RESEARCH-ARTICLE

Adaptive Computing in Memory Meets Conventional Batteryless Platforms

KHAKIM AKHUNOV, University of Trento, Trento, TN, Italy

KASIM SINAN YILDIRIM, University of Trento, Trento, TN, Italy

JONGOUK CHOI, University of Central Florida, Orlando, FL, United States

CHANGHEE JUNG, Purdue University, West Lafayette, IN, United States

Published: 08 October 2025
Online AM: 01 September 2025
Accepted: 22 August 2025
Revised: 02 June 2025
Received: 31 October 2024

Adaptive Computing in Memory Meets Conventional Batteryless Platforms

KHAKIM AKHUNOV, Department of Information Engineering and Computer Science, University of Trento, Trento, Italy and imec, Leuven, Belgium
KASIM SINAN YILDIRIM, Department of Information Engineering and Computer Science, University of Trento, Trento, Italy
JONGOUK CHOI, Department of Computer Science, University of Central Florida, Orlando, United States
CHANGHEE JUNG, Computer Science, Purdue University, West Lafayette, United States

Computing In-Memory (CIM) with emerging nonvolatile memory (NVM) technologies is promising for batteryless systems since it removes the need for explicit backup and energy-hungry data transfer between the processor and memory. However, existing CIM solutions are not effective in accelerating memory-bound inference tasks efficiently on batteryless systems. They operate at relatively low frequencies, complicate application development, and do not consider energy harvesting dynamics to optimize their throughput. To address the issues, this article presents a novel CIM-based batteryless computing platform, called Viadotto, that provides efficient and adaptive acceleration for memory-bound computing workloads. Viadotto meets adaptive CIM and microcontroller-based (MCU-based) conventional batteryless platforms for the first time. Basically, Viadotto exposes a programming model supported by its compiler and a pipelined memory controller, which hides low-level CIM operations from applications. Furthermore, its runtime issues CIM operations in an energy-efficient manner and optimizes throughput in a programmer-transparent way by adapting CIM parallelism to react to ambient power dynamics. Our evaluation shows that Viadotto outperforms existing CIM solutions for batteryless systems by 48%.

CCS Concepts: • **Computer systems organization → Embedded hardware;**

Additional Key Words and Phrases: Intermittent computing, computing in-memory, adaptive system, computational RAM

ACM Reference Format:
Khakim Akhunov, Kasim Sinan Yildirim, Jongouk Choi, and Changhee Jung. 2025. Adaptive Computing in Memory Meets Conventional Batteryless Platforms. *ACM Trans. Embedd. Comput. Syst.* 24, 6, Article 159 (October 2025), 26 pages. https://doi.org/10.1145/3765623

1 Introduction

Over the past decade, researchers have enabled eco-friendly and low-power solutions for **Internet of Things** (**IoT**) by developing batteryless energy-harvesting devices [3]. Such energy-bound edge

Authors' Contact Information: Khakim Akhunov, Department of Information Engineering and Computer Science, University of Trento, Trento, Trentino–Alto Adige, Italy and imec, Leuven, Flemish Brabant, Belgium; e-mail: khakim.akhunov@imec.be; Kasim Sinan Yildirim, Department of Information Engineering and Computer Science, University of Trento, Trento, Trentino–Alto Adige, Italy; e-mail: kasimsinan.yildirim@unitn.it; Jongouk Choi, Department of Computer Science, University of Central Florida, Orlando, Florida, United States; e-mail: jongouk.choi@ucf.edu; Changhee Jung, Computer Science, Purdue University, West Lafayette, Indiana, United States; e-mail: chjung@purdue.edu.

ACM 1539-9087/2025/10-ART159
https://doi.org/10.1145/3765623

devices are increasingly used in environmental monitoring, wildlife tracking, and in-situ machine learning [13, 14, 36, 44, 65]. For instance, animal-borne sensors [31, 67] powered by solar energy must classify sounds or images in real time to detect species-specific behaviors. Battery-free air quality monitors [80] embedded in buildings or factories often perform on-device classification using memory-intensive **convolutional neural networks** (**CNNs**). Similarly, agricultural sensors [38, 50] must periodically process high-resolution visual or acoustic data to detect plant health or pest activity, where a task is often limited by memory bandwidth and energy availability. These applications are memory-bound due to the size of the input data and models.

