a major challenge as a fixed configuration becomes significantly inefficient. Accordingly, we emphasized the necessity of *runtime reconfiguration* [56], to maintain low latency under a varying power supply. This concept was realized as the DynBal middleware [56], which indirectly evaluates the performance of configurations considering input power, and dynamically reconfigures the underlying inference engine in a lightweight manner, balancing data reuse and refetch costs under fluctuating power conditions. DynBal demonstrated that even a simple implementation of runtime reconfiguration can significantly speed up inference, especially under high intermittency.

Orthogonally, we found that networks with *dynamic* behavior introduced additional challenges when executed on our intermittent inference engines (Section 2.1.1). Fundamentally, existing intermittent inference approaches assume deterministic execution, where predefined computation order is required for correct recovery, and therefore they directly support static networks. By contrast, dynamic networks exhibit inherently *non-deterministic execution*, adapting their structure to the input to improve accuracy and performance trade-offs, thereby reordering execution via computation skipping [29, 51]. We observed that existing approaches failed to capture non-deterministic progress information in dynamic networks, which vary across inferences. This poses a key challenge, as it leads to incorrect recovery, corrupting output features and degrading accuracy. Therefore, we recently proposed *non-deterministic progress accumulation* [55], a methodology for capturing the interrupted computation and its non-deterministic information, with minimal preservation overhead. Low-overhead progress indicators are identified based on network structure and indicator update frequency, and their collective update patterns are leveraged to minimize the data volume and number of transfers for progress preservation. Our NodPA middleware [55] enhances HAWAII by implementing this methodology, ensuring correct and fast inference, primarily for highly dynamic networks.

2.2 Intermittent-friendly Design-time Tools

While inference engines execute a given pre-trained DNN model with a fixed architecture, designing DNNs for resource-constrained systems commonly uses two complementary approaches: *neural architecture search* (NAS) [31, 36] and *neural network pruning* [57, 59], as shown in Figure 1. As the network architecture design space is vast, NAS is often used to automatically design a high-accuracy DNN for the target dataset. In a typical NAS framework, the architecture search space defines the candidate set of architectures, and the search strategy is the algorithm that optimizes a specific objective (e.g., accuracy). Hardware-aware NAS extends this process by jointly exploring the network architecture and inference configuration design spaces, incorporating hardware metrics (e.g., latency) into the optimization objective, and enforcing resource constraints (e.g., VM and NVM capacity) to ensure the derived DNN is both deployable and efficient on the target system. In contrast, network pruning compresses a given model, by removing relatively unimportant weight parameters, trading accuracy for lower storage requirements. Pruning approaches differ in their pruning criterion which estimates the importance of weights and their pruning strategy which removes relatively less importance weights to achieve the optimization objective while maintaining model accuracy.

2.2.1 Intermittent-aware Neural Architecture Search. Deploying NAS solutions under intermittent power revealed that conventional NAS frameworks, being unaware of intermittency, are unsuitable

for intermittent systems. We observed that NAS tends to find solutions that maximize data reuse for lower latency [31], making them energy-expensive, and since NAS ignores configuration spaces related to intermittent execution, the harvested energy may be underutilized. Thus, a fundamental challenge is that, the derived DNNs may become *inefficient* by violating the intermittent inference latency requirement and, more seriously, they may even become *unsafe* by experiencing non-termination. In light of this, we presented iNAS [41], the first *intermittent-aware NAS* framework that incorporates intermittent execution behavior into NAS. It follows a general principle that finding high-accuracy DNNs for intermittent systems requires balancing data reuse and progress preservation and recovery overheads, without exceeding the harvested energy in each power cycle. Unlike hardware-aware NAS solutions, iNAS's solutions avoided non-termination, met latency requirements with comparable accuracy, and achieved greater latency reductions for complex architectures under small energy budgets.

Although safe and timely, we noticed that iNAS solutions still imposed high *intermittency management overhead* inherent to the network architecture, wasting harvested energy on progress preservation and recovery instead of inference, which degraded latency. We further observed that directly minimizing intermittency management overhead in NAS compromises accuracy, a challenging issue because it risks overlooking high-accuracy architectures in favor of ones with low overhead and also low accuracy. This motivated us to examine how architectural characteristics influenced the accuracy-overhead interplay. Our study [42] showed architectural parameters differ in their *sensitivity* to overhead because their computation to combined preservation and recovery cost ratios vary with their distinct computation and data access behaviors. To design DNNs with better overhead–accuracy trade-offs, we derived general guidelines that exploit overhead sensitivity and integrated them into TiNAS [42], a modernized intermittent-aware NAS framework, refining its search space and strategy. These guidelines enabled TiNAS to efficiently identify low-overhead, high-accuracy DNNs, especially for larger datasets with broader search spaces.

2.2.2 Intermittent-aware Neural Network Pruning. Similar to NAS, conventional network pruning frameworks are also unaware of intermittent execution, assuming deployment under continuous power [59]. We observed that the hardware usage of a DNN optimized for continuous power differs significantly from a DNN optimized for intermittent power. In continuous inference, computed outputs are kept in VM to maximize data reuse, so NVM reads and accelerator computations dominate inference latency. In contrast, intermittent inference frequently preserved accelerator outputs and progress indicators, making NVM writes dominant. Existing pruning approaches improve continuous inference latency by reducing the number of NVM reads and computations. Therefore, the main challenge is that, a model pruned for continuously-powered systems may perform suboptimally on intermittent systems. We addressed this with iPrune [35], a simple yet effective *intermittent-aware pruning framework* that reduces intermittent inference latency while maintaining accuracy. Its core idea is to remove weights that contribute more to intermittent inference latency but have low sensitivity to accuracy. This provided higher compression compared to energy-aware pruning with similar accuracy, while consistently speeding up intermittent inference under various power profiles.

2.3 Powering a Low-Carbon Future

We summarize the key insights of this section as follows:

- Hardware-accelerated intermittent inference enables fine-grained progress preservation in parallel with accelerator computation, reducing the risk of non-termination.
- Runtime inference reconfiguration balances data reuse and refetch for low latency under dynamic power, while capturing non-determinism correctly recovers dynamic networks.
- While incorporating intermittent execution behavior into NAS can find safe models, directly minimizing intermittency management overhead may compromise accuracy.
- Efficient intermittent inference requires tailored compression, as pre-optimized compact models may become suboptimal when executed intermittently.

