Protean: An Energy-Efficient and Heterogeneous Platform for Adaptive and Hardware-Accelerated Battery-free Computing

Abu Bakar[Ψ] Rishabh Goel [Ψ] Jasper de Winkel[★] Jason Huang[†] Saad Ahmed[Ψ] Bashima Islam[◇]
Przemysław Pawełczak[★] Kasım Sinan Yıldırım[‡] Josiah Hester[Ψ]
[Ψ]Georgia Institute of Technology [†]Northwestern University [★]Delft University of Technology [‡]University of Trento
[◇]Worcester Polytechnic Institute

ABSTRACT

Battery-free and intermittently powered devices offer long lifetimes and enable deployment in new applications and environments. Unfortunately, developing sophisticated inference-capable applications is still challenging due to the lack of platform support for more advanced (32-bit) microprocessors and specialized accelerators—which can execute data-intensive machine learning tasks, but add complexity across the stack when dealing with intermittent power. We present `Protean` to bridge the platform gap for inference-capable battery-free sensors. Designed for runtime scalability, meeting the dynamic range of energy harvesters with matching heterogeneous processing elements like neural network accelerators. We develop a modular "plug-and-play" hardware platform, `SuperSensor`, with a reconfigurable energy storage circuit that powers a 32-bit ARM-based microcontroller with a convolutional neural network accelerator. An adaptive task-based runtime system, `Chameleon`, provides intermittency-proof execution of machine learning tasks across heterogeneous processing elements. The runtime automatically scales and dispatches these tasks based on incoming energy, current state, and programmer annotations. A code generator, `Metamorph`, automates conversion of ML models to intermittent safe execution across heterogeneous compute elements. We evaluate `Protean` with audio and image workloads and demonstrate up to 666x improvement in inference energy efficiency by enabling usage of modern computational elements within intermittent computing. Further, `Protean` provides up to 166% higher throughput compared to non-adaptive baselines.

CCS CONCEPTS

• **Computer systems organization** → **Embedded hardware**; **Embedded software**; **Reconfigurable computing** .

KEYWORDS

Protean, Intermittent Computing, Energy Harvesting Platform

ACM Reference Format:
Abu Bakar, Rishabh Goel, Jasper de Winkel, Jason Huang, Saad Ahmed, Bashima Islam, Przemysław Pawełczak, Kasım Sinan Yıldırım, Josiah Hester. 2022. `Protean`: An Energy-Efficient and Heterogeneous Platform for Adaptive and Hardware-Accelerated Battery-free Computing. In *ACM Conference on Embedded Networked Sensor Systems (SenSys '22), November 6–9, 2022, Boston, MA, USA*. ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3560905.3568561

SenSys '22, November 6–9, 2022, Boston, MA, USA

ACM ISBN 978-1-4503-9886-2/22/11.
https://doi.org/10.1145/3560905.3568561

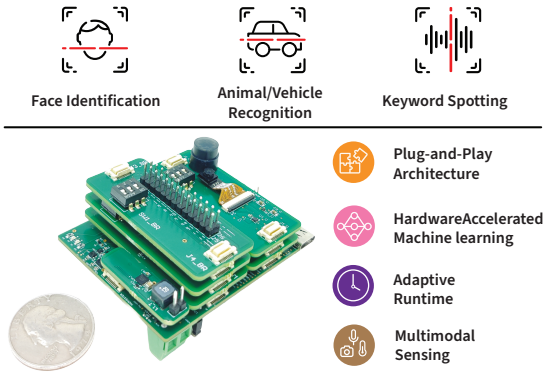

Figure 1: `SuperSensor`: a rapid prototyping system for battery-free, inference-focused applications. A thermal harvester board and five (stacked) peripheral boards are connected to the carrier board. All these boards are controlled and operated by an accelerator-based ARM MCU (placed on the back.)

1 INTRODUCTION

Today's smart devices have short battery lifetimes, high installation and maintenance costs, and rapid obsolescence—all leading to the explosion of electronic waste in the past two decades. These problems will worsen as the number of connected devices grows to one trillion by 2035 [6]. For at least the past decade, researchers have explored battery-less, energy-harvesting computing devices as a sustainable alternative. Ambient energy from sunlight, motion, thermal gradients, and even microbes [37, 63] is stored in capacitors to power computation, sensing, actuation, and communication.

These battery-less devices compute intermittently due to the dynamic and unpredictable nature of energy harvesting. Power failures can occur multiple times a second, whereupon volatile state (stack, registers, time) is lost. Recovering gracefully and efficiently from those interruptions has been the theme for a decade of intermittent computing research across hardware [14, 32, 79], software [34, 59, 77], wireless [25, 55] and tools [12, 23, 30]. These advances have yielded significant progress: batteryless devices have been shot into space [57], played Nintendo Game Boy games [16], been programmed in Python [49], Rust [83], and JavaScript [52], conducted deep inference [27, 43, 69], and simple vision tasks [18], and even safeguarded heritage sites [2].

However, expert programmers still find it challenging to quickly build useful things with these devices, while novices find them confounding. Furthermore, constrained and weak hardware makes machine learning or signal processing workloads challenging to execute. The most impressive demonstrations of intermittent computing mentioned above are all highly tuned, bespoke solutions that do not offer foundations for general approaches. The average battery-less devices are passive, low capability, unreliable, and less valuable

for applications where data-intensive operations and inference require reactive, interactive, or highly dynamic systems. To address this challenge, this paper describes an end-to-end inference-focused high-performance "plug-n-play" platform to *bring the average up* by tackling the following challenges, see Figure 1.

Low Performance and Scalability: Existing general-purpose platforms for research [14, 18, 32] rely on a decade-old family of Texas Instruments'16-bit MSP430 FRAM series MCUs [38], which has only 256 KB of FRAM, 8 KB of SRAM, a single-pipelined processor clocked at 16 MHz at maximum, and low energy accelerator (LEA) that supports energy-efficient vector-based signal processing. However, these MCUs are quite limited for executing data-intensive computational loads of contemporary machine learning applications. For instance, neural network inference on these MCUs for simplistic workloads like MNIST and keyword spotting [27] and JPEG compression [18] take multiple seconds of continuous computation. To reduce the computational demand of inference and its execution time to some extent, recent works [43, 69, 87] presented techniques for multi-exit inference and weight pruning. Several other studies [27, 47, 54] exploited LEA to accelerate inference, which can only support limited parallelism for a set of dense vector operations [27]. The field needs drastically expanded capability to support future applications and execute highly 'parallelizable' and data-intensive computational loads efficiently.

Lack of Cross-stack Adaptation and Heterogeneity: Energy harvesters have a wide dynamic power output range. For instance, a solar panel the size of a credit card can output from hundreds of microamperes in deep shade to a quarter of an ampere in bright sunlight—a difference of three to four orders of magnitude. This solar panel could power a streaming image recognition task using a neural network accelerator in bright sunlight; on the other hand, it could barely power motion detection on a 16-bit MCU in a deep shade. The time-varying power (or energy) creates a possibility to trade-off latency for performance (for example, accuracy) by adjusting operations across *heterogeneous* processing components of different efficiency and performance [17], to meet the growing demand for inference-focused battery-free computing applications.

Unfortunately, current batteryless sensing hardware platforms are monolithic [14, 15, 79, 92], making adaptation across heterogeneous hardware fitting a wide dynamic range impossible. Even "flexible" platforms like Flicker [32] and Capybara [14] only allow for flexible energy management and peripheral choice, limiting computational resources to a 16-bit microcontroller (TI MSP430 [38]), losing out on the potential performance increases of modern 32-bit ARM MCUs and accelerators. Recent trends in computer architecture have made this gap easier to fill; off-chip accelerators are now available off the shelf, providing for the first time a very dynamic range of computational resources for small devices, from FPGAs, vector processors, to CNN accelerators. Existing platforms miss out on high-performance possible from scaling tasks across *heterogeneous* computing modalities like accelerators and MCUs for energy-efficient inference-focused applications. We need platforms that support computing heterogeneity and efficient operation in the face of dynamic energy harvesting availability and power failures.

Losing to Obsolescence: Upgrading intermittent computing platforms to use more powerful processing elements (e.g., exploiting commercial off-the-shelf 32-bit ARM processors paired with on-board accelerators) keeps with traditions of the community to upgrade sensor platforms as new paradigms and capabilities emerge. However, supporting future upgrades is also crucial since the current computing landscape is exploding with heterogeneity—from low-power FPGAs, CGRAs, vector processors, and application-specific on-chip accelerators. Any "platform" will be immediately obsolete if it cannot be seamlessly upgraded with the new hardware.

To address above challenges, we present `Protean`, a multi-part framework for speeding up prototyping and deployment of robust, adaptive, and inference-driven battery-free, energy harvesting applications. `Protean` consists of three main sub-systems that encapsulate specific contributions.

(1) `SuperSensor`, a modular plug-and-play hardware design that supports multiple harvesters, sensors, and MCUs with the help of novel interconnects. It consists of an energy measurement unit and an adaptive reconfigurable energy storage unit that powers a carrier board, a *processor* module, and up to six *peripheral* modules. In this paper, we limited our efforts to one *processor* module based on an ARM Cortex MCU with a hardware accelerator, two harvester modules, and five *peripheral* modules consisting of six sensors and two communication modalities. Everything combined allows for rapid prototyping from very simple to deep-inference applications.
(2) `Chameleon`, an adaptive task-based runtime that maintains forward progress and memory consistency despite intermittent power. It dynamically switches between different performance tiers of an application (that may map across multiple computational elements, i.e., accelerator, MCU) based on predicted energy. `Chameleon` is the first general, adaptive runtime system for ARM-based MCUs that makes the best use of available energy by dynamically switching between multiple tiers of computation for a single application.
(3) `Metamorph`, a code generating tool that takes user input, and standard CNN (TinyML) models and outputs an intermittence-safe task-based implementation. It allows for quick experimentation of different configurations of computing elements, including accelerator and MCUs.

At the time of publication, we will release the `Protean` platform as an open-source, open hardware resource for the research community and will share a kit consisting of all hardware modules we developed with interested groups at the conference. We envision `Protean` as stimuli for batteryless sensing research that enables researchers from the broader systems community to easily build and test new batteryless applications.

2 BATTERYLESS PLATFORM (R)EVOLUTION

Battery-free embedded systems are enabling technologies for future applications in smart energy, transportation, environmental monitoring, wearables, smart cities, and beyond. They offer a more sustainable, long-term solution compared to traditional battery-powered sensors. Intermittent computing is the fundamental model of computation underlying battery-free embedded systems. Relying on volatile harvested energy makes computation, communication, and actuation very likely to be *intermittent*. Figure 2 shows this

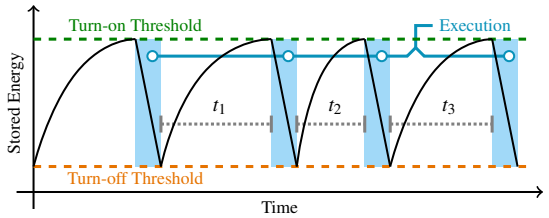

Figure 2: Battery-free energy-harvesting computers fail intermittently. Research problems come from preserving the state of execution across power failures.

intermittent operation where power failures intermingle with moments of operation, and the runtime must string together fragments of execution to meet application goals. Power failures become a frequent occurrence. Recovering gracefully and efficiently from those power failures has been the theme for intermittent computing research [33, 56] since 2011.

Intermittent computing has gone from a niche exploration in 2011 to an area with significant academic and industrial research efforts. With battery-free devices safeguarding national monuments [2], in space [57], and in homes [45], and in gaming consoles [16]. Industry efforts like Arm's Triffid project [70] are building low-power microprocessors meant for battery-free operation, noting that "The greatest challenge the Internet of things faces is how those 'things' will be powered." Nokia Bell Labs' has initiated efforts on wearable computing, angling towards battery-free wearables [69], and Communications of the ACM did multiple features on battery-free systems [48, 68, 71, 84], and most recently, a feature with an explicit focus on *intermittent computing* [80].