These devices operate exclusively by gathering energy from ambient energy sources and storing the harvested energy in their storage capacitor [2, 12, 26, 28, 58]. Given the spatio-temporal variability of ambient power, batteryless devices are subject to frequent power failures and cannot operate continuously. When the capacitor energy is depleted, such devices are power-interrupted to charge the capacitor and wake up again. That is, the execution is *intermittent* due to the unstable nature of energy harvesting. Power failures during program execution can result in the loss of volatile computational states (e.g., local/global variables and registers), preventing the forward progress of computation. Batteryless devices rely on nonvolatile memory to back up such states before a failure occurs, and they restore these states whenever power comes back [20, 83, 90].

Due to frequent power failures in intermittent systems, backing up the computational states creates significant traffic between the processor and nonvolatile memory, leading to considerable time and energy overhead [79]. This traffic is exacerbated by memory-bound workloads. The computing in-memory [33, 66] paradigm creates an opportunity to reduce this traffic by employing nonvolatile memory as a processing element. In particular, **computational RAM** (**CRAM**) is an emerging memory architecture that can carry out computations inside the memory by directly setting memory elements to store the output of the logic operations [82]. CRAM with nonvolatile memory elements is ideal for intermittent computing since it is *power-failure resilient*, ensuring that each in-memory operation is failure-atomic [73]. This feature enables the forward progress of computation on the fly, eliminating the need for explicit backup and energy-hungry data transfer out of memory. It also facilitates the efficient intermittent execution of memory-bound computational loads directly in nonvolatile memory with a high degree of parallelism.

Only two recent works utilized CRAM for intermittent computing. Resch et al. [73, 74] have presented a custom stand-alone chip built using CRAM for machine learning-based energy harvesting applications. This chip's primitive **memory controller** (**MC**) fetches a low-level instruction from a dedicated CRAM tile and issues the instruction on the tiles for data. Unfortunately, programming this chip is a challenge. In particular, programmers need to use their primitive assembly instructions to manage all the data flow and intermittent processing within tiles at the lowest level, which makes this design very rigid from a practical point of view. Besides, Akhunov et al. [7, 8] have proposed a more flexible MCU-based system, which runs the general application tasks and uses CRAM as an in-memory accelerator for data-intensive operations. Programmers use high-level C language interfaces to map memory-bound tasks into this accelerator. However, they still need to manually keep track of data allocation and computational flow on tiles, complicating application development significantly. Furthermore, the architecture involves a second MCU as an MC for CRAM, much more power-consuming than the CIM hardware (by multiple orders of magnitude).

Another problem with the mentioned solutions is their inefficiency in terms of throughput. Firstly, they are limited to operating at relatively low frequencies due to their restricted MCs. Besides, they do not consider energy harvesting dynamics to optimize their throughput. Adaptation and response to energy harvesting dynamics are essential to intermittent computing systems. For instance, activating more CRAM tiles to increase the degree of parallelism might lead to more frequent power failures, decreased throughput, and even no computational progress when the

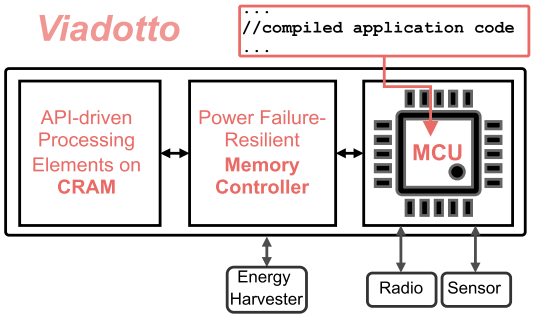


Fig. 1. Viadotto comprises a CRAM with API-drive processing elements, an MC, and an MCU. The MCU controls peripherals and offloads the macroinstructions that represent data-intensive computational tasks on CRAM through the MC. The MC receives macroinstructions and executes them intermittently on CRAM.

ambient power is low. Similarly, when the ambient power is high, there is an opportunity to activate more tiles to perform more tasks and increase the throughput.

In short, existing CIM solutions targeting batteryless systems are promising initial steps; however, they are still far behind in providing energy-efficient and effective intermittent acceleration for memory-bound computing tasks, e.g., machine learning inference. Hence, their practicality and, in turn, widespread adoption are major concerns. This calls for a new CIM solution that can handle three main challenges while ensuring transparency by concealing complexity from applications. These challenges are (Ch1) managing space allocation and dataflow on CRAM to provide full programming flexibility and transparency, (Ch2) optimizing throughput by adapting tile-level parallelism to react to ambient power dynamics, and (Ch3) integration of emerging CRAM technology into conventional MCU-based batteryless platforms via an efficient hardware interface.