Nevertheless, TinyML's *carbon footprint*[46] poses ongoing sustainability challenges. Its *operational emissions* arise from inference energy consumption scaled by the power source's carbon intensity, while *embodied emissions* stem from the amortized carbon impact of hardware manufacturing, transportation, and disposal over the device's lifetime inferences. Although runtime optimizations (Section 2.1) can reduce energy use, they may not lower operational emissions if inference occurs during periods of high carbon intensity. Likewise, maximizing data reuse reduces latency but increases NVM wear by writing larger blocks, shortening device lifetime and limiting embodied carbon amortization. Inference engines can mitigate these issues by scheduling tasks based on priority and carbon intensity variation over time and location, while balancing data reuse and NVM writes to maintain low latency and device longevity. Moreover, at design time, tools (Section 2.2) may overlook the model-device synergy that shapes the overall carbon footprint. For instance, power supply units and sensors contribute significantly to embodied emissions [46]. Although increasing the size of these components improves performance, supports longer deployments, and allows more accurate, complex DNNs, it also leads to higher embodied emissions. Therefore, the interplay between the DNN model, device specifications, and carbon emissions should be considered early in system design by exploring combined design spaces that holistically optimize for performance and sustainability.

3 Fast Inference on Emerging Hardware

In this section, we emphasize that the inefficiency of intermittent inference arises mainly from two factors: (1) the significant computational overhead associated with traditional DNN models, and (2) the inherent architectural limitations of conventional Von Neumann-based ultra-low-power microcontrollers. We present our recent efforts targeting alternative inference algorithms and hardware architectures to improve the efficiency of intermittent inference.

3.1 Algorithmic and Architectural Inefficiencies

DNN models are computationally heavy for battery-free sensors. Even a simple DNN layer has thousands of MAC operations. During deep inference, all DNN layers are executed non-selectively and sequentially, introducing a significant computational burden. Furthermore, to ensure resilience against power failures across power cycles, input and output activations of each layer should be stored in NVM. This backup process generates significant memory traffic and latency, resulting in slow and energy-inefficient inference [62]. As

a result, DNN inference can take several seconds to minutes, even for simple models [20]. Furthermore, DNNs are latency-agnostic and cannot adapt to the instability and unpredictability of ambient energy sources, which often result in high latency in battery-free systems. Therefore, we need a different inference approach that is computationally fast, offers minimal state retention and backup overheads, is responsive to energy harvesting dynamics, and is compact to fit within the limited memory budgets of these platforms.

Besides, ultra-low-power microcontrollers are still inefficient. Battery-free devices are typically built around simple ultra-low-power programmable cores [10, 25]. These Von Neumann and Harvard cores are not optimized for energy efficiency since they waste a significant amount of energy for control and datapath operations [28], spending only 5–10% of available energy on useful computations [26]. Furthermore, memory-intensive inference workloads exacerbate the computational inefficiency due to frequent data movement between memory and compute units, creating data transfer bottlenecks [7, 13]. Besides, these cores lack power-failure resilience and lose computational state during power outages [48, 64]. State backup and recovery operations create a significant data transfer bottleneck between the processor and NVM. Another important aspect is the parallel execution of inference workloads. Most battery-free computing platforms are single-core and do not offer parallelism, which is critical for the efficient execution of DNNs [5]. In short, we need new power failure-resilient architectures that remove the mentioned inefficiencies, offer parallelism, and respond to the changes in the environmental power availability, using the power and energy most optimally for the efficient intermittent execution of battery-free inference.

3.2 Fast Inference for Battery-free Sensors

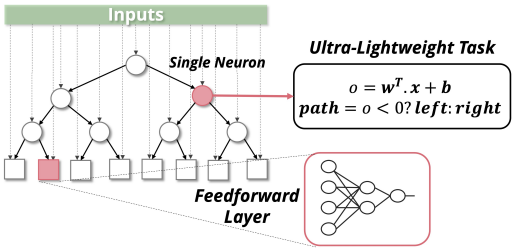

Figure 2: Fast-Inf Inference Algorithm. o is the output, x is the input, w is the weight vector, and b is the bias of the node.

To address the computational challenges, we introduced Fast-Inf [15], a new lightweight inference algorithm that enables ultra-fast inference on battery-free devices. Fast-Inf utilizes a binary tree-based neural network architecture, offering logarithmic time complexity with minimal backup and runtime memory requirements. This architecture exploits conditional computation [12], activating only the most relevant network components for each input, thereby significantly enhancing energy efficiency while maintaining accuracy. At the core of Fast-Inf is the Fast Feedforward (FFF) network [12], a binary tree-based architecture (Figure 2) where each internal node represents a single neuron and each leaf corresponds to a small feedforward subnetwork. Inference is a tree traversal, as depicted in Figure 2, where each neuron computes an intermediate output o via a dot product, which choses the left or right branch. This procedure is reiterated until a leaf is reached, whose outputs provide the network outputs.

Fast-Inf exploits conditional execution since only a part of the tree is considered during inference, which enables logarithmic inference time, achieving ultra-fast, energy-efficient performance under tight resource constraints. During inference, the information that needs to be checkpointed by the intermediate nodes is just the decision variable o_i, which is only a single bit. Besides, we just need to check the last decision variable to proceed with the decided leaf node. So the recovery is also very lightweight.

Fast-Inf is best suited for tasks solvable by fully connected networks (FCNs), as its architecture effectively decomposes FCNs into a tree structure. Consequently, it achieves performance comparable to FCNs in applications where instantaneous sensor readings are mapped to discrete classes, e.g., human activity recognition. For tasks with strong spatiotemporal structure, Fast-Inf remains effective for small-scale inputs, e.g., keyword spotting. Overall, Fast-Inf significantly reduces memory usage (up to 6× fewer parameters and 4420× smaller runtime buffers), achieves up to 700× faster inference with lower energy consumption, and introduces a lightweight inference engine (6× smaller code and 1000× lower overhead).

3.3 Intermittent Inference in Nonvolatile Memory

We addressed hardware-level inefficiencies by utilizing the emerging CIM paradigm [44], which improves both energy efficiency and inference throughput. As mentioned, backing up computational states to NVM frequently incurs substantial time and energy overhead due to increased memory traffic, especially for memory-bound workloads [50]. The CIM paradigm offers a promising solution by enabling computation directly within memory, removing the need for explicit backup operations and providing inherent resilience to power failures.