As intermittent computing has grown, so too has the maker movement. The maturity of the maker movement and stability of longtime hardware manufacturers AdaFruit and Sparkfun has led to standardization across hardware platforms in many ways, from communication protocols to interconnects (like Adafruit's Feather specification and Sparkfun's MicroMod platform), and sensor breakouts. Beyond hobbyists, students, and makers, experts in research labs and industry regularly rely on these vendors and their platforms to rapidly prototype high-performing inference-capable sensing applications. These community-supporting platforms with built-in modularity, and standardization, which include support for diverse sensors and computational resources, provide a blueprint for any platform for intermittent computing, as well as a backbone of components and tools for hardware prototyping.

2.1 *Intermittent Computing* Platform Progress

Intermittent computing platforms generally employ a non-volatile memory [77], typically a FRAM [14, 32] due to its higher endurance compared to FLASH and longer lifetime, that allows for fast checkpointing of the program state between power failures [3, 10]. Initial work in the space focused on software that tried to mitigate the shortcomings of intermittent operation either by instrumenting programs with checkpoints [46, 58, 77], or by rewriting applications using task-based programming models [34, 59, 89]. Hardware and platform approaches have focused on reducing the cost of checkpointing [36], managing energy more efficiently to reduce power failures, increasing event detection [14, 31, 32], and getting a rough estimation of the time elapsed between power failures [35, 76].

Energy management drove platform development, with Federated Energy [31] providing a way to partition energy per component, which reduces power failures due to error (i.e., the tragedy of the common energy store), and provides simpler scheduling around energy dynamics. Flicker [32] and Capybara [14] took this further with programmable energy storage, enabling dynamic hardware that could associate individual energy stores with computational and sensing tasks. Both tasks and energy stores could be fine-tuned and changed over time, providing great flexibility to the developer.

These general-purpose platforms provided crucial things: Flicker and Capybara provided a means of *scalability* where energy could be stored longer to be used for specific tasks. Both provided *programmability*, exposing hardware knobs to the programmer for increased control despite intermittent power. Flicker provided *modularity* by allowing a developer to mix-and-match sensors, peripherals, and harvesters, to compose an application. However, these platforms miss out on the high performance possible from scaling tasks across heterogeneous computing modalities like accelerators and MCUs for energy-efficient inference-focused applications.

2.2 Moving Forward

From this broader context, we distill four trends, or requirements (R) from recent work in intermittent computing, the maker movement, and the sensor networks community, as a way to guide platform development and sustain research.

R1: Inference. Machine learning in battery-free platforms relies upon application-tailored software support [27, 43, 69] or custom hardware [28]. These tailored solutions are not scalable as they cannot support heterogeneous hardware and diverse energy sources. Needed are mechanisms to scale inference ability based on energy.

R2: Energy-Aware Adaptation and Scalability All approaches around energy-aware intermittent-computing adaptation in existing literature use software-based techniques [43, 61] or heuristics [9, 69] to estimate energy availability due to the lack of hardware support. Predictions stemming from these approaches are prone to error and are not energy-efficient. Moreover, existing adaptive runtime systems only focus on adjusting compute complexity based on harvesting conditions and lack in changing the hardware configurations (for example, energy harvesting and storage circuity) to get better energy efficiency. Dynamically adjusting energy storage capacity based on workload/task size has been proposed [14], but still lack in energy-aware adaptation. Scaling architectures for intermittent computing have also been proposed [17], and are needed to support advanced and robust applications.

R3: Modularity. Modularity and expandability have been critical components of hardware platforms due to the high cost (in expertise, time, and money) of building hardware from scratch [21, 32]. In the past few years, major platforms have been introduced that embrace modularity and have broad maker industry support, including the Sparkfun MicroMod [82] platform, which allows modularity up to and including the processor itself (via the standardized on the M.2 interconnect). Our design is inspired by the MicroMod model. While we cannot predict the future, this level of community buy-in and existing infrastructure makes us stronger in our belief that we can build devices and frameworks that can hang around for a longer time, supporting a research community.

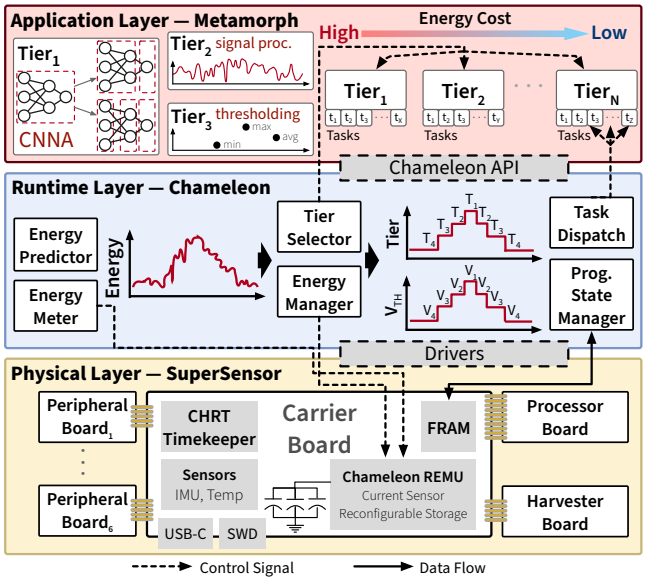

Figure 3: The three main components of Protean and how they interact to support adaptive accelerated applications.

R4: Programmability. Recent work has enabled intermittent computing of applications programmed with Python [49], Block-based languages [52], and Rust [83]. Tools for profiling [9], debugging [12], and simulation or emulation of energy harvesting [30] have been released to aid the developer. In this vein, future platforms, like ours, must strongly consider the developer at the other end, who often struggle to compose a batteryless program in order to ensure forward progress and memory consistency [9, 13, 51, 89]. Simplifying the workflow is key to an effective platform.

3 PROTEAN OVERVIEW

We design and build Protean around the four requirements established above, intended for developers who want to build inference-focused, adaptive, robust battery-free applications. Our goal is to (i) provide multiple hardware options in terms of computing modalities, peripherals, and harvesting technologies, (ii) enable energy-efficient inference applications, (iii) provide resilient runtime support for managing program state and memory across power failures, and (iv) allow rapid development and testing of different configurations of machine learning models.

We achieve these goals with a cross-stack approach (see Fig. 3), building (i) SuperSensor, a modular hardware platform inspired by Sparkfun's MicroMod [82] interconnect method, with a dynamically reconfigurable energy storage circuit, (ii) Chameleon, an adaptive task-based runtime system that provides intermittency-proof execution of adaptive machine learning tasks across heterogeneous processing elements (in our prototype, a 32-bit ARM core, and a CNN accelerator). The runtime dynamically dispatches these tasks based on incoming energy and program state, and arbitrates data movement for greater energy efficiency; and (iii) Metamorph, a code generator for transforming ML models developed in state-of-the-art frameworks (TensorFlow, PyTorch) into intermittence-safe C programs with little to no user intervention. The following sections present a detailed design of Protean's components (see Figure 4).

3.1 SuperSensor: Modular Platform Design

SuperSensor is a modular plug-and-play hardware design that supports four distinct modules—harvesters boards, sensors and radio peripheral boards, processors boards, and the carrier/main board. The boards, functions, and an example of how they work together are shown in Figure 4. The design is partly influenced by a leading maker company, Sparkfun, which has lent strong support in the past few years to the MicroMod ecosystem [82]. MicroMod separates dedicated processor boards (i.e., Teensy, Artemis, RPi ICs with minimal supporting circuitry), and carrier boards that host broadly application-specific functions (e.g., LCDs, environmental sensing). Much like MicroMod, SuperSensor is a solderless, modular interface ecosystem that uses the M.2 standard. SuperSensor separates carrier and processor boards, but we go a step further to allow more fine-grained control of peripherals (radios, sensors), which can be stacked on top of each other. Finally, we developed energy harvesting boards, similar to Flicker [32], that can be attached to any carrier board. We discuss specific functionalities per board below.

Carrier Board: This board is the brawn of SuperSensor. The intention of the carrier board is to fit all the absolute necessities for successful intermittent computing into one place, encompassing the lessons and designs of the last decade. Each of these functions must exist outside of the main processing unit. Those essential functions are: *checkpoint memory, energy management, timekeeping, debugging*, and *expansion interconnects*. Checkpoint memory (usually FRAM) is an external byte-addressable non-volatile memory for checkpoint storage between power failures. This is essential since very few MCUs beyond the MSP430FR series have this onboard. Energy management is essential to both extract the most energy from a harvester, as well as carefully distribute energy to needed peripherals, and at the right times. To allow for greater flexibility for developing adaptive runtimes, we include power measurement circuitry on the carrier board as part of the energy management unit. MCU-external timekeeping circuitry is needed since internal clocks lose state on power failure. Previous work has shown the criticality of timekeeping for maintaining data freshness, and scheduling tasks despite outages [34, 35, 43, 61]. We use the recently proposed Cascaded Hierarchical Remanence Timekeeper (CHRT) [15] that offers time tracking for long periods with high resolution. Finally, the carrier board includes dedicated interconnects for all other boards, peripherals, processors, and harvesters. A block diagram is shown at the bottom of Figure 3.

Processor Board: This is the brain of SuperSensor consisting of a microcontroller (MCU) and minimum supporting circuitry. By separating the processor and carrier boards, SuperSensor remains agile to new developments in MCUs, allowing for upgrades and alternate builds without significant disruption to the ecosystem. The processor board is programmed by the developer and hosts the runtime that maintains the forward progress and memory consistency of intermittently running applications. It manages peripheral control, energy, adaptation, etc. The processor board connects to the carrier board via a standard M.2 connector and is compatible with the Sparkfun's MicroMod processor board pinouts [82]. Thus any available Sparkfun MicroMod microcontroller board (Artemis, Teensy, RPi, ESP32) can be used with our carrier board, allowing

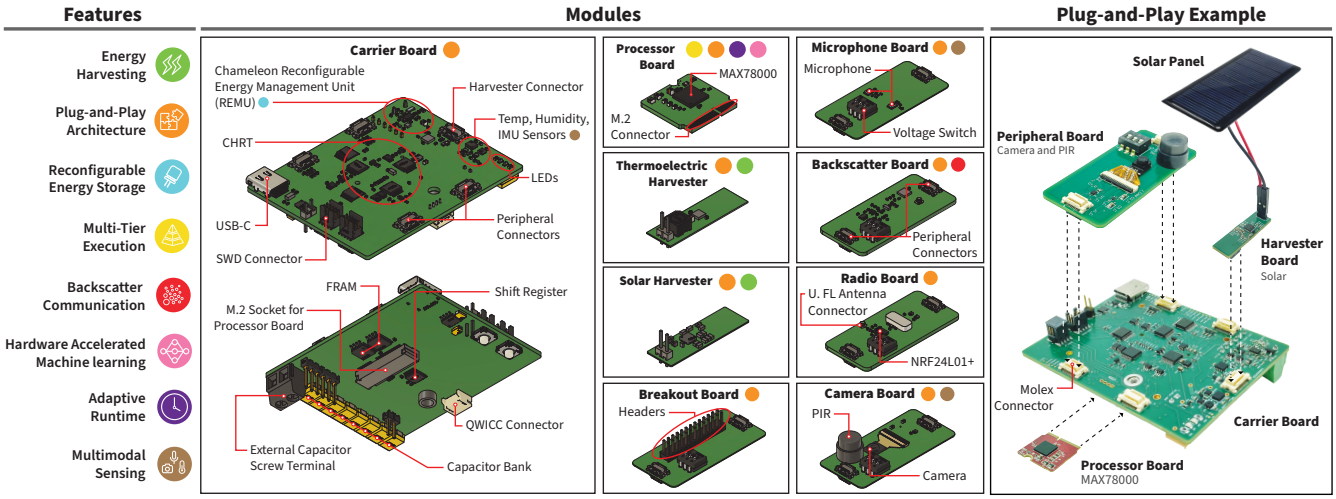

Figure 4: Overview of the capabilities, modules, and important features of the hardware/software platform. Interchangeable compute modules plug into the carrier board, and stacks of peripheral boards can be attached to a common bus to enable rapid prototyping for sophisticated batteryless devices.

for significant flexibility. In the future, as system-on-chips become more heterogeneous (e.g., the MAX78000 [19] which consists of an ARM Cortex M4 core, RISC-V core, and a CNN accelerator), these can also be supported.