This article introduces Viadotto, the first CIM-based batteryless computing platform that offers efficient and adaptive acceleration for memory-bound computing workloads. Viadotto uniquely combines adaptive CIM and MCU-based conventional batteryless platforms, bringing them together for the first time. As depicted in Figure 1, Viadotto comprises an MCU, which executes general-purpose computational and sensing tasks using de facto backup strategies for intermittent computing. While programmers interact with Viadotto through its high-level API-based programming model, mapping memory-bound tasks from their C programs into CRAM, the platform seamlessly handles the low-level logic. This is achieved by Viadotto's runtime library, which is automatically linked with the application binary produced by the Viadotto compiler. The compiler abstracts the low-level CIM operations by generating optimized versions of the CRAM code that take advantage of tile-level parallelism. This approach enables the system to adapt dynamically to varying ambient power conditions by selecting the most performant code version based on the available power. The intricacies of data flow management and low-level operations in CRAM are managed by Viadotto's power failure-resilient MC, ensuring reliable execution even during power interruptions. Our simulation-based evaluation against the state-of-the-art showed that Viadotto outperforms the existing CIM solutions for intermittent computing by 48% in terms of inference throughput.

In summary, Viadotto presents the following key novelties:

(1) Viadotto is the first end-to-end sensing platform that integrates emerging CIM technology with conventional MCU-based batteryless computing.
(2) Viadotto's unique power failure-resilient MC bridges conventional MCU-based intermittent computing and CIM. It enables the pipelined execution and improves the operating frequency by more than 3× compared to existing intermittent CIM solutions.

(3) Viadotto is the first that dynamically controls the tile-level parallelism to improve throughput under irregular ambient power and reduce the number of power failures.
(4) Viadotto is the first, thanks to its compiler, MC, and runtime, to expose a high-level programming interface that enables programmers to accelerate their memory-bound tasks intermittently without having to deal with intricate CIM operations and adaptation logic.

2 Batteryless Compute-In-Memory

Most batteryless systems are built around microcontrollers that incorporate both volatile memory (e.g., SRAM) and nonvolatile memory in their architecture [28, 41, 47]. In these systems, power failures are frequent and can interrupt program execution, leading to non-termination if the volatile state is lost and computation must restart from the beginning [63]. To mitigate this issue, many systems back up the volatile computational state into NVM before a power outage and restore it upon recovery. However, this introduces crash inconsistency, where a power failure during a backup operation may interrupt writes to NVM and leave data in a partially updated state [72]. This is particularly problematic in the presence of **Write-After-Read** (**WAR**) dependencies, where a program reads a value, modifies it, and writes it back. If power fails after the read but before the write completes, the subsequent recovery may operate on stale or inconsistent data, violating the expected idempotence of restartable computation [23, 59]. Such consistency issues are further exacerbated in memory-bound on-device intelligent tasks, which often process data streams or maintain internal state over time, requiring both precise synchronization between volatile and nonvolatile memory and robust support for atomic, repeatable operations across intermittent power cycles.

Several solutions have been proposed to eliminate non-termination and keep NVM consistent during intermittent execution [19, 21, 23, 62, 64, 76, 88]. For instance, the checkpoints [6, 20, 48, 59, 63, 63, 79, 90] back up the current program state in NVM at specific points in the code. In the wake of power failure, a recovery runtime restores the backed-up state and resumes the power-interrupted program. One way to trigger checkpoints is to check the energy storage periodically and only back up when the voltage hits a certain low threshold, known as **just-in-time** (**JIT**) checkpoint. In general, checkpoints introduce a non-negligible time and energy cost due to the traffic between the processor and NVM. This traffic is exacerbated by the *memory-bound* on-device intelligence tasks, which are becoming prevalent in batteryless systems [36, 43, 53, 65]. The reason is that, during intermittent inference, model parameters and input/output activations are stored in NVM. The processor needs to bring these values into its registers, perform operations, and write back the results in NVM. Similarly, control and datapath overheads contribute significantly to the energy overhead [37].

The CIM paradigm creates an opportunity to process memory-bound inference tasks directly in NVM efficiently. Moreover, it offers parallelism that cannot be supported by the de facto MCU-based platforms for batteryless computing. CIM can activate different blocks in NVM to compute independent computational tasks in parallel. Besides, dynamic ambient power creates an opportunity to adapt the level of parallelism by consuming excess energy that cannot be stored in small storage capacitors [4, 6, 16, 61]. In short, CIM has great potential to boost the throughput of intermittent applications with modern machine learning workloads.