Computational Random Access Memory (CRAM) is an emerging spintronics-based memory array that can perform logic operations directly in memory cells and store the output [63]. This contrasts with traditional Von Neumann architectures, which transfer data from memory to processor registers for processing, then storing the results back in registers before transferring them again to memory. CRAM with NVM elements is ideal for intermittent CIM since it is *power-failure resilient*, ensuring that each in-memory operation is failure-atomic [47, 48]. This eliminates the need for explicit backups and costly data movement, enabling efficient, failure-atomic computation of memory-bound workloads under intermittent power, with high parallelism and minimal overhead.

We introduced PiMCo [6], a flexible MCU-based system where the CRAM serves both as an NVM and a CIM accelerator for intermittent inference. As shown in Figure 3, our specialized memory controller bridges the microcontroller with CRAM. It is responsible for receiving high-level instructions from the microcontroller and triggering necessary operations in situ on CRAM. CRAM supports only primitive logic instructions, such as AND, OR, and XOR. We introduced macroinstructions that are a sequence of these primitives. Macroinstructions consist of arithmetic and logical operations, such as multiplication, addition, logical and, movement, and loop operations that help to implement more complex operations. Developers can use high-level C interfaces to map inference workloads

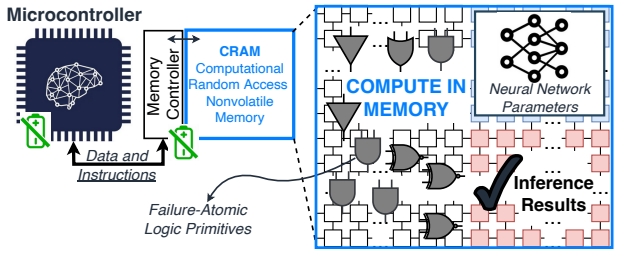

Figure 3: CRAM and PimCo memory controller enables efficient and programmable intermittent CIM.

to CRAM. Our memory controller converts the received high-level MCU instructions and maps them to a sequence of macroinstructions, which define the sequence of primitive operations on CRAM.

By executing inference workloads on CRAM, programmers can benefit from a high degree of parallelism and acceleration. We utilized column-level parallelism, allowing the same CRAM primitive logical operation to be applied to multiple columns in CRAM simultaneously. For example, in convolutional neural networks, the same kernel is used on different sections of the input image to generate a convolved output feature matrix. As a result, PiMCo executes these repeated operations, which consist of independent multiply-and-accumulate operations, in parallel.

CRAM primitive operations are inherently idempotent and can safely be restarted upon power recovery. Furthermore, our memory controller is power failure resilient by design since it has an embedded NVM to back up the progress of the execution of the macroinstructions on CRAM. Therefore, PiMCo ensures failure-atomic acceleration of inference workloads on CRAM. Besides, the MCU can employ several checkpointing strategies to ensure the correct intermittent execution of other parts of the application.

Adapting to energy harvesting dynamics is critical for intermittent systems: activating more parallelism on CRAM can increase throughput when ambient power is sufficient, but doing so under low power may cause frequent failures, reduced throughput, or even computational failures. Without dynamic adaptation, CIM systems miss the opportunity to optimize performance based on available energy. We implemented a runtime library that selects the best parallelism configuration based on ambient energy availability. Overall, PiMCo improves the performance of the state-of-the-art commercial low energy accelerator (LEA) [53] for battery-free systems by up to 8× and energy efficiency by up to 150×.

3.4 Towards In-Sensor Battery-free Inference

Our recent works have shown that eliminating algorithmic and architectural inefficiencies leads to significant energy savings and improvements in throughput and latency. This requires:

- Inference algorithms that can be executed adaptively by activating only a part of the model.
- Power failure resilient architectures that minimize the data traffic between the processor and NVM during inference.

Looking forward, the in-sensor computing paradigm can maximize energy and power efficiency in battery-free inference systems. Bringing inference computations closer to the sensor—where data is generated—drastically reduces data movement overhead, achieving orders-of-magnitude energy efficiency gains compared to conventional architectures. Embedding CRAM in battery-free sensors [47] and running computationally efficient inference algorithms like Fast-Inf directly on CRAM will bring unprecedented efficiency, which is crucial for energy harvesting systems.

Fast-Inf showed the potential of adaptive and conditional inference computation, yet significant scope for further exploration remains. Mixture-of-experts, early exit mechanisms, and concepts from dynamic networks [29] can be blended to optimally activate only the most relevant subnetworks based on input characteristics and available energy. Lightweight controllers can be designed to direct inference decisions, determining which subnetworks to activate and which expert paths to select.

4 Efficient and Reliable Battery-free Networks

Previous sections enabled reliable and efficient DNN inference on standalone battery-free devices, but these devices need to communicate their results, to realize distributed learning applications. Therefore, the next big goal is scaling to many devices operating maintenance-free for decades [4], which requires new methods and tools for efficient and reliable battery-free wireless networking.

4.1 The Battery-free Networking Challenge

Networking is crucial for time synchronization, sensor calibration, federated learning, and distributed sensing and control. Although wireless communication between battery-free devices and continuously-powered base stations have been successful [9], enabling direct communication between battery-free devices remains important. Since long-range transmissions require more power, large energy storage is needed for atomic wireless transmission, which is unsustainable. Therefore, a mesh network with short hop relays is fundamentally more efficient and sustainable, as well as inherently more reliable and scalable [34].

For real-world IoT applications, wireless networking must satisfy four core requirements: *reliability* with 99.999% multi-hop message delivery, *predictability* of end-to-end reliability and latency at design time and runtime, *adaptability* to environmental changes and varying application demands, and *efficiency* within strict memory, energy, bandwidth, and compute limits. These goals are already challenging in battery-powered networks with unreliable links and changing topologies, while battery-free networks face the added challenge of *spatio-temporal energy dynamics*. The harvested energy depends on time and location. For instance, bridge sensors harvest varying vibrational energy, based on when and where a car crosses. As battery-free devices operate intermittently for brief intervals due to fluctuating harvested energy, spatial variability across a network causes asynchronous behavior, where some devices are active while others are recharging. The same holds for wireless power transfer in future 6G ambient IoT, where received power depends on environment and mobility. This makes communication challenging, as the sender and receiver must be powered simultaneously, and the difficulty grows with the network size.