Peripheral Boards: Peripheral boards connect to the carrier board to add functionality via sensors, actuators, radios, and other breakout modules. All peripherals boards use a common peripheral bus of our design that provides analog IOs, digital IOs, and digital bus lines, including QSPI, SPI, I2C, UART, I2S, and a parallel camera interface (PCIF). This shared bus supports the vast majority of available sensors and peripherals and enables `SuperSensor` to be used in various applications as almost all off-the-shelf sensing, and communication components use one of the interfaces our peripheral bus supports. These are comparable to "function" boards within the MicroMod system, providing a dedicated "function" for the entire build. Unlike MicroMod, we enable a through board stacking capability to not limit the number of peripherals that can be used.

Harvester Boards: These boards harvest energy from different environmental sources. Many energy sources exist—solar, kinetic, vibration, radio frequency (RF), thermal, and microbial—all of them provide energy differently. Some provide direct current (DC) while others generate alternating current (AC) and at a variety of different voltages and currents. Also, every harvester has different internal characteristics that require different circuitry to pull out maximum power in a particular context. The harvester boards are meant to support all these operations without requiring any change on the rest of the circuitry (*carrier*, *processor*, and *peripheral* boards.) These have no analog in the MicroMod platform, which was designed for tethered, or battery-powered operation.

3.2 Adaptive Reconfigurable Energy Storage

Within `SuperSensor`, energy management and storage must be paid special attention, just as in previous platforms for intermittent computing (which were nearly completely focused on this aspect [14, 29, 32]). The key focus of the energy management system (called the `Chameleon` Reconfigurable Energy Management Unit) is to scale, as in, to be dynamically configurable to provide energy for different tiers of execution, with tiers taking increasingly more energy (see top of Figure 3). The main challenge is in balancing capacitor size and program responsiveness.

Capacitor size in intermittently-powered systems is a trade-off between responsiveness and run-time energy requirements. A bigger capacitor will keep the device operating for longer, but will also take a long time to charge (and even longer when ambient energy is scarce). In a scenario with a single static capacitor with a constant energy input, the capacitor needs to be able to sustain the system for the longest uninterruptible system task. For smaller uninterruptible tasks, the system still must wait untill a full charge, introducing a penalty in the form of latency.

Existing reconfigurable energy storage options have tried to address this issue, which include a federated storage approach [31] where each peripheral has its own capacitor and is charged only when needed, or an array of capacitors that are switched in and out [14] based on application requirements. Both require multiple components for controlling the charge of each capacitor, are therefore costly, and result in energy fragmentation [91]. Another approach, Morphy [88], adjusts storage capacity programmatically but is not resilient to power failures. A combination of these approaches, malleable storage capacity, runtime control, and responsiveness, is needed to enable scalable machine learning applications.

`SuperSensor` addresses this need with a new kind of reconfigurable energy storage architecture built around a single supercapacitor and single control unit. The architecture dynamically adapts the charging threshold of a single supercapacitor, effectively modifying the amount of energy stored in the system. Lowering the threshold increases responsiveness, and increasing the threshold leads to a longer on-time. The threshold is set with a non-volatile potentiometer and a comparator via the runtime. When enough energy is harvested in the supercapacitor, the comparator triggers a latch turning the system on. With the system on, another comparator monitors the supercapacitor voltage, turning the latch off

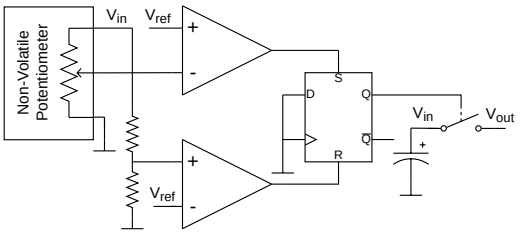

Figure 5: The architecture of adaptive reconfigurable energy storage that allows for modifications in the turn-on threshold of the system at run-time. Thresholds can be set dynamically for faster recovery from power failures in low-energy conditions.

when the voltage drops too low, and switching the system off (Figure 5). The non-volatile potentiometer retains the same resistor value unless changed by the runtime and is therefore agnostic to frequent power failure. This architecture greatly simplifies reconfigurable storage and could easily be integrated into custom silicon. We will discuss in later sections how this reconfigurability allows for runtime adaptation to achieve higher throughput.

3.3 Multi-tiered Tasks Runtime: Chameleon

SuperSensor provides a number of useful tools: energy monitoring and management, timekeeping, access to peripherals, and access to multiple computational elements. But hardware support alone cannot ensure the best configuration for the platform under variable energy conditions and unavoidable power failures of batteryless systems [7]. We developed Chameleon to leverage the capability of the hardware for scalable, inference-focused intermittent computing.

Runtime systems for intermittent computing checkpoint state before a power failure at compiler or programmer specified boundaries, and restore state after a power failure, resuming execution from the checkpoint. Task-based systems wrap program functions into atomic tasks and connect those tasks in a task graph. The state is saved at the task boundaries. Task-based intermittent computing runtime systems offer a high degree of programmability and low overhead [3, 9, 10, 13, 34, 60, 62, 77, 89] over automatic compilation approaches. Checkpointing adds overhead, thus reducing the energy available for program execution [10]. This translates into latency and missed events/deadlines. Tasks are rigid; if the task energy cost is higher than the energy ever harvested, the task will never execute. Suppose the frequency of power failures and the length of power outages is higher than usual. In that case, these systems fail to produce useful outputs within a reasonable time and waste vital energy on checkpointing and non-useful computations.

Adapting task energy cost increases availability and responsiveness. Recent work has adapted task-based execution by degrading tasks to meet deadlines [62], combining tasks when energy is available to save overhead [62], and using software techniques to scale machine learning based on energy trends [9, 43, 69].

However, none of these techniques scale tasks *across heterogeneous hardware*. Likely because these approaches used 16-bit platforms, which constrained the scope of contributions. Supporting adaptation across heterogeneous computing platforms is non-trivial, as the same task might be drastically different depending on where it is executed. Put simply, a signal processing routine on an

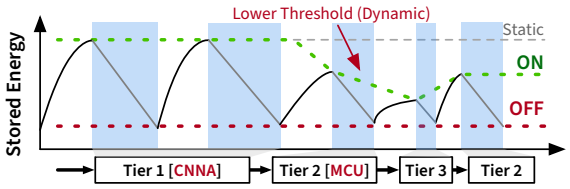

Figure 6: Illustration how Protean adaptively scales its execution across different hardware components. A simulated trace of the energy stored is shown. As energy harvesting slows down, Chameleon moves execution to a lower tier (from CNN accelerator to the MCU) to provide timely computation.

FPGA would be written in Verilog, a CNN would require trained weights, while an MCU would execute instructions. Furthermore, a CNN might host various implementations of the same routine that may trade off latency for performance. The central question is then: how can a programmer design and manage *task replicas* that can be dispatched by a runtime system to various computing elements, and at various quality, depending on available energy?

The core idea of Chameleon is to embrace scalability in hardware as well as software, providing a seamless way to degrade or upgrade tasks across diverse computational units. The basic idea is shown in Figure 6: as the rate of energy decreases (shown in the lower slope of the stored energy line), Chameleon changes the threshold for starting computation, and switches to a lower tier, which in this case means switching the main inference task from being hosted on the CNN accelerator, to host on the MCU (in a lighter form). Basically, a *tier* is a set of *tasks* that together form a control flow graph, whereas tasks are atomic code blocks that perform sensing, computations, communication, etc. [13, 34, 59, 89]. Chameleon allows the programmer to write multiple tiers of the same application with lower computational complexity—which can execute on different computational units (e.g., MCU, accelerators, or both)—that can help maintain the latency and deadlines requirements under changing energy conditions. Each tier is a complete application with potentially different approaches to solving the same inference problem (i.e., deep learning, signal processing) and is computationally independent of all other tiers. A lower tier must require less energy than a higher tier. Chameleon's scheduler (tier selector) can automatically adapt to the best tier under given energy conditions, as shown in Figure 3. Threshold and tier selection is assisted by an energy prediction model which leverages energy measurements and other heuristics for estimating current and future energy availability and choosing which tier to dispatch.

3.4 Metamorph: Intermittent-Safe CodeGen

Runtime systems cannot do everything, and developing multiple tiers of a single application can be challenging. Using standard tools for TinyML in an intermittent computing workflow becomes challenging as these tools have no conception of how to persist state across power failures. Metamorph is a developer-facing code generation tool, that bridges the gap between existing ML tools like PyTorch and TensorFlow [65, 74, 85], and intermittent computing. Further, Metamorph simplifies the development of multi-tier inference-focused applications for intermittently powered devices. We built Metamorph, as the glue holding the runtime system

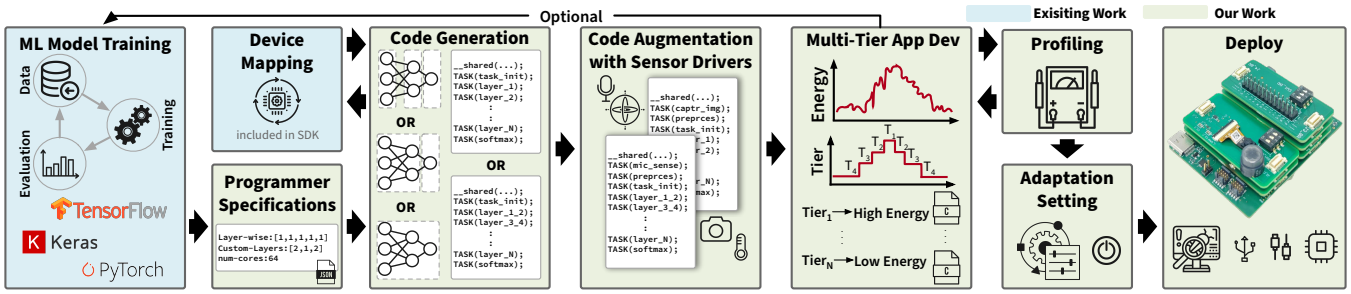

Figure 7: Typical workflow for programmers developing inference-focused applications with Protean. Models that are designed and trained using state-of-the-art frameworks are too computationally intensive to execute in one power cycle. Metamorph transforms large models into smaller chunks for execution across multi-tier compute elements with low programmer burden.

(Chameleon) and heterogeneous hardware platform (SuperSensor) together.

Metamorph wraps existing workflows to provide an automated way of generating intermittence-safe application code. Utilizing Metamorph, a developer can iteratively explore different design points of the application without dealing with underlying code generation frameworks or worrying about power failures. With Metamorph, the programmer only specifies different design points, i.e., the type of code to be generated as well as the number of layers (for CNNs) and tasks for the application. Based on these inputs, Metamorph automatically generates relevant code by invoking the right commands with correct compiler flag invocations. Then, Metamorph parses generated code to extract all functions from the code and classifies them into two categories. The first category of functions is specific to each task and has to be tailored according to the inputs and outputs of the task. In contrast, functions belonging to the second category must be performed only once during the application execution. These functions are called either at the start or end of the application. Metamorph first wraps all such functions with appropriate language constructs of Chameleon's runtime so that they are intermittence safe and identifies data across each task that needs to be saved across reboots. Variables are assigned to a special language construct exposed by Chameleon's runtime that allocates them in the non-volatile memory (FRAM) while ensuring a copy of these variables is also placed in the SRAM for faster access. These steps are necessary to ensure that the code generated by Metamorph allows safe resumption of code even after power failure and the least number of computations are lost.