2.1 Prior Works on Computing in NVM

Qui et al. [71] have introduced the first intermittent accelerator for MAC operations built on top of RRAM crossbars, which is an emerging NVM that offers CIM with resistive memory [17, 91]. Unfortunately, the RRAM crossbar suffers from lower endurance and requires **analog-to-digital converters** (**ADCs**), causing significant area and energy overhead for batteryless systems.

MOUSE [73, 74] and PiMCo [7, 8] utilize a **magnetic tunnel junction** (**MTJ**)-based CRAM, a highly energy-efficient memory substrate suitable for intermittent computing. CRAM can implement logic functions (e.g., NOT, BUFF, AND, NAND, OR, NOR, and MAJ3) when predefined voltage values are applied to its bit lines. CRAM backs up atomically and immediately, ensuring forward progress and memory consistency—see [73, 92] for more details.

Current batteryless CIM solutions show promise as initial steps. However, they still lag behind in providing energy-efficient and practical intermittent acceleration for memory-bound computing tasks. MOUSE does not include an MCU and is a stand-alone application-specific chip based on CRAM. It allocates different CRAM tiles for program instructions and data. Its MC reads the instruction tiles and issues the instructions on the target data tiles. It imposes a massive programming burden due to its restricted architectural design, tile organization, and primitive instruction set, forcing programmers to explicitly manage all the data flow and parallelism at the lowest level.

The CIM paradigm is suitable for data-intensive workloads, but MCU is critical to support other types of tasks, such as resource management, data sensing, preprocessing, and transmitting. PiMCo introduces a high-level MCU-centered batteryless architecture that uses CIM as an accelerator. However, this architecture requires manual tracking of data allocation and computational flow on tiles, complicating application development significantly. Besides, it uses a second MCU as an MC for CRAM, which is significantly more power-consuming than the CIM hardware.

Both MOUSE and PiMCo have no compilation and adaptation support, which is crucial for efficient response to sporadic ambient power [6, 15].

2.2 Challenges of Intermittent Computing in NVM

Viadotto addresses the following unique challenges to boost the efficiency of intermittent computing systems by utilizing CIM via CRAM. (1) Firstly, CRAM requires offloading data to its tiles for computing in memory and aligning the data in such a way that data movement within its tiles is minimized while the degree of parallelism is kept high. Since the entire CRAM space is available both for performing operations and for holding data, CRAM tile allocation and data alignment policy become a crucial issue. Viadotto's compiler performs tile allocation so that data allocation and alignment within the tiles obey a specific layout (see Sections 3.1 and 3.3.2). (2) Secondly, CRAM requires the processed input and output data to be aligned either vertically or horizontally, depending on the parallelism approach used: row or column [92]. The reuse of memory space plays a critical role in seamlessly running complex data-intensive applications despite memory constraints. Viadotto removes the burden of keeping track of dataflow and logic operations in memory owing to its MC, microinstructions, and tile layout (see Sections 3.1, 3.2, and 3.3.2). (3) Thirdly, abstracting away the details of the low-level architecture and making the CRAM as transparent as possible not only to the developers but also to the varying power and energy availability during CIM operations is not easy. Viadotto enables full transparency thanks to its compiler run-time adaptation that supports its high-level programming model (see Sections 3.3 and 3.3.2). (4) Lastly, the mentioned challenges make integration of the emerging CRAM technology into conventional MCU-based batteryless platforms even more challenging, preventing the widespread adaption and practicality of CIM-based intermittent computing. Viadotto overcomes this challenge thanks to its advanced MC interface (see Sections 3.2).

3 Viadotto: Adaptive Intermittent CIM

Viadotto is the first CIM-based end-to-end batteryless computing platform. It meets adaptive CIM with conventional batteryless computing, and efficiently accelerates memory-bound workloads. Viadotto comprises three key components abstracted in Figure 1 and detailed in this section: the

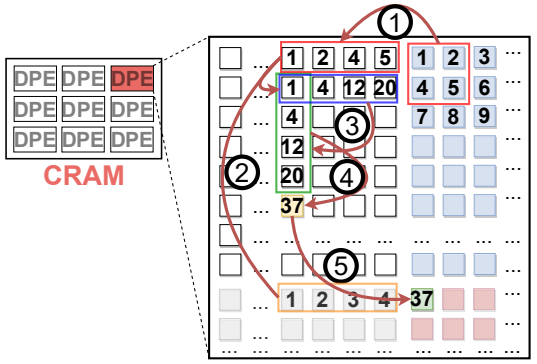


Fig. 2. In-memory computation flow and the conceptual structure of an individual DPE.