Existing low-power wireless protocols for battery-based devices primarily focuses on saving energy by synchronizing radio-on times [1], which is inapplicable to battery-free systems because they cannot become active anytime. Unlike radios, backscatter transceivers communicate by reflecting radio signals (e.g., TV and Wi-Fi), with prior research focusing on improving range and throughput by using cables or batteries and avoiding intermittency through abundant ambient energy [37, 39].

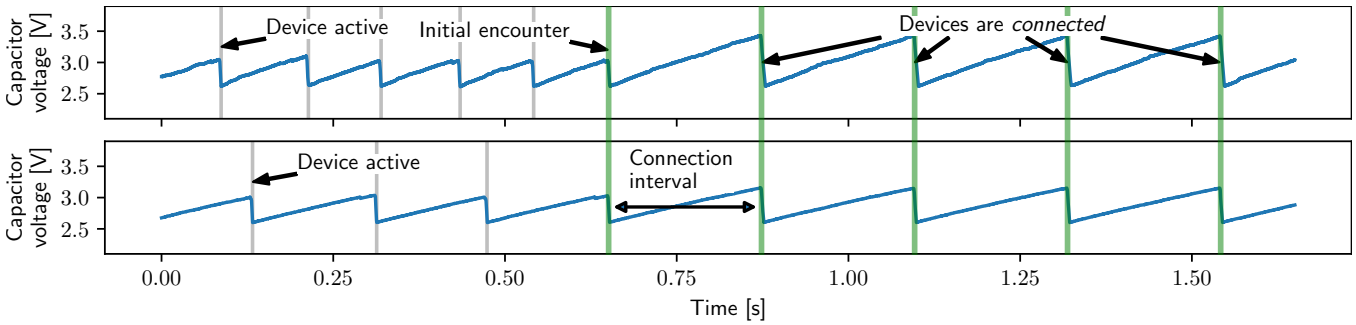

Figure 4: Using Bonito, battery-free devices learn and exchange statistical models of their charging times with the goal of maintaining a connection across consecutive encounters for efficient bi-directional unicast communication. (Taken from [24].)

4.2 Methods and Tools to Bootstrap Battery-free Networks

To close this research gap, we began in 2018 by asking: *Can we exploit the spatio-temporal characteristics of real-world energy environments to enable efficient and reliable battery-free networks?* To answer this question, we developed a suite of tools and methods. This includes the Shepherd [21] and Shepherd Nova [22] tools for capturing energy environment characteristics, and the Find [23] and Bonito [24] methods for enabling energy-aware synchronization and communication. These methods were efficiently implemented on real hardware using Riotee [25], our open-source, commercially available, battery-free, hardware-software platform.

4.2.1 Capturing and Reproducing Energy Environments. An important challenge is to conduct realistic and repeatable battery-free networking experiments in the lab, while also being able to benchmark community progress [4]. Shepherd [21] is a portable testbed with distributed nodes synchronized via GPS or Ethernet using PTP. In recording mode, Shepherd nodes capture harvesting voltage and current at high resolution (3 µA, 50 µV, 100 kHz) with tight node synchronization (about 1 µs), enabling detailed analysis of spatial and temporal energy dynamics. The traces have revealed some interesting properties, such as similar harvesting current patterns across networked devices. In replay mode, nodes emulate energy traces for battery-free devices under tightly synchronized conditions, allowing realistic and repeatable lab testing. Advancing these ideas, Shepherd Nova [22], offers a free, public, remotely accessible testbed for energy-harvesting experiments on shared infrastructure, enabling community progress to be objectively measured. Compared to Shepherd, Shepherd Nova supports diverse input formats (e.g., IV surfaces) and emulates the full harvesting circuitry and energy storage, allowing tests of different capacitors and converters with high precision and accuracy, closely mirroring real-world setups. These testbeds allowed the design and evaluation of the following battery free networking solutions.

4.2.2 Observing and Adapting to the Energy Environment. A key challenge in battery-free networks is enabling devices to discover each other and maintain synchronization. Devices wake up asynchronously due to varying energy availability, leading to hundreds of missed encounters before the devices become active at the same time by chance. Find [23] aims to quickly generate a first encounter, enabling timely neighbor discovery and clock synchronization through bidirectional message exchange. Interleaved wake-up patterns are broken via randomized delays drawn from a geometric distribution, which each node adapts at runtime based on variations in local charging times. Bonito [24] helps to maintain synchronization after an initial encounter by having devices agree on a new connection interval at each meeting (as shown in Figure 4), ensuring future encounters with a user-defined probability. The core idea is that each device learns and updates a statistical model of its charging times, and shares the parameters during encounters to compute the next wake-up time, enabling reliable message exchange across consecutive wake-ups. As charging times often follow well-known distributions (e.g., gaussian or exponential), lightweight statistical methods were used to learn the distribution parameters.

4.3 Learning for Reliable and Timely Communication

The main takeaways of this section is as follows:

- Understanding spatio-temporal energy availability is crucial to developing practical methods for battery-free networks.
- Precise and accurate energy harvesting traces are key for realistic and repeatable battery-free networking experiments.
- Randomized waiting breaks the interleaved wake-up patterns of battery-free network nodes, minimizing neighbor discovery latency.
- Learning the charging pattern and adapting the wake-up times allows for efficient communication between neighbors.

With perfect knowledge of the underlying charging-time distribution, our solutions (Section 4.2.2) can compute the minimum feasible connection interval. Therefore, a promising future direction is to utilize a DNN model that can learn complex, multimodal charging-time patterns at runtime, leading to increased reliability and reduced delays. Moreover, for further enhanced prediction, nodes can exchange key parameters or summaries of their learned charging-time models to exploit statistical dependence in the joint distribution. However, the challenge is to fit these richer models within the tight memory and energy budgets of battery-free nodes and to support any increased communication. A possible solution is to train these models offline, using traces obtained from our testbeds, with further on-device fine-tuning at runtime.