4 IMPLEMENTATION

Herein we describe the particular design and implementation details to realize a full-featured, end-to-end platform. Importantly, we specify the workflow, shown in Figure 7, of a typical developer using Protean.

4.1 SuperSensor Hardware Platform

We proceed with the description of the hardware components of SuperSensor. Figure 4 shows individual modules currently built, and an example of how things connect.

Carrier Board: The board was designed for ease of use and extensive flexibility for benchtop and real-world experimentation. Sparkfun's MicroMod [82] M.2 connector is used to connect processor boards to the carrier. An array of eight tantalum capacitors (removable) provide energy storage, and a screw terminal allows energy storage expansion for radial capacitors. For the dynamic turn-on voltage of the system, a Maxim Integrated's non-volatile digital potentiometer [42] is used. The voltage threshold is set by changing the resistance of one of its resistors which is done by Chameleon via an SPI transaction. Two 1 MB FRAMs [40] are provided for checkpoint memory which potentially enough for the target applications. For each peripheral board, two 20-pin slim stack Molex connectors [67] are used. The carrier board has five such connectors, allowing for connecting six peripheral boards and one harvester board at a time. Carrier board also houses energy management and measurement circuit, Cascaded Hierarchical Remanence Timekeeper (CHRT) [15], commonly used IMU, temperature and humidity sensors, SWD debugging connectors, USB-C for serial messages and power while testing, LEDs and push-buttons. Eight load switches are used on the board to control power individually to six external peripheral boards and the onboard sensors. These load switches are controlled by a shift register which reduces the required GPIOs for power control from eight to three. Finally, a Qwiic connector [81] enables quick interfacing SuperSensor with off-the-shelf sensors and other peripherals.

Processor Board: For the first processor board, we chose a Maxim Integrated's MAX78000 MCU [19] as it is an advanced system-on-chip (SoC) featuring an Arm Cortex-M4, an ultra-low-power deep neural-network accelerator, and extensive memory resources. Its highly flexible accelerator architecture allows networks to be trained in conventional tools like PyTorch and TensorFlow. These networks can then be converted for execution on the MAX78000 using tools provided by Maxim. The processor board is attached to the carrier board via an M.2 connector that exposes power, UART, SPI, I2C, CAN, USB, AUDIO, SWD, analog IOs, and digital IOs to the rest of the components. This allows the processor to have control over everything in SuperSensor. For instance, REMU is controlled via SPI and an analog IO, CHRT via three analog and three digital IOs, FRAM via SPI, and peripheral boards via other interfaces as required. If an MCU does not have enough pins, it may still be used

for the processor board, but may not have access to all the features of SuperSensor.

Peripheral Board: Peripheral boards can be connected to the carrier board to add functionality to the device. We have built peripheral boards for a microphone, radio, backscatter, camera, and pin breakouts. The carrier board has two parallel peripheral board slots, and each peripheral board can stack other peripherals on top, up to a max of three boards in a stack using 20-pin slim stack Molex connectors [67]. This enables us to support six peripheral boards with a single carrier board. Each peripheral board slot has access to an independent I2C, SPI, and UART, which allows most peripherals to be used in either slot. Some interfaces are specific to the module slot: only one peripheral connector has access to the I2S audio interface, while the second peripheral has access to PCIF for a camera.

Harvester Board: We have developed a solar and a thermometric harvester using Texas Instrument's BQ25570 [41] maximum power point tracking chip, and Matrix Mercury [39] boost converter. Harvester boards sit on top of the 20-pin slim stack connector. Each harvester board has access to 2 GPIO pins for control over the harvester circuit. These pins can also be used for harvesters that communicate while providing power wirelessly (RF, NFC, etc.).

4.2 Chameleon Runtime

In this section, we describe implementation details of Chameleon to enable multi-tier task execution for intermittent computing.

Efficient State Retention: We implemented Chameleon on top of InK [89]. InK reads/writes data at each task boundary, which worked well for MSP430FR series MCUs that have on-chip (small) FRAM, but is not ideal for our larger, off-chip, and more data-intensive platform (i.e., more data over a slower bus). We address this in two ways. First, we employ a hybrid approach by maintaining two copies of the program state and memory, one in SRAM and another in FRAM, as memory is abundant compared to 16-bit platforms, but the bus between them is slow. All data accesses are done from the SRAM, which allows faster reads and writes. Whenever the program execution crosses a task boundary, data is *written* back to FRAM to ensure persistence in case a power failure occurs. Second, we skip copying data to the FRAM by specifying a threshold energy for the capacitor, i.e., a Just-In-Time (JIT) technique, used by many previous approaches [10, 62]. The threshold energy corresponds to the maximum energy required to save the state at each task boundary. At each task boundary, Chameleon either continues on without saving, avoiding redundant checkpoints (if enough energy), or saves everything onto FRAM to preserve progress.

Chameleon uses double buffers checkpoints similar to TICS [51], using two non-volatile memory buffers (active and scratch) to guard against data loss when power failure happens during checkpointing. Whenever there is a need to copy the data from SRAM to FRAM, it always writes data onto the scratch buffer and switches buffers only when the write is successfully completed.

Energy Prediction: The Chameleon Reconfigurable Energy Management Unit (Chameleon REMU) allows the system to measure harvested power at any instance of time. Chameleon periodically measures the harvested power P using the current sensor on the carrier board that sits between the harvester connector and the rest of the circuit of SuperSensor. It pushes measured power to a FIFO vector containing the last few samples and calculates a moving average P_{curr} of this vector. The predicted energy (E_{pred}) is then assigned to one of three values, HIGH, MEDIUM and LOW, calculated as

$$E_{\text{pred}} = \begin{cases} \texttt{HIGH}, & \text{if } P_{\text{curr}} \geq 0.8P_{\text{max}}, \\ \texttt{MEDIUM}, & \text{if } 0.6P_{\text{max}} < P_{\text{curr}} < 0.8P_{\text{max}}, \\ \texttt{LOW}, & \text{if } P_{\text{curr}} \leq 0.6P_{\text{max}}, \end{cases} \quad (1)$$

where P_{max} is the maximum harvestable power (which can be found in datasheet or via measurement.) From Equation (1), if the current harvested power is greater than 80% of the maximum harvestable power of a particular harvester, Chameleon considers/predicts it as HIGH energy, MEDIUM if between 60% and 80%, and LOW otherwise. Once a prediction is made, Chameleon sets a turn-on threshold of the SuperSensor based on this predicted energy availability. If harvested energy is HIGH, a higher threshold is set so that the system can accumulate more energy, thus allowing it to do more computations (Figure 6). These power windows and their corresponding voltage thresholds are configurable in Chameleon. In our implementation, we use 3.15 V for HIGH, 3 V for MEDIUM, and 2.85 V for LOW energy.

Multi-Tier Execution: Chameleon allows developers to switch dynamically among multiple tiers of an application. All tiers provide a level of application quality. When multiple tiers are defined by the user using the above described energy prediction approach, Chameleon dispatches the appropriate tier so that a tier with high energy requirement can be matched with HIGH energy availability. The tier-switching is done using the following adaptation logic.

$$T(t{+}1) = \begin{cases} T(t)\uparrow, & \text{if } E_{\text{pred}} \neq \texttt{LOW} \text{ and } T_{\text{min}} \leq T(t) \leq \frac{1}{3}T_{\text{max}}, \\ T(t)\downarrow, & \text{if } E_{\text{pred}} \neq \texttt{HIGH} \text{ and } \frac{2}{3}T_{\text{max}} \leq T(t) \leq T_{max}, \\ T(t), & \text{if } E_{\text{pred}} = \texttt{MEDIUM} \text{ and } \frac{1}{3}T_{\text{max}} < T(t) < \frac{2}{3}T_{\text{max}}, \end{cases} \quad (2)$$

where *T(t)* represents the current tier at time t, $\uparrow$ and $\downarrow$ show adapting up and down (i.e., moving towards a higher or a lower-tier, one tier at a time), respectively, and T_{max} and T_{min} correspond to the highest and lowest tier, respectively. The basic idea behind equation (2) is to upgrade or degrade application compute complexity depending on predicted energy and past trends by trying to map HIGH energy to top 33%, LOW energy to lowest 33%, and MEDIUM energy to middle 34% of the tiers.

Both dynamic turn-on thresholds, and multi-tier adaptive execution, can be independently enabled/disabled using a predefined macro in Chameleon's configuration file. If developers cannot implement lower tiers of the application, they can simply enable dynamic thresholding while keeping the multi-tier execution disabled, and Chameleon will still be able to adapt and provide higher overall energy efficiency with better response time, as we will see in Section 5.

4.3 Metamorph Code Generator

We combine insights on program transformation for intermittent safe execution along with industry-standard tools and workflows so that trained CNNs can be converted for execution on Protean

(specifically the MAX78000). Metamorph uses Maxim's SDK [65] for code generation in a three-stage pipeline—all implemented in Python, see Figure 7.

The first stage parses a JSON file that includes the programmer's specifications, including the number of accelerator cores to be used and the task division configuration, to know the type of code to be generated. Metamorph can generate three types of intermittence-safe code; vanilla, layer-wise, and with custom-layer division. In vanilla, all layers of the network are executed in one task, whereas the layer-wise has one layer per task. In custom-layers, Metamorph reads programmer desired number of layers per application task. Any of these configurations may be required depending on the application. For instance, if the application is to be deployed in a very low energy harvesting environment, the vanilla configuration might not be the best as it may never complete any inference due to continuous interruption by power outages. Based on the programmer's specifications, which are informed by application requirements, Metamorph passes the appropriate flags to the Maxim tools to generate the code for each application task.

In the second stage, Metamorph creates all tasks based on programmer specifications, using APIs and the CNN synthesis tool developed by Maxim [66]. Specifically, for a Tier_1 implementation, the stage configures input channels, loads weights and bias values for the task associated with each set of layers, and adds enable/disable code at the start/end of each task to make sure the accelerator is ON only when the task is performing the inference Afterward, Metamorph identifies the data shared between tasks and allocates global variables accordingly.

The third stage is responsible for making the program intermittent safe. It wraps the code into Chameleon's API to ensure that each layer's output is read and written correctly and persists across reboots. Checkpoint/restore logic operations are inserted at task boundaries. All global variables are mapped to a shared space using the __shared() API of Chameleon that remains persistent across reboots. Additionally, each task must use correct addresses to load the input and copy the output from/to the accelerator SRAM; otherwise, the output of the layer would be incorrect. To ensure this, Metamorph identifies addresses for each task to (un)load data from/to the accelerator's SRAM to/from MAX78000's SRAM to ensure correct resumption/restart of each task upon power failure and ensure energy-efficient execution of code.

5 EVALUATION

We test Protean components across a variety of applications, benchmarks, and energy conditions. We find that: Protean reduces energy per inference; tiered execution provides up to 8x more inferences in varied environments; and that we see up to 667x speedup on common ML benchmarks.

Applications/Benchmarks. We evaluate Protean using three different acoustic and vision applications/benchmarks, each having three tiers that utilize different processing units on the SuperSensor. Table 1 shows details of the different tiers of the applications along with the utilized processing unit. Tier_1 is a deep learning algorithm, while the other tiers are signal processing or classic machine learning algorithms. We use standard datasets to train the networks. For speech recognition, referred to as Keyword Spotting (KWS), we use a subset of Google's Speech Command dataset [86] where we have 20 different words, e.g., *Up*, *One*, *Right*, to detect. For image recognition, we used the CIFAR-10 dataset [53] consisting of ten classes of 32×32 pixel color images of automobiles and animals. For face identification, referred to as Face ID, we use MaximCeleb [64], a dataset created by Maxim containing the faces of 30 celebrities.