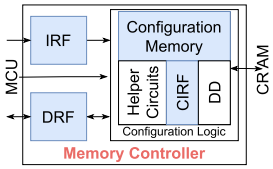
Dataflow Processing Elements (DPEs) on CRAM, the MC, and the MCU—which is the heart of any embedded device—interfacing several peripherals. The MCU provides a general programming model that hides the underlying technical details and employs the standard JIT checkpointing to make forward progress despite power failures while executing the application. It offloads data-intensive computational tasks to CRAM, which performs energy-efficient parallel computation in NVM. Viadotto provides a uniform hardware/software interface, achieving seamless connection between general-purpose MCU and CRAM.

(1) Viadotto structures CRAM tiles as DPEs, which group MTJs into several tiles and can execute a set of logical operations in memory. This organization helps to trace the flow of data and operations, making it effortless to map data-intensive computations to CRAM. (2) MC facilitates programming flexibility. It is responsible for receiving high-level instructions from the MCU and converting them into appropriate commands and signals to trigger the necessary operations on the DPEs. It requires no actions from programmers to handle power failures during computation in memory. The MC and MCU share the common bus and communicate via memory-mapped registers. (3) With Viadotto's compiler and library support, programmers can use a high-level C language supported and need not deal with low-level CRAM operations. (4) Viadotto enables adaptive control of tile-level parallelism, allowing the intermittent system to scale with respect to varying dynamic power, thereby increasing the throughput.

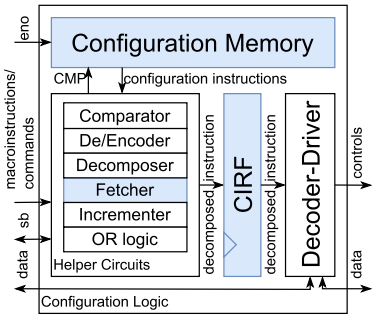
3.1 Microarchitectural Structure of CRAM

In Viadotto, CRAM comprises tiles of uniform size, called DPEs, which have no direct connections with each other since they are self-sufficient computational blocks (see Figure 2, left-hand side). This approach sacrifices memory space for redundant data but eliminates the data movement bottleneck and the necessity of shared data access scheduling. Moreover, the absence of inter-tile wiring frees the space for CRAM cells. Each DPE is a full-fledged computing unit that can independently work on an assigned task. When DPEs execute simultaneously, they process different input data but perform the same CRAM operation at each clock cycle, which simplifies the control path from the MC. Our design allocates four distinct regions (see Figure 2, right-hand side) within DPE for specific purposes: to store two operands (blue and gray cells); to perform in-memory computation (white cells); and to store the result (red cells). This layout simplifies the design and facilitates the mapping of an application to the DPEs [7].

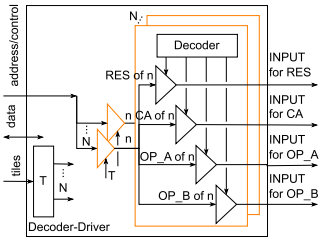
Dataflow. The arrows inside the DPE depicted in Figure 2 show the data flow during the MAC (multiply and accumulate) computation. Before starting an operation, the initial data is placed in the input sections (i.e., two operands). ①When an in-memory operation is requested, essential



(a) High-level architecture showing the flow of data and control between the MCU, MC, and CRAM tiles.



(b) Internal configuration logic of the MC, highlighting how macroinstructions are mapped to low-level control sequences.



(c) Simplified schematic of the Decoder-Driver (DD) component, which translates configuration instructions into CRAM activation signals.

Fig. 3. Overview of Viadotto's MC.

data from the blue section of activated DPEs is brought and properly aligned in the white cells. ②Then, the elements of two operands are simultaneously multiplied. At this stage, the intermediate result appears in a row below (blue-framed) due to column-level parallelism exploited by Viadotto. ③Next step is lining up the multiplication results in a column to prepare for simultaneous addition. ④The summed-up result again appears at the bottom (yellow cell). ⑤Finally, the result is moved to the result section. A more detailed dataflow presentation can be found in [7].

Structured Parallelism. Note that the CRAM layout is out of the scope of our work, but thanks to the flexible MC, other options are possible. We recognize that the current layout of CRAM allows for different levels of parallelism. First, since our implementation takes advantage of the *column-level parallelism*, several bits, bytes, and even words in a row can be processed in parallel. Second, additional connections between columns in DPEs enable inter-column data transfers, which is the *row-level parallelism*. This parallelism helps to copy a block of data between columns at