5 Embracing Errors in TinyML: From Earth to Space

The inherent robustness of DNNs to noisy or incorrect data is vastly overlooked, especially when running intermittently. Literature [2] does exist that studies hardware faults in DNN execution using mainstream hardware architectures. The key takeaway is that the outcome of DNN processing is highly robust to data errors. This

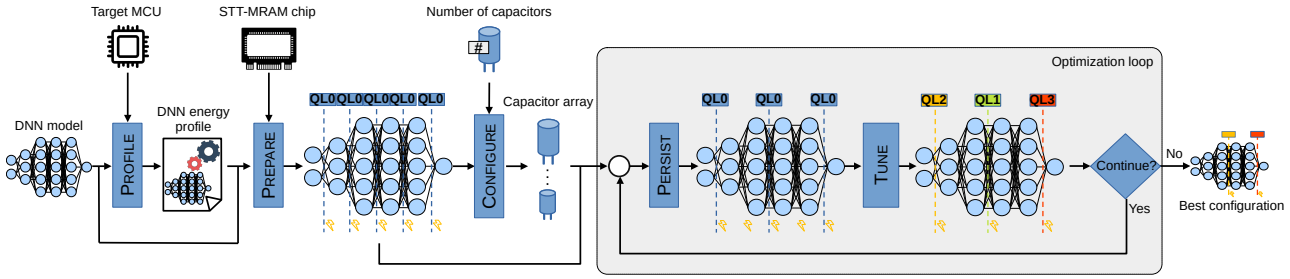

Figure 5: Overview of INTERCEPT.

capability is exploited in systems that process data in a slightly inexact way, reducing resource consumption at the cost of accuracy losses [8]. The key question is how this may possibly play together with intermittent executions on resource-constrained devices. Combine this with the increasing availability of low-power NVM technology other than FRAM, which is equally snubbed.

Our research work of the last few years exploits these observations as a stepping stone to save resources or enable unexplored deployment scenarios. We discuss our journey through new memory technologies and their trade-offs, as well as our recent efforts at deploying intermittent TinyML workloads in outer space.

5.1 We Are Not Married with FRAM

FRAM eventually replaced Flash memories in intermittent systems because of lower energy figures, more flexible operation, and the availability of MCUs with built-in FRAM. Compared with Flash, however, it is generally limited in size, forcing programmers to shape the application logic around memory limitations [38].

Emerging nonvolatile memory technologies include STT-MRAM and ReRAM. Both provide larger storage space compared to FRAM and comparable energy consumption. Their construction process, however, prevents straight integration into MCUs, requiring off-chip interactions usually through SPI or I2C. STT-MRAM exposes a unique knob: 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 [17]. This behavior is due to *stochastic switching*: depending on the current setting, a memory cell may fail to commute. These errors are stochastic in nature, thus they appear randomly in written data.

5.2 Saving Resources Thanks to Errors

Using STT-MRAM, it turns out can save resources when running intermittent DNN workloads without requiring changes to existing models. This intuition is made concrete with INTERCEPT (*INTERmittent inferenCE – Persist & Tune*) [11]: a compile-time toolchain that provides support for intermittent inference.

INTERCEPT functioning is based on multiple stages, as shown in Figure 5. Given the DNN model, we first Profile its energy consumption using existing tools [3] 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. Using a multi-capacitor architecture [14, 61], based on the output of Prepare, a Configure stage determines the capacitor array that ensures eventual completion of the inference process.

Next, the Persist algorithm processes the initial configuration to determine an efficient placement of state persistence operations. 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. We embrace, rather than avoid, the write errors possibly occurring by carefully controlling the current setting to reduce energy consumption, subject to a *hard* constraint on accuracy losses.

The output of Tune may potentially change the energy patterns along the inference process. For example, state persistence operations that are energy-hungry before applying Tune may become energy-savvy, compared to other state persistence operations that may 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.

We evaluate INTERCEPT across three different platforms and six diverse neural networks, compared with the original unmodified DNN. We demonstrate that INTERCEPT provides from a maximum of 64.4% to a minimum of 21% energy gain, corresponding to a maximum (minimum) 2.98x (1.36x) throughput speedup, in exchange for a maximum 1% accuracy loss. The 1% bound is arguably immaterial for most applications and is usually "lost in noise" [45]. Additional details and performance insights are nonetheless available [11].

5.3 Breaking It Into Space

The ability to locally exercise ML models is an asset when communication to the back-end is plainly impossible or extreme bandwidth constraints exist. This is precisely the scenario emerging with the recent rise of COTS hardware deployments in space and particularly in low-earth orbit (LEO). These designs offer cheaper operation and more flexible planning and mission management compared to monolithic designs. On the other hand, space devices such as CubeSats are extremely resource-constrained and subject to erratic energy provisioning patterns [58], picturing a scenario akin to those outlined earlier. Crucially, communication to the Earth is sporadic, unreliable, and bandwidth-constrained. This trend poses the question as to whether it is possible to run intermittent TinyML workloads in space.

Albeit the use of COTS hardware reduces costs, it also exposes the device to the woes of outer space, including radiations that may cause hardware faults normally not happening on the Earth. Recent literature demonstrates that regular fault-tolerance mechanisms may not necessarily operate efficiently in this environment [58]. The key question is whether deploying NVM as support for state persistence operation in space operations is viable at all.

We answer this question *experimentally*, by building an experimental CubeSat we deploy at 732 km from the Earth. We launch the satellite on November, 4th 2024 using the Polar Satellite Launch

Vehicle of the Indian space research organization. It was expected to ensure roughly three months of operation. The initial estimates were far exceeded as the satellite remained operational until the end of February, 2025. The space vehicle, shown in Figure 6, is built based on the 1Unit CubeSat platform of EnduroSat [19]. The UHF Transceiver II module from EnduroSat provides downlink communications to the Earth, by relying on the SatNOGS [49] global network of satellite ground-stations.

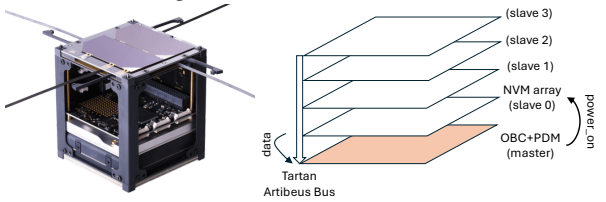

Figure 6: Experimental CubeSat platform.