Table 1: Selected applications to evaluate Protean.

Application		Speech Recognition	Image Recognition	Face Identification
Tier_1	**Algo.**	Key Word Spotting 4× Conv1D 5× CONV2D 1× FC	Image Classification 6× Conv2D 5× FC	Face Recognition 8× Conv2D 1× FC
	Proc.	MCU + Accelerator	MCU + Accelerator	MCU + Accelerator
Tier_2	**Algo.**	Voice Detection VAD	Image Classification BNN Classifier 3x FC	Face Detection Binary Classifier
	Proc.	MCU	MCU	MCU
Tier_3	**Algo.**	Sound Detection Signal Variance	Object Detection Color Variance	Object Detection Color Variance
	Proc.	MCU	MCU	MCU

Performance Metric. We consider the following performance metrics for the evaluation.

- *Energy per Inference* is the energy required to finish processing inference of one data sample. This metric represents the efficiency of energy usage.
- *Recovery Time* is the time between two active/power-on cycles. Recovery time reflects the responsiveness of the system.
- *Throughput* is the number of inferences performed by the system. It shows the amount of computation that can be performed by the system in a certain duration.
- *Memory Profile* is the memory consumed by different tiers of the application.
- *Developer Effort* is a way of anticipating programmer's burden based on the lines of code they have to write.

Note that the chosen application algorithms are taken from existing literature as they only matter in providing computational complexity for evaluating Protean. Moreover, Protean's focus is only on the energy and latency of algorithm execution. In other words, the goal of Protean is to provide high throughput with lower recovery time and energy per inference. Therefore, the accuracy of the algorithms is out of the scope of this paper.

Energy conditions. In order to fully understand the benefits of different components of Protean, we conduct experiments in a controlled environment to test the resilience of Protean in extreme settings. For that, we synthesize three power traces and emulate them via Qoitech's Otii Arc [75]. These synthetic traces represent low (E_{low}), high (E_{high}), and variable (E_{variable}) energy conditions in the wild. We also record and emulate RF traces for the whole system evaluation in real-world settings. We use a 915 MHz Powercast transmitter TX91501B [73] with the Powercast P2110B controller [72] and record traces with a dipole and a patch antenna. While harvesting with the dipole antenna, the power is variable because of external factors like people walking between transmitter and receiver (RF_1), and with the patch antenna, the power is rather consistent with minor variability (RF_2). These traces are recorded and emulated via Ekho [30] so that every run of the experiment gets the same repeatable conditions.

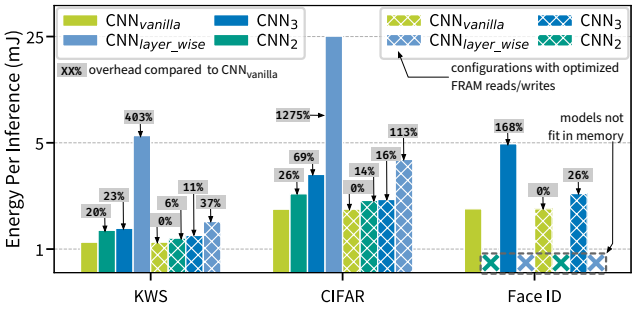

Figure 8: Energy per inference is shown for different task divisions. Smaller tasks always have higher overhead due to time spent on data movement from accelerator RAM to MCU RAM and then from MCU RAM to off-chip FRAM. We optimize our implementation to access FRAM only when the capacitor voltage falls below a certain threshold.

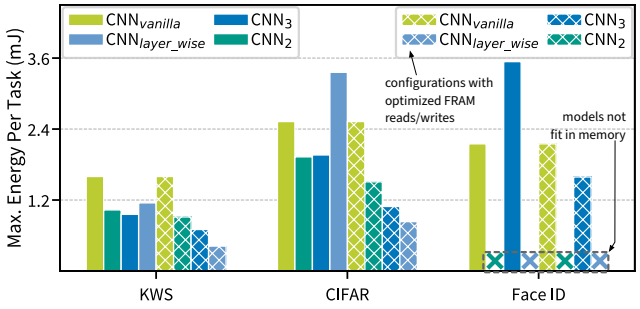

Figure 9: Configurations with more tasks have their workload divided into smaller chunks, reducing the maximum energy required to run each task. This translates to a smaller capacitor required to run the application, and in turn, a smaller device footprint with a faster recovery time in general.

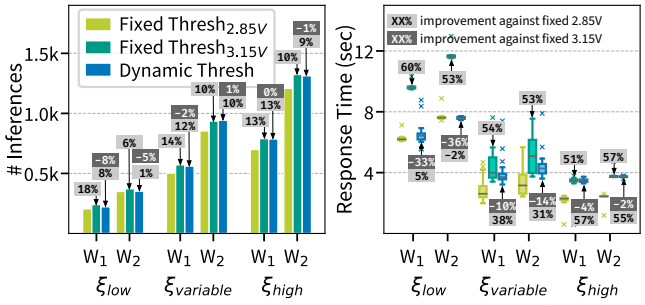

Figure 10: Lowering the threshold configuration (2.85 V) is more responsive but not energy-efficient (completes fewer inferences) due to program state backup/recovery overhead. Dynamic thresholding is as efficient as a higher threshold configuration (3.15 V) but with a much better response time.

5.1 Micro-Evaluation

Before diving deep into the end-to-end Protean evaluation, we first dissect the performance of individual components of the Protean system that are essential to the gains achieved overall. It will enable us to understand the reasoning behind why Protean is able to achieve better performance compared to other implementations of the application.

5.1.1 Energy per Inference. We start by evaluating our system on continuous power to measure the overhead incurred to ensure safe execution under the intermittent power supply. Figure 8 shows the results for energy-per-inference for different application configurations generated by Metamorph.

We can observe that the application configurations having more than one task (CNN_2, CNN_3, and CNN_{layer_wise}) incur a significant amount of energy overhead due to FRAM reads/writes at each task boundary. The overhead is directly proportional to the number of tasks in the application and the amount of data being read/written from/to the FRAM. We optimize our implementation to reduce this overhead by skipping FRAM reads/writes if the capacitor voltage is above a certain threshold, i.e., Chameleon will read/write to/from the FRAM only if a power failure is imminent. This allows for saving energy which is used to perform useful computations and to make more progress on the same charge. This optimization's outcome is also shown in Figure 8. As can be observed, it significantly reduces the overhead and makes the energy-per-inference comparable to the execution of the $CNN_{vanilla}$ (one task only), which is not intermittence-safe in extreme energy harvesting conditions. We can observe this behavior, especially, in CIFAR's CNN_{layer_wise} configuration. As the size of task-shared variables is bigger than other applications, CNN_{layer_wise} has a very high energy overhead compared with other configurations, and the same reasoning holds for all applications. It must be noted that for some configurations of the FaceID application, we could not run the application as it was impossible to load the code in memory as the size was too big.

Figure 9 shows the benefit of increasing the number of tasks per application. Increasing the number of tasks in an application decreases the workload per task, reducing the energy required to run the task and causing a reduction in the size of the capacitor needed for executing that workload, thus ensuring a faster recovery from power failures. It must be noted that if the system decides to save the data onto FRAM, our optimized version's energy consumption will still be similar to what we have for CNN_{layer_wise}. Existing literature has proposed a wide range of solutions for reducing the data saved at each task boundary [3, 13, 16, 59, 77]. Our system can benefit from any of the works to reduce the energy overhead even further. Proposing a new method to reduce the overhead is out of the scope of this paper.

5.1.2 Dynamic Thresholding. Protean is equipped with an adaptive reconfigurable energy management unit that allows Chameleon to turn the MCU ON at any voltage level. Here, we evaluate the performance of Protean's dynamic thresholding subsystem on two workloads. The first one (W_1) uses MCU and the accelerator, whereas the second (W_2) uses only the MCU. We report the number of inferences performed as well as response time for both workloads. Figure 10 shows the results for these two metrics under three different synthetic energy profiles and compares the result against a fixed threshold setting for 2.85 V and 3.15 V, respectively.

Seeing Figure 10 (left), we can observe that the number of inferences performed by fixed 3.15 V is always greater than that of fixed 2.85 V. It is primarily because at 3.15 V the capacitor can accumulate more energy compared to when it is allowed to charge until 2.85 V. Therefore, more energy allows the application to go farther on the same charge, and the application is able to perform more inferences compared to 2.85 V threshold. However, an increase in the number

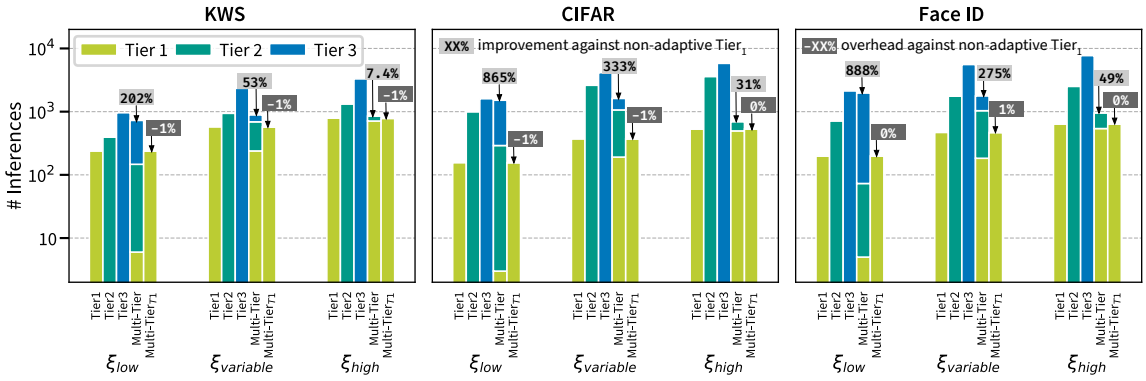

Figure 11: Tier$_1$'s execution is desired but not always possible due to varying energy harvesting conditions. Chameleon's multi-tier execution dynamically switches between application tiers based on predicted energy and completes more inferences than non-adaptive Tier$_1$ executions while incurring a very low overhead.

of inferences comes at the cost of increased response time. For fixed 3.15 V configuration, the capacitor needs to wait for a longer duration of time until voltage reaches 3.15 V before turning on the MCU (Figure 10 (right)). This is a problem as most intermittent computing applications have tasks with real-time deadlines/events or may want to run the system at a lower voltage for better energy consumption [4, 43, 61].

With Chameleon's dynamic voltage thresholding, Protean allows the system to dynamically adapt to the changing energy conditions, thus allowing it to start early when energy is low (i.e., faster recovery) and perform more inferences by charging the capacitor to a higher threshold when energy is high (i.e., higher throughput). Compared to the fixed 3.15 V configuration, the system performs a similar number of inferences and has a similar response time as fixed 2.85 V configuration. This can be validated from the result shown in Figure 10 (right). The response time of the Protean is better than the fixed thresholding method, especially when the incoming energy is low—a norm with batteryless systems.

5.1.3 Multi-Tier Execution. We now dissect the performance of Chameleon's multi-tier execution capability, which allows for running different versions of an application with varying computational complexity (application tiers) while maintaining desired throughput and responsiveness under variable energy supply. We compare the number of inferences performed by each tier of an application with a multi-tier approach and show that it performs better than running a single tier. Figure 11 shows the number of inferences performed by each application configuration on three synthetic power profiles.

Running Tier$_1$ most of the time is desired as it is more accurate. However, running this tier may not always be possible as the energy required to run Tier$_1$ is very high. Chameleon's multi-tier execution allows the system to automatically shift to a lower tier requiring smaller energy to perform the inference if the incoming energy is low and vice versa. It enables the system to perform more inferences than Tier$_1$ or Tier$_2$ would do when running alone. It can also be noted that the number of inferences for the Multi-Tier$_{T_1}$ approach is slightly smaller than that of Tier$_1$. This configuration is running three tiers, but all of them have the same Tier$_1$ program running. We do this specifically to measure the overhead of switching from one tier to the other, and as can be seen in Figure 11, this overhead is minimal as the number of inferences performed by the system is almost the same as the Tier$_1$ configuration.

Table 2: Comparison of energy usage of Protean.

Dataset	Protean		Existing SotA		Improvement
	#classes	(mJ)	#classes	(mJ)	
KWS	20	1.23	10 [69]	313	254.47×
CIFAR	10	2.12	2 [8]	17	8.02×
Face ID	30	2.32	-	-	-
MNIST	10	0.06	10 [27]	40, 27	666.7×, 450×

5.1.4 Comparison with State-of-the-Art. We compare our platform with state-of-the-art systems and show Protean performs better in terms of energy consumption when using the same benchmarks used by existing approaches. Table 2 lists the comparison.

We observe that Protean outperforms the existing state-of-the-art system by showing an improvement of 666× for MNIST, 82× for the KWS application, and the trend holds for CIFAR. It must be noted here that this is the least we can achieve with our platform as the number of classes supported by existing systems are very small. With the higher number of classes, the complexity of the machine learning model increases, and so does its computational complexity and energy consumption at run-time. Due to Protean enabling us to use accelerators on intermittent power, making a one-to-one comparison with existing systems will always favor Protean in terms of performance (by multiple orders of magnitude). In a sense, this is important, as Protean is the first to allow access to these powerful computational resources for intermittent computing. Then, due to our low energy consumption, we are able to run an emerging Face ID application that, to the best of our knowledge, has not been done before in the intermittent computing domain. With our platform design and novel interconnection of peripherals, memory, and MCU, we have significantly reduced the energy consumption of the applications.

5.2 Memory Overhead

Table 3 shows the memory overhead of the multi-tier approach used by Chameleon. Tier$_1$ is the default configuration for the runtime. Tier$_2$ and Tier$_3$ are the implementations that allow the degraded inference to ensure that the responsiveness and throughput requirements of the application are met. These tiers add a memory overhead of 38.7% in addition to Tier$_1$'s implementation for the

Table 3: Memory Overhead for multi-tier approach.

Application	Configuration	Memory Usage (KB)			
		.nvm	.bss	.text	.data
KWS	$Tier_1$	32.4	18.14	280.39	2.44
	$Tier_2$	16.16	9.36	77.84	2.44
	$Tier_3$	4.16	3.36	67.31	2.5
	Multi-tier	52.44	28.14	324.11	2.5
CIFAR	$Tier_1$	128.27	66.14	397.17	2.44
	$Tier_2$	2.66	2.6	56.25	21.01
	$Tier_3$	0.16	1.35	59.04	2.44
	Multi-tier	130.79	67.14	411.15	21.01
Face ID	$Tier_1$	85.53	44.64	321.18	2.44
	$Tier_2$	2.98	2.76	133.18	2.44
	$Tier_3$	2.98	2.76	130.22	2.44
	Multi-tier	98.19	47.64	483.16	2.44

Table 4: Size of application implementations (Lines of Code) in `Protean`.

Application	`Protean`				Vanilla
	$Tier_1$	$Tier_2$	$Tier_3$	Total	
KWS	11589	816	674	13079	8689
CIFAR	15925	840	200	16965	12365
Face ID	10562	3695	2489	16746	9781

KWS application. The memory overhead for CIFAR and FaceID applications is 2% and 6%, respectively. The memory overhead is overshadowed by the increase in the number of inferences and improved response time. Note that $Tier_2$'s and $Tier_3$'s implementation for each application is not `Protean` dependent and can be written by the programmer in any way required.

5.3 Developer Effort

Table 4 shows the number of lines of code a developer has to use with `Protean`. Vanilla is the version that is preferred by the programmer to run on the device. This version is mostly generated with the help of Maxim's SDK as it involves memory mapping, loading/unloading thousands of weights and bias values, etc.; however, it is unsafe to run on harvested energy. To convert it into a $Tier_1$'s implementation, the programmer needs to add almost ≈3000 lines of code. This is cumbersome and difficult for the programmer. `Metamorph` can automatically generate most code required for conversion with minimal support from the programmer. For $Tier_2$ and $Tier_3$, the developer only has to add 11% , 6%, and 37% additional lines of code for KWS, CIFAR, and FaceID applications, respectively, which again is application-specific.

5.4 End-to-end System Evaluation

We now evaluate our system with dynamic voltage thresholding configured and tier-switching ON. Figure 12 reflects the throughput (number of inferences) and response time of `Protean` for two different RF energy traces. We observe 12% to 166% improvement in inference throughput and up to 11% faster recovery from non-adaptive baselines across all applications. Face ID application achieves the maximum performance boost over the baseline for variable low energy trace, i.e., RF_1, as `SuperSensor` supports dynamic turn-on voltage based on incoming energy facilitating tier adaptation to

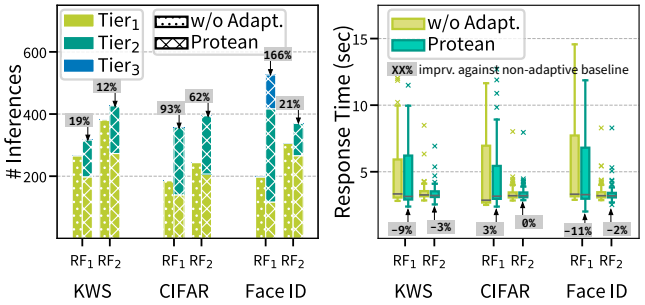

Figure 12: Protean has higher throughput with less recovery time when powered using real-world RF energy trace.

make the best use of the variable available energy. Higher variability in the energy results in more adaptation by `Chameleon` to ensure the system provides timely output. This end-to-end analysis shows the performance of `Protean` with real-world harvested energy, which further validates the results of our micro-evaluation in complex and real scenarios.

6 RELATED WORK

In this section, we summarize other related and complementary works beyond those described in the background (Section 2).

Energy Harvesting Sensing Platforms: Many general-purpose platforms have been proposed since the early days of wireless sensor networks [1, 14, 32, 44]. Some use MSP430FR series MCUs [14, 15, 32] while others do not [1, 21, 44, 79], depending whether the target is an intermittent computing application or just an energy-harvesting/energy-neutral deployment, respectively, as the former offer fast and efficient on-chip non-volatile FRAM, which is crucial for all intermittent computing systems. Other platforms are either tied to specific applications [22], a particular harvesting environment [79], or are inconvenient to support intensive intermittent machine learning inference while providing ease of development. Various vector-dataflow designs [28] and coarse-grained reconfigurable architectures [11, 26] have been proposed for the low-power acceleration at the edge. However, the feasibility of these solutions in a real-world scenario is yet to be tested. In a nutshell, no platform exists that allows the integration of new (future) MCUs to enable long-term real-world deployments required for testing the feasibility of new systems. `Protean`, in general, and `SuperSensor`, in particular, is build on these efforts to enable the rapid development of the next generation of compute-intensive intermittent inference applications via different energy sources, heterogeneous processing elements, and multiple peripherals.

Intermittent Computing: In software-based solutions to intermittent computing, programmers need to use checkpoints in their programs [3, 50, 51, 77] or write their code using task-based programming models [34, 59, 78, 89, 90]. Checkpoints and tasks ensure computation progress and memory consistency of programs despite frequent power outages. While these solutions made battery-free intermittent computing possible, they come with additional overhead (especially for memory-intensive operations [27]) and require code restructuring and porting for different energy environments [20]. Some studies focused on reducing the intermittent computing overhead and power failures with efficient checkpointing [3, 51] and

efficient energy management [10, 14, 88]. Protean is complementary to these efforts, and as these designs become more widely available, they can easily be integrated into the Protean ecosystem, with minor modifications in Chameleon.

Adaptation in Intermittent Computing: Adapting application execution based on the incoming energy is crucial to maximizing application throughput in a battery-free system. Protean is not the first to explore adaptation for intermittent computing. Zygarde [43], Catnap [61], Eperceptive [69], and REHASH [9] all develop energy-aware task schedulers, many for machine learning tasks. Zygarde [43] is an energy-aware task scheduler that uses a model of the harvested energy to schedule tasks. However, its adaptiveness focuses on executing different depths of a neural network and is not generalized for all applications. Catnap [61] isolates energy for real-time tasks and uses the remaining energy to run other tasks. Morphy [88] employs adaptive energy management algorithms for fast charging to an operational voltage to maximize active time. Eperceptive [69] performs the best effort solution where multi-resolution inputs are used based on different energy availability. REHASH [9] performs adaptation based on software-based heuristics to enable higher sensor coverage, completion rates, or throughput, depending on the application. Adamica [5] dynamically reconfigures multicore architecture to use the power most optimally. Contrary to these systems, Protean is the first to explore adaptive execution across heterogeneous system components.

7 DISCUSSION AND FUTURE WORK

The work in this paper is a start towards more capable applications with energy harvesting (and usually battery-free) devices. We anticipate the platform serving as a jumping-off point for many lines of research. In this section, we discuss current limitations of Protean, and describe some potential avenues for future research.

Intra-tier Adaptation. Chameleon focuses on increasing throughput and responsiveness by switching between tiers of different accuracy, i.e., (inter-tier adaptation) based on the available energy. We did not consider intra-tier adaptation by trading-off accuracy within a tier. In the future, we plan to design a scheduler that optimizes the throughput, responsiveness, and accuracy by employing a hybrid scheduling algorithm.

Multicore Adaptation. Multi-tier execution might not be the only method for adapting compute complexity. As in Adamica [5], we tried multicore adaptation to scale the number of active cores to increase/decrease the parallelism and energy consumption. However, we did not see significant improvements with multicore adaptation on MAX78000. The latency increased by decreasing the number of cores while power decreased, and vice versa, but the energy per inference remained more or less the same. Nevertheless, Metamorph can automatically generate multiple tiers of the same model for future MCUs offering multicore adaptation benefits. This feature would reduce the programmers' burden as they will not need to develop low-energy tiers manually for a particular application.

Diving Deep into Heterogeneity. Besides MAX78000, which has an on-chip accelerator but does not have embedded non-volatile memory, Ambiq's Apollo 4 MCU is also a good fit for SuperSensor since it has an on-chip non-volatile memory. SuperSensor can also be extended by utilizing an ultra-low-power FPGA as an adaptive and flexible accelerator. In the future, we would like to bring more hardware heterogeneity to SuperSensor to boost computational power that will enable more applications on the battery-free edge.

Energy Prediction. Chameleon's energy monitoring support paves the path for developing sophisticated energy prediction algorithms. Most works in the literature can perform long-term energy predictions (e.g., from hours to weeks [24]). However, in many scenarios, e.g., kinetic energy harvesting while walking, energy distribution changes much more rapidly. Zygarde [43] proposed a probability metric (η-factor) for predicting energy availability, but it requires collecting large amounts of data offline. Though we opt for a simple online moving average based energy prediction approach to show the effectiveness of SuperSensor and Chameleon in regards to adaptation, future work on Protean will focus on more advanced short-term energy prediction algorithms (having more time granularity) that does not require large amounts of data collection.

Code Generator Improvements. Now that the baseline code generating tool is implemented in the Protean ecosystem in the form of Metamorph, support of more architectures can be imagined. Though Metamorph's current implementation is based on Maxim's SDK, the three-stage pipeline will remain unchanged even if any other SoC/architecture is used. We anticipate that with the open-source release, we can partner with the community to improve the integration of other SoCs/SDKs, and enable future research.