The hardware aboard the satellite uses a layered master-slave design. The device at the bottom serves as the satellite's master On-board Computer (OBC). Its design is centered on a space-rated version of IBM's 6x86 CPU. Despite being an almost 30-year old design, its space-rated version is still deployed on space vehicles as space software written for it withstood extensive testing using formal methods and throughout multiple space missions [43, 52]. A radiation-strength aluminum shield separates each of the layers.

Besides logging of primary mission-related parameters and general bookkeeping, the software aboard the OBC controls a custom Power Distribution Module (PDM) integrated within the OBC board. The PDM uses the energy coming from four CTJ30 CESI Solar cells [18] to power the OBC. The OBC instructs the PDM to provide power to one or more of the four slave devices onboard. The OBC determines what device to power among the four slaves depending on their individual energy figures and the amount of experimental data output up to a given point, in an attempt to ensure fairness of energy allocations. The slave devices relay data to the OBC through a simplified version of the Tartan Artibeus bus [16].

One of the slave devices is a custom board equipped with a TI MSP430 MCU and three NVM chips connected through SPI: a Fujitsu FRAM MB85RS64V chip, a Fujitsu ReRAM MB85AS8MT chip, and an Everspin MRAM MR10Q010 chip. Each time the board is powered on, the MCU executes a predefined set of operations that deterministically produces a known bit sequence, which is eventually dumped on each of the NVM chips. The content of each memory dump is communicated to the OBC and later offloaded to Earth. Comparing the dumps from the CubeSat with the known bit sequences allowed us to spot faults in the NVM operation.

We are in the process of analyzing the data. Table 1 shows a sneak peek on the results. The table, together with additional data processing, allow us to draw a few early, yet crucial observations:

- There are no evident fault patterns: bit flips appear to be randomly distributed as "salt and pepper" [11].
- The FRAM chip is the *least robust* and the one showing highest variability in dependability performance.
- The MRAM chip is by far the *most robust* and also the one showing the least variability in dependability performance.
- The ReRAM chip stays somehow halfway between these extremes in absolute robustness and variability.

These observations should be weighted against other factors, including the storage space offered by a given chip and the energy figures. We can therefore draw one key preliminary conclusion: the "salt and pepper" pattern is no different compared to the errors occurring in STT-MRAM chips due to current scaling, therefore, techniques embracing these fault patterns, including INTERCEPT, *would perform equally well*, at least in principle, in outer space even though faults are not caused by current scaling.

Memory	Size	Mean flips	Std dev	Mean (%)
FRAM	65536	96.05	232.08	0.15%
MRAM	1048576	1.52	2.64	0,00014%
ReRAM	8388608	95.27	51.13	0,0011%

Table 1: Bit flips per memory type.

6 Concluding Remarks

This paper presents key technologies that address fundamental barriers to advancing intermittent systems for intelligent deep learning. As intermittent TinyML becomes increasingly ubiquitous, we envision a future with stretched application goals, featuring large-scale networks of interconnected battery-free devices operating reliably even in extreme environments, such as space or in-body implants, where battery replacement is infeasible. Our efforts have enabled sustainable sense-and-report type of workloads across many application domains, such as wildlife tracking and smart farming, where energy autonomy and successful task completion are paramount. However, a considerable gap remains before intermittent TinyML can achieve widespread adoption to realize its stretched application goals. To bridge this gap, the research agenda must extend beyond embedded software to a broader roadmap that spans hardware, networking, AI research, and their intersections. In particular, moving beyond off-the-shelf hardware toward emerging architectures is essential to boost performance for time-critical inference. Likewise, scalable custom networking protocols are crucial for instant reconnections and robust communication as battery-free networks scale and environments change rapidly. Future work also requires developing green AI models with inherently low carbon footprints to sustain billions of battery-free deployments. We therefore advocate powering a sustainable future for intelligence through intermittent TinyML, calling for contributions from diverse research fields and inviting the broader community to build upon our publicly available tools and methodologies.

Acknowledgement

This work was supported in part by the National Science and Technology Council, Taiwan, under Grant NSTC 113-2628-E-001-004-MY3, and by Academia Sinica under Grant AS-IA-113-M04-ASSA. It was also 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] K. S. Adu-Manu, N. Adam, C. Tapparello, H. Ayatollahi, et al. 2018. Energy-Harvesting Wireless Sensor Networks (EH-WSNs): A Review. *ACM TOSN* 14, 2, Article 10 (April 2018), 50 pages.

[2] M. H. Ahmadilivani, M. Taheri, J. Raik, M. Daneshtalab, et al. 2024. A Systematic Literature Review on Hardware Reliability Assessment Methods for Deep Neural Networks. *ACM CSUR* 56, 6 (2024), 1–39.

[3] S. Ahmed, A. Bakar, N. A. Bhatti, M. H. Alizai, et al. 2019. The betrayal of constant power×time: Finding the missing joules of transiently-powered computers. In *Proc. of ACM SIGPLAN/SIGBED LCTES*. 97–109.

[4] S. Ahmed, B. Islam, K. S. Yildirim, M. Zimmerling, et al. 2024. The Internet of Batteryless Things. *CACM* 67, 3 (Feb. 2024), 64–73.

[5] K. Akhunov and K. S. Yildirim. 2022. Adamica: Adaptive Multicore Intermittent Computing. *Proc. of the ACM IMWUT* 6, 3 (2022), 1–30.

[6] K. Akhunov and K. S. Yıldırım. 2023. CRAM-Based Acceleration for Intermittent Computing of Parallelizable Tasks. *IEEE TETC* 12, 1 (2023), 48–59.

[7] K. Akhunov, E. Yildiz, and K. S. Yildirim. 2023. Enabling Efficient Intermittent Computing on Brand New Microcontrollers via Tracking Programmable Voltage Thresholds. In *Proc. of ENSsys*. 16–22.

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

[9] S. Babatunde, A. Alsubhi, J. Hester, and J. Sorber. 2024. Greentooth: Robust and Energy Efficient Wireless Networking for Batteryless Devices. *ACM TOSN* 20, 3, Article 66 (April 2024), 31 pages.

[10] A. Bakar, R. Goel, J. De Winkel, J. Huang, et al. 2022. Protean: An Energy-Efficient and Heterogeneous Platform for Adaptive and Hardware-Accelerated Battery-Free Computing. In *Proc. of ACM SenSys*. 207–221.