8 CONCLUSION

This paper presents Protean, a unified framework for robust and adaptive execution of inference-driven applications on a battery-free platform. Protean contributes to the entire development stack with SuperSensor, a modular plug-and-play hardware; Chameleon, an adaptive task-based runtime system for heterogeneous intermittent systems; and Metamorph, a code generator to convert traditional CNN models to intermittent-safe task-based CNN models. SuperSensor pushes the boundary of battery-free computing systems by supporting more efficient ARM-based MCUs and CNN accelerators that achieve 666× more energy-efficient inferences than traditional battery-free platforms. Chameleon reduces the recovery time from power failure by 11% while achieving 166% higher throughput than non-adaptive counterparts in real-world settings. Finally, Metamorph speeds up the development process while reducing programmer burden.

ACKNOWLEDGMENTS

We would like to thank Brian Rush and the MAX78000 team at Analog Devices for pointers on working with the accelerator chip and SDK, we thank Tommy Cohen for assistance with all graphics. This research is based upon work supported by the National Science Foundation (NSF) under award numbers, CNS-2145584, CNS-2038853, and CNS-2030251. It was also supported by the Netherlands Organisation for Scientific Research (NWO), and partly funded by the Dutch Ministry of Economic Affairs (EZK), through TTW Perspective program ZERO (P15-06) within Project P1. Any opinions, findings, conclusions, or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the NSF, NWO, or EZK.

REFERENCES

[1] Joshua Adkins, Branden Ghena, Neal Jackson, Pat Pannuto, Samuel Rohrer, Bradford Campbell, and Prabal Dutta. 2018. The signpost platform for city-scale sensing. In *2018 17th ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN)*. IEEE, 188–199.

[2] Mikhail Afanasov, Naveed Anwar Bhatti, Dennis Campagna, Giacomo Caslini, Fabio Massimo Centonze, Koustabh Dolui, Andrea Maioli, Erica Barone, Muhammad Hamad Alizai, Junaid Haroon Siddiqui, and Luca Mottola. 2020. Battery-Less Zero-Maintenance Embedded Sensing at the MithræUm of Circus Maximus. In *Proceedings of the 18th Conference on Embedded Networked Sensor Systems* (Virtual Event, Japan) *(SenSys '20)*. Association for Computing Machinery, New York, NY, USA, 368–381. https://doi.org/10.1145/3384419.3430722

[3] Saad Ahmed, Naveed Anwar Bhatti, Muhammad Hamad Alizai, Junaid Haroon Siddiqui, and Luca Mottola. 2019. Efficient intermittent computing with differential checkpointing. In *Proceedings of the 20th ACM SIGPLAN/SIGBED International Conference on Languages, Compilers, and Tools for Embedded Systems*. 70–81.

[4] Saad Ahmed, Qurat Ul Ain, Junaid Haroon Siddiqui, Luca Mottola, and Muhammad Hamad Alizai. 2020. Intermittent Computing with Dynamic Voltage and Frequency Scaling.. In *EWSN*. 97–107.

[5] Khakim Akhunov and Kasim Sinan Yildirim. 2022. AdaMICA: Adaptive Multicore Intermittent Computing. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies* 6, 3 (2022), 1–30.

[6] ARM. 2019. An update on Arm's AI journey toward a trillion connected devices. Retrieved May 1, 2022 from https://www.arm.com/company/news/2019/09/an-update-on-arm-s-ai-journey-toward-a-trillion-connected-devices

[7] Abu Bakar and Josiah Hester. 2018. Making sense of intermittent energy harvesting. In *Proceedings of the 6th International Workshop on Energy Harvesting & Energy-Neutral Sensing Systems*. 32–37.

[8] Abu Bakar, Tousif Rahman, Alessandro Montanari, Jie Lei, Rishad Shafik, and Fahim Kawsar. 2022. Logic-based intelligence for batteryless sensors. In *Proceedings of the 23rd Annual International Workshop on Mobile Computing Systems and Applications*. 22–28.

[9] Abu Bakar, Alexander G Ross, Kasim Sinan Yildirim, and Josiah Hester. 2021. REHASH: A Flexible, Developer Focused, Heuristic Adaptation Platform for Intermittently Powered Computing. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies* 5, 3 (2021), 1–42.

[10] Domenico Balsamo, Alex S Weddell, Geoff V Merrett, Bashir M Al-Hashimi, Davide Brunelli, and Luca Benini. 2014. Hibernus: Sustaining computation during intermittent supply for energy-harvesting systems. *IEEE Embedded Systems Letters* 7, 1 (2014), 15–18.

[11] Thilini Kaushalya Bandara, Dhananjaya Wijerathne, Tulika Mitra, and Li-Shiuan Peh. 2022. REVAMP: a systematic framework for heterogeneous CGRA realization. In *Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*. 918–932.

[12] Alexei Colin, Graham Harvey, Brandon Lucia, and Alanson Sample. 2016. An Energy-interference-free Hardware/Software Debugger for Intermittent Energy-harvesting Systems. In *Proc. ASPLOS* (April 2–6). ACM, Atlanta, GA, USA, 577–589.

[13] Alexei Colin and Brandon Lucia. 2016. Chain: tasks and channels for reliable intermittent programs. In *Proceedings of the 2016 ACM SIGPLAN International Conference on Object-Oriented Programming, Systems, Languages, and Applications*. 514–530.

[14] Alexei Colin, Emily Ruppel, and Brandon Lucia. 2018. A reconfigurable energy storage architecture for energy-harvesting devices. In *Proceedings of the Twenty-Third International Conference on Architectural Support for Programming Languages and Operating Systems*. 767–781.

[15] Jasper de Winkel, Carlo Delle Donne, Kasim Sinan Yildirim, Przemysław Pawełczak, and Josiah Hester. 2020. Reliable timekeeping for intermittent computing. In *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*. 53–67.

[16] Jasper de Winkel, Vito Kortbeek, Josiah Hester, and Przemyslaw Pawelczak. 2020. Battery-free Game Boy. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies* 4, 111 (2020), 1–26.

[17] Harsh Desai and Brandon Lucia. 2020. A power-aware heterogeneous architecture scaling model for energy-harvesting computers. *IEEE Computer Architecture Letters* 19, 1 (2020), 68–71.

[18] Harsh Desai, Matteo Nardello, Davide Brunelli, and Brandon Lucia. 2022. Camaroptera: A Long-Range Image Sensor with Local Inference for Remote Sensing Applications. *ACM Trans. Embed. Comput. Syst.* (jan 2022).

[19] Analog Devices. 2020. Artificial Intelligence Microcontroller with Ultra-Low-Power Convolutional Neural Network Accelerator. Retrieved June 15, 2022 from https://datasheets.maximintegrated.com/en/ds/MAX78000.pdf

[20] Çağlar Durmaz, Kasım Sinan Yıldırım, and Geylani Kardas. 2022. Virtualizing Intermittent Computing. *IEEE Internet of Things Journal* (2022).

[21] Prabal Dutta, Jay Taneja, Jaein Jeong, Xiaofan Jiang, and David Culler. 2008. A building block approach to sensornet systems. In *Proceedings of the 6th ACM conference on Embedded network sensor systems*. 267–280.

[22] Francesco Fraternali, Bharathan Balaji, Yuvraj Agarwal, Luca Benini, and Rajesh Gupta. 2018. Pible: battery-free mote for perpetual indoor BLE applications. In *Proceedings of the 5th Conference on Systems for Built Environments*. 168–171.

[23] Kai Geissdoerfer, Mikołaj Chwalisz, and Marco Zimmerling. 2019. Shepherd: A portable testbed for the batteryless iot. In *Proceedings of the 17th Conference on Embedded Networked Sensor Systems*. 83–95.

[24] Kai Geissdoerfer, Raja Jurdak, Brano Kusy, and Marco Zimmerling. 2019. Getting more out of energy-harvesting systems: Energy management under time-varying utility with preact. In *Proceedings of the 18th International Conference on Information Processing in Sensor Networks*. 109–120.

[25] Kai Geissdoerfer and Marco Zimmerling. 2021. Bootstrapping battery-free wireless networks: Efficient neighbor discovery and synchronization in the face of intermittency. In *18th USENIX Symposium on Networked Systems Design and Implementation (NSDI 21)*. 439–455.

[26] Graham Gobieski, Ahmet Oguz Atli, Kenneth Mai, Brandon Lucia, and Nathan Beckmann. 2021. Snafu: an ultra-low-power, energy-minimal CGRA-generation framework and architecture. In *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 1027–1040.

[27] Graham Gobieski, Brandon Lucia, and Nathan Beckmann. 2019. Intelligence beyond the edge: Inference on intermittent embedded systems. In *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*. 199–213.

[28] Graham Gobieski, Amolak Nagi, Nathan Serafin, Mehmet Meric Isgenc, Nathan Beckmann, and Brandon Lucia. 2019. Manic: A vector-dataflow architecture for ultra-low-power embedded systems. In *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*. 670–684.

[29] Andres Gomez, Lukas Sigrist, Michele Magno, Luca Benini, and Lothar Thiele. 2016. Dynamic energy burst scaling for transiently powered systems. In *2016 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 349–354.

[30] Josiah Hester, Timothy Scott, and Jacob Sorber. 2014. Ekho: Realistic and Repeatable Experimentation for Tiny Energy-harvesting Sensors. In *Proceedings of the 12th ACM Conference on Embedded Network Sensor Systems* (Memphis, Tennessee) *(SenSys '14)*. ACM, New York, NY, USA, 330–331.

[31] Josiah Hester, Lanny Sitanayah, and Jacob Sorber. 2015. Tragedy of the coulombs: Federating energy storage for tiny, intermittently-powered sensors. In *Proceedings of the 13th ACM Conference on Embedded Networked Sensor Systems*. 5–16.

[32] Josiah Hester and Jacob Sorber. 2017. Flicker: Rapid prototyping for the batteryless internet-of-things. In *Proceedings of the 15th ACM Conference on Embedded Network Sensor Systems*. 1–13.

[33] Josiah Hester and Jacob Sorber. 2017. The Future of Sensing is Batteryless, Intermittent, and Awesome. In *Proc. SenSys* (Nov. 6–8). ACM, Delft, The Netherlands, 21:1–21:6.

[34] Josiah Hester, Kevin Storer, and Jacob Sorber. 2017. Timely execution on intermittently powered batteryless sensors. In *Proceedings of the 15th ACM Conference on Embedded Network Sensor Systems*. 1–13.

[35] Josiah Hester, Nicole Tobias, Amir Rahmati, Lanny Sitanayah, Daniel Holcomb, Kevin Fu, Wayne P. Burleson, and Jacob Sorber. 2016. Persistent Clocks for Batteryless Sensing Devices. *ACM Trans. Embed. Comput. Syst.* 15, 4 (Aug. 2016), 77:1–77:28.

[36] Matthew Hicks. 2017. Clank: Architectural Support for Intermittent Computation. In *Proc. ISCA* (June 24–28). ACM, Toronto, ON, Canada, 228–240.

[37] Christopher Howe, Paolo Bombelli, Anand Savanth, Alberto Scarampi, Stephen Rowden, David Green, Andreas Erbe, Erland Arstol, Ivana Jevremovic, Martin Hohmann-Marriott, et al. 2022. Powering a Microprocessor by Photosynthesis. (2022).