[11] R. Barjami, A. Miele, and L. Mottola. 2024. Intermittent inference: Trading a 1% Accuracy Loss for a 1.9x Throughput Speedup. In *Proc. of ACM SenSys*. 647–660.

[12] P. Belcak and R. Wattenhofer. 2023. Fast Feedforward Networks. *arXiv preprint arXiv:2308.14711* (2023).

[13] L. Caronti, K. Akhunov, M. Nardello, K. S. Yıldırım, et al. 2023. Fine-grained Hardware Acceleration for Efficient Batteryless Intermittent Inference on the Edge. *ACM TECS* 22, 5, Article 82 (Sept. 2023), 19 pages.

[14] A. Colin, E. Ruppel, and B. Lucia. 2018. A Reconfigurable Energy Storage Architecture for Energy-harvesting Devices. In *Proc. of ACM ASPLOS*. 767–781.

[15] Leonardo Lucio Custode, Pietro Farina, Eren Yildiz, Renan Beran Kilic, et al. 2024. Fast-Inf: Ultra-Fast Embedded Intelligence on the Batteryless Edge. In *Proc. of ACM SenSys*. 239–252.

[16] B. Denby et al. 2022. Tartan Artibeus: A Batteryless, Computational Satellite Research Platform. In *Small Sat. Conf.*

[17] T. Devolder, J. Hayakawa, K. Ito, H. Takahashi, et al. 2008. Single-Shot Time-Resolved Measurements of Nanosecond-Scale Spin-Transfer Induced Switching: Stochastic Versus Deterministic Aspects. *PRL* 100 (2008), 057206. Issue 5.

[18] EnduroSat. 2025. 2 CESI Solar Cells CTJ30. https://satsearch.co/products/endurosat-1u-cubesat-solar-panel.

[19] EnduroSat. 2025. EnduroSat 1U CubeSat Platform. https://www.endurosat.com.

[20] Pietro Farina, Subrata Biswas, Eren Yıldız, Khakim Akhunov, et al. 2024. Memory-efficient Energy-adaptive Inference of Pre-Trained Models on Batteryless Embedded Systems. In *Proc. of EWSN*. 1–12.

[21] K. Geissdoerfer, M. Chwalisz, and M. Zimmerling. 2019. Shepherd: A Portable Testbed for the Batteryless IoT. In *Proc. of ACM SenSys*. 83–95.

[22] K. Geissdoerfer, I. Splitt, M. Sokolowski, C. Herrmann, et al. 2025. Shepherd Nova: A Public Testbed for Rigorous Experiments Under Repeatable Energy-Harvesting Conditions. In *Proc. of MobiSys*. 1–13.

[23] K. Geissdoerfer and M. Zimmerling. 2021. Bootstrapping Battery-free Wireless Networks: Efficient Neighbor Discovery and Synchronization in the Face of Intermittency. In *Proc. of USENIX NSDI*. 439–455.

[24] K. Geissdoerfer and M. Zimmerling. 2022. Learning to Communicate Effectively Between Battery-free Devices. In *Proc. of USENIX NSDI*. 419–435.

[25] K. Geissdoerfer and M. Zimmerling. 2024. Riotee: An Open-source Hardware and Software Platform for the Battery-free Internet of Things. In *Proc. of ACM SenSys*. 198–210.

[26] G. Gobieski, S. Ghosh, M. Heule, T. Mowry, et al. 2022. RipTide: A Programmable, Energy-Minimal Dataflow Compiler and Architecture. In *Proc. of IEEE/ACM MICRO*. 546–564.

[27] G. Gobieski, B. Lucia, and N. Beckmann. 2019. Intelligence Beyond the Edge: Inference on Intermittent Embedded Systems. In *Proc. of ACM ASPLOS*. 199–213.

[28] G. Gobieski, A. Nagi, N. Serafin, M. M. Isgenc, et al. 2019. MANIC: A Vector-Dataflow Architecture for Ultra-Low-Power Embedded Systems. In *Proc. of IEEE/ACM MICRO*. 670–684.

[29] Y. Han, G. Huang, S. Song, L. Yang, et al. 2021. Dynamic Neural Networks: A Survey. *IEEE PAMI* 44, 11 (2021), 7436–7456.

[30] S. Islam, J. Deng, S. Zhou, C. Pan, et al. 2022. Enabling Fast Deep Learning on Tiny Energy-harvesting IoT Devices. In *Proc. of IEEE/ACM DATE*. 921–926.

[31] Weiwen Jiang, Xinyi Zhang, Edwin H-M Sha, Lei Yang, et al. 2019. Accuracy vs. Efficiency: Achieving Both through FPGA-Implementation Aware Neural Architecture Search. In *Proc. of IEEE/ACM DAC*. 1–6.

[32] C.-K. Kang, H. R. Mendis, C.-H. Lin, M.-S. Chen, et al. 2022. More is Less: Model Augmentation for Intermittent Deep Inference. *ACM TECS* 21, 5, Article 49 (Oct. 2022), 26 pages.

[33] H. Kwon, P. Chatarasi, M. Pellauer, A. Parashar, et al. 2019. Understanding Reuse, Performance, and Hardware Cost of DNN Dataflow: A Data-Centric Approach. In *Proc. of IEEE/ACM MICRO*. 754–768.

[34] J. N. Laneman, D. N. C. Tse, and G. W. Wornell. 2004. Cooperative Diversity in Wireless Networks: Efficient Protocols and Outage Behavior. *IEEE Trans. Inf. Theory*. 50, 12 (2004), 3062–3080. doi:10.1109/TIT.2004.838089

[35] C.-C Lin, C.-Y. Liu, C.-H. Yen, T.-W. Kuo, et al. 2023. Intermittent-Aware Neural Network Pruning. In *Proc. of IEEE/ACM DAC*. 1–7.

[36] Ji Lin, Wei-Ming Chen, John Cohn, Chuang Gan, et al. 2020. MCUNet: Tiny Deep Learning on IoT Devices. In *Proc. of NeurIPS*. 11711–11722.

[37] V. Liu, A. Parks, V. Talla, S. Gollakota, et al. 2013. Ambient Backscatter: Wireless Communication out of Thin Air. In *Proc. of the ACM SIGCOMM*.

[38] A. Maioli and L. Mottola. 2021. Alfred: Virtual Memory for Intermittent Computing. In *Proc. of ACM SenSys*. 261–273.

[39] A. Y. Majid, M. Jansen, G. O. Delgado, K. S. Yildirim, et al. 2019. Multi-hop Backscatter Tag-to-Tag Networks. In *Proc. of IEEE INFOCOM*. 721 – 729.

[40] Maxim Integrated. 2021. MAX78000 Ultra-low-power MCU with Arm Cortex-M4 and a CNN Accelerator. https://datasheets.maximintegrated.com/en/ds/MAX78000.pdf.

[41] H. R. Mendis, C.-K. Kang, and P.-C. Hsiu. 2021. Intermittent-Aware Neural Architecture Search. *ACM TECS* 20, 5s (Sept. 2021), 64:1–27.

[42] H. R. Mendis, C.-H. Yen, C.-K. Kang, and P.-C. Hsiu. 2025. Intermittent-Friendly Neural Architecture Search: Demystifying Accuracy and Overhead Trade-offs. *IEEE TCAD* (March 2025), 1–14.

[43] L. Mottola et al. 2010. Anquiro: Enabling efficient static verification of sensor network software. In *Proc. of ICSE SESENA*.

[44] O. Mutlu, S. Ghose, J. Gómez-Luna, and R. Ausavarungnirun. 2022. A Modern Primer on Processing in Memory. In *Emerging computing: from devices to systems: looking beyond Moore and Von Neumann*. Springer, 171–243.

[45] H. Noh, T. You, J. Mun, and B. Han. 2017. Regularizing Deep Neural Networks by Noise: Its Interpretation and Optimization. In *Proc. of NIPS*. 5115–5124.

[46] Shvetank Prakash, Matthew Stewart, Colby Banbury, Mark Mazumder, et al. 2023. Is TinyML Sustainable? Assessing the Environmental Impacts of Machine Learning on Microcontrollers. *arXiv preprint arXiv:2301.11899* (2023).

[47] S. Resch, S. K. Khatamifard, Z. I. Chowdhury, M. Zabihi, et al. 2020. MOUSE: Inference In Non-volatile Memory for Energy Harvesting Applications. In *Proc. of IEEE/ACM MICRO*. 400–414.

[48] S. Resch, S. K. Khatamifard, Z. I. Chowdhury, M. Zabihi, et al. 2022. Energy-efficient and Reliable Inference in Nonvolatile Memory under Extreme Operating Conditions. *ACM TECS* 21, 5, Article 57 (Dec. 2022), 36 pages.

[49] SatNOGS. 2025. Open Source Global Network of Satellite Ground-stations. https://www.satnogs.org.

[50] W. Song, S. Kaxiras, L. Mottola, T. Voigt, et al. 2023. Silent Stores in the Battery-less Internet of Things: A Good Idea?. In *Proc. of EWSN*. 40–45.

[51] M. Sponner, B. Waschneck, and A. Kumar. 2024. Adapting Neural Networks at Runtime: Current Trends in At-Runtime Optimizations for Deep Learning. *ACM CSUR* 56, 10 (2024), 248:1–40.

[52] P. Stakem. 2004. Migration of an Image Classification Algorithm to an Onboard Computer for Downlink Data Reduction. *J. Aero. Comp. Info. Comm* (2004).

[53] Texas Instruments. 2016. MSP430™ Microcontrollers - Low-Energy Accelerator. https://www.ti.com/lit/an/slaa720/slaa720.pdf.

[54] C.-K. Kang, H. R. Mendis, C.-H. Lin, M.-S. Chen, et al. 2020. Everything Leaves Footprints: Hardware Accelerated Intermittent Deep Inference. *IEEE TCAD* 39, 11 (Nov. 2020), 3479–3491.

[55] C.-H. Yen, H. R. Mendis, T.-W. Kuo, and P.-C. Hsiu. 2025. Catch Non-determinism If You Can: Intermittent Inference of Dynamic Neural Networks. *ACM TECS* (2025), 1–20.

[56] C.-H. Yen, H. R. Mendis, T.-W. Kuo, and P.-C. Hsiu. 2023. Keep in Balance: Runtime-reconfigurable Intermittent Deep Inference. *ACM TECS* 22, 5s, Article 124 (Sept. 2023), 25 pages.

[57] T. Wang, K. Wang, H. Cai, J. Lin, et al. 2020. APQ: Joint Search for Network Architecture, Pruning and Quantization Policy. In *Proc. of IEEE/CVF CVPR*. 2078–2087.

[58] A. E. Yaacoub, T. Voigt, P. Ruemmer, and L. Mottola. 2025. Fault Tolerance in Space with Heterogeneous Hardware: Experiences from a 68-day CubeSat Deployment in LEO. In *Proc. of EWSN*.

[59] T.-J. Yang, Y.-H. Chen, and V. Sze. 2017. Designing Energy-Efficient Convolutional Neural Networks Using Energy-Aware Pruning. In *Proc of IEEE/CVF CVPR*. 6071–6079.

[60] C.-H. Yen, H. R. Mendis, T.-W. Kuo, and P.-C. Hsiu. 2022. Stateful Neural Networks for Intermittent Systems. *IEEE TCAD* 41, 11 (Nov. 2022), 4229–4240.

[61] K. S. Yıldırım, A. Y. Majid, D. Patoukas, K. Schaper, et al. 2018. Ink: Reactive kernel for tiny batteryless sensors. In *Proc. of ACM SenSys*. 41–53.

[62] E. Yıldız, L. Chen, and K. S. Yıldırım. 2022. Immortal Threads: Multithreaded Event-driven Intermittent Computing on Ultra-Low-Power Microcontrollers. In *Proc. of USENIX OSDI*. 339–355.

[63] M. Zabihi, Z. I. Chowdhury, Z. Zhao, U. R. Karpuzcu, et al. 2018. In-Memory Processing on the Spintronic CRAM: From Hardware Design to Application Mapping. *IEEE TC* 68, 8 (2018), 1159–1173.

[64] J. Zeng, J. Jeong, and C. Jung. 2023. Persistent Processor Architecture. In *Proc. of IEEE/ACM MICRO*. 1075–1091.