[38] Texas Instruments Inc. 2017. MSP430FR59xx Mixed-Signal Microcontrollers (Rev. F). Retrieved May 15, 2022 from https://www.ti.com/lit/ds/symlink/msp430fr5969.pdf

[39] MATRIX Industries. 2020. Nanopower Energy Harvesting Synchronous Boost Converter. Retrieved June 15, 2022 from https://www.mouser.com/pdfDocs/MATRIX_Industries_06122019_MCRY12-125Q-42DI-1602534.pdf

[40] Infineon. 2020. 8MB EXCELON LP Ferroelectric RAM. Retrieved June 20, 2022 from https://www.infineon.com/dgdl/Infineon-CY15B108QN_CY15V108QN_Excelon(TM)_LP_8-Mbit_(1024K_X_8)_Serial_(SPI)_F-RAM-DataSheet-v10_00-EN.pdf?fileId=8ac78c8c7d0d8da4017d0ee7134b6ff4

[41] Texas Instruments. 2020. Ultra Low power Harvester power Management IC. Retrieved June 15, 2022 from https://www.ti.com/product/BQ25570

[42] Maxim Integrated. 2004. 256-Tap, Nonvolatile, SPI-Interface, Digital Potentiometers. Retrieved Sept 29, 2022 from https://datasheets.maximintegrated.com/en/ds/MAX5422-MAX5424.pdf

[43] Bashima Islam and Shahriar Nirjon. 2020. Zygarde: Time-Sensitive On-Device Deep Inference and Adaptation on Intermittently-Powered Systems. *Proc. ACM Interact. Mob. Wearable Ubiquitous Technol.* 4, 3.

[44] 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.

[45] Dhananjay Jagtap and Pat Pannuto. 2021. Repurposing cathodic protection systems as reliable, in-situ, ambient batteries for sensor networks. In *Proceedings of the 20th International Conference on Information Processing in Sensor Networks (co-located with CPS-IoT Week 2021)*. 357–368.

[46] Hrishikesh Jayakumar, Arnab Raha, and Vijay Raghunathan. 2014. QuickRecall: A low overhead HW/SW approach for enabling computations across power cycles in transiently powered computers. In *2014 27th International Conference on VLSI Design and 2014 13th International Conference on Embedded Systems*. IEEE, 330–335.

[47] Chih-Kai Kang, Hashan Roshantha Mendis, Chun-Han Lin, Ming-Syan Chen, and Pi-Cheng Hsiu. 2020. Everything leaves footprints: Hardware accelerated intermittent deep inference. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 39, 11 (2020), 3479–3491.

[48] Keith Kirkpatrick. 2014. World without wires.

[49] Vito Kortbeek, Abu Bakar, Stefany Cruz, Kasim Sinan Yildirim, Przemysław Pawełczak, and Josiah Hester. 2020. BFree: Enabling Battery-Free Sensor Prototyping with Python. *Proc. ACM Interact. Mob. Wearable Ubiquitous Technol.* 4, 4, Article 135 (Dec. 2020), 39 pages. https://doi.org/10.1145/3432191

[50] Vito Kortbeek, Souradip Ghosh, Josiah Hester, Simone Campanoni, and Przemysław Pawełczak. 2022. WARio: efficient code generation for intermittent computing. In *Proceedings of the 43rd ACM SIGPLAN International Conference on Programming Language Design and Implementation*. 777–791.

[51] Vito Kortbeek, Kasim Sinan Yildirim, Abu Bakar, Jacob Sorber, Josiah Hester, and Przemysław Pawełczak. 2020. Time-sensitive intermittent computing meets legacy software. In *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*. 85–99.

[52] Christopher Kraemer, Amy Guo, Saad Ahmed, and Josiah Hester. 2022. Battery-free MakeCode: Accessible Programming for Intermittent Computing. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies* 6, 1 (2022), 1–35.

[53] Alex Krizhevsky. 2009. Learning multiple layers of features from tiny images. (2009).

[54] Seulki Lee and Shahriar Nirjon. 2019. Neuro. zero: a zero-energy neural network accelerator for embedded sensing and inference systems. In *Proceedings of the 17th Conference on Embedded Networked Sensor Systems*. 138–152.

[55] Vincent Liu, Aaron Parks, Vamsi Talla, Shyamnath Gollakota, David Wetherall, and Joshua R Smith. 2013. Ambient backscatter: Wireless communication out of thin air. *ACM SIGCOMM computer communication review* 43, 4 (2013), 39–50.

[56] Brandon Lucia, Vignesh Balaji, Alexei Colin, Kiwan Maeng, and Emily Ruppel. 2017. Intermittent Computing: Challenges and Opportunities. In *Proc. SNAPL* (May 7–10). Alisomar, CA, USA, 8:1–8:14.

[57] Brandon Lucia, Brad Denby, Zachary Manchester, Harsh Desai, Emily Ruppel, and Alexei Colin. 2021. Computational Nanosatellite Constellations: Opportunities and Challenges. *GetMobile: Mobile Comp. and Comm.* 25, 1 (jun 2021), 16–23.

[58] Brandon Lucia and Benjamin Ransford. 2015. A simpler, safer programming and execution model for intermittent systems. *ACM SIGPLAN Notices* 50, 6 (2015), 575–585.

[59] Kiwan Maeng, Alexei Colin, and Brandon Lucia. 2017. Alpaca: Intermittent execution without checkpoints. *Proceedings of the ACM on Programming Languages* 1, OOPSLA (2017), 1–30.

[60] Kiwan Maeng and Brandon Lucia. 2018. Adaptive dynamic checkpointing for safe efficient intermittent computing. In *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*. 129–144.

[61] Kiwan Maeng and Brandon Lucia. 2020. Adaptive low-overhead scheduling for periodic and reactive intermittent execution. In *Proceedings of the 41st ACM SIGPLAN Conference on Programming Language Design and Implementation*. 1005–1021.

[62] Amjad Yousef Majid, Carlo Delle Donne, Kiwan Maeng, Alexei Colin, Kasim Sinan Yildirim, Brandon Lucia, and Przemysław Pawełczak. 2020. Dynamic task-based intermittent execution for energy-harvesting devices. *ACM Transactions on Sensor Networks (TOSN)* 16, 1 (2020), 1–24.

[63] Gabriel Marcano and Pat Pannuto. 2022. Soil Power? Can Microbial Fuel Cells Power Non-Trivial Sensors?. In *Proceedings of the 1st ACM Workshop on No Power and Low Power Internet-of-Things (LP-IoT'21)*. Association for Computing Machinery, New York, NY, USA, 8–13.

[64] MaximAI. 2020. Face Identification Using MAX78000. https://www.maximintegrated.com/en/design/technical-documents/app-notes/7/7364.html

[65] MaximAI. 2022. The MAX78000 SDK. https://github.com/MaximIntegratedAI/MAX78000_SDK

[66] MaximAI. 2022. The MAX78000 SDK. https://github.com/MaximIntegratedAI/ai8x-synthesis

[67] Molex. 2020. Molex SlimStack Receptacle. Retrieved June 20, 2022 from https://www.molex.com/molex/products/part-detail/pcb_receptacles/0529910208

[68] Gregory Mone. 2017. Sensors on the Brain. *Commun. ACM* 60, 4 (2017), 12–14. https://doi.org/10.1145/3048380

[69] Alessandro Montanari, Manuja Sharma, Dainius Jenkus, Mohammed Alloulah, Lorena Qendro, and Fahim Kawsar. 2020. ePerceptive: energy reactive embedded intelligence for batteryless sensors. In *Proceedings of the 18th Conference on Embedded Networked Sensor Systems*. 382–394.

[70] James Myers. 2020. Project Triffid: Billions of Batteryless Arm Devices? Arm Blueprint. https://www.arm.com/blogs/blueprint/batteryless-arm-devices

[71] Joseph A Paradiso. 2020. Technical perspective: The future of large-scale embedded sensing. *Commun. ACM* 63, 12 (2020), 91–91.

[72] Powercast. 2016. The Powercast P2110B Powerharvester. Retrieved June 17, 2022 from https://www.powercastco.com/wp-content/uploads/2016/12/P2110B-Datasheet-Rev-3.pdf

[73] Powercast. 2018. The Powercast TX91501B Powercaster. Retrieved June 17, 2022 from https://www.powercastco.com/wp-content/uploads/2019/10/User-Manual-TX-915-01B-Rev-A-1.pdf

[74] PyTorch. 2020. PyTorch. https://pytorch.org/

[75] Qoitech. 2020. Otii Arc. Retrieved June 17, 2022 from https://www.qoitech.com/otii/

[76] Amir Rahmati, Mastooreh Salajegheh, Dan Holcomb, Jacob Sorber, Wayne P. Burleson, and Kevin Fu. 2012. TARDIS: Time and Remanence Decay in SRAM to Implement Secure Protocols on Embedded Devices without Clocks. In *Proc. Security* (Aug. 8–10). USENIX, Bellevue, WA, USA, 1–16.

[77] Benjamin Ransford, Jacob Sorber, and Kevin Fu. 2011. Mementos: System support for long-running computation on RFID-scale devices. In *Proceedings of the sixteenth international conference on Architectural support for programming languages and operating systems*. 159–170.

[78] Emily Ruppel and Brandon Lucia. 2019. Transactional concurrency control for intermittent, energy-harvesting computing systems. In *Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation*. 1085–1100.

[79] Alanson P. Sample, Daniel J. Yeager, Pauline S. Powledge, Alexander V. Mamishev, and Joshua R. Smith. 2008. Design of an RFID-based battery-free programmable sensing platform. *IEEE Trans. Instrum. Meas.* 57, 11 (Nov. 2008), 2608–2615.

[80] Esther Shein. 2021. A Battery-Free Internet of Things. *Commun. ACM* 64, 7 (June 2021), 16–18. https://doi.org/10.1145/3464937

[81] Sparkfun. 2020. Qwiic Connect System. Retrieved June 15, 2022 from https://www.sparkfun.com/qwiic

[82] Sparkfun. 2020. What is MicroMod? Retrieved June 15, 2022 from https://www.sparkfun.com/micromod

[83] Milijana Surbatovich, Limin Jia, and Brandon Lucia. 2021. Automatically enforcing fresh and consistent inputs in intermittent systems. In *Proceedings of the 42nd ACM SIGPLAN International Conference on Programming Language Design and Implementation*. 851–866.

[84] Vamsi Talla, Bryce Kellogg, Benjamin Ransford, Saman Naderiparizi, Joshua R. Smith, and Shyamnath Gollakota. 2017. Powering the next Billion Devices with Wi-Fi. *Commun. ACM* 60, 3 (2017), 83–91.

[85] TensorFlow. 2020. TensorFlow Lite for Microcontrollers. https://www.tensorflow.org/lite/microcontrollers

[86] Pete Warden. 2018. Speech commands: A dataset for limited-vocabulary speech recognition. *arXiv preprint arXiv:1804.03209* (2018).

[87] Yawen Wu, Zhepeng Wang, Zhenge Jia, Yiyu Shi, and Jingtong Hu. 2020. Intermittent inference with nonuniformly compressed multi-exit neural network for energy harvesting powered devices. In *2020 57th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 1–6.

[88] Fan Yang, Ashok Samraj Thangarajan, Sam Michiels, Wouter Joosen, and Danny Hughes. 2021. Morphy: Software Defined Charge Storage for the IoT. In *Proceedings of the 19th ACM Conference on Embedded Networked Sensor Systems*. 248–260.

[89] Kasım Sinan Yıldırım, Amjad Yousef Majid, Dimitris Patoukas, Koen Schaper, Przemyslaw Pawelczak, and Josiah Hester. 2018. Ink: Reactive kernel for tiny batteryless sensors. In *Proceedings of the 16th ACM Conference on Embedded Networked Sensor Systems*. 41–53.

[90] Eren Yıldız, Lijun Chen, and Kasim Sinan Yıldırım. 2022. Immortal Threads: Multithreaded Event-driven Intermittent Computing on {Ultra-Low-Power} Microcontrollers. In *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22)*. 339–355.

[91] Eren Yildiz and Kasim Sinan Yildirim. 2020. Defragmenting Energy Storage in Batteryless Sensing Devices. In *Proceedings of the 8th International Workshop on Energy Harvesting and Energy-Neutral Sensing Systems*. 36–42.

[92] Hong Zhang, Jeremy Gummeson, Benjamin Ransford, and Kevin Fu. 2011. Moo: A batteryless computational RFID and sensing platform. *University of Massachusetts Computer Science Technical Report UM-CS-2011-020* (2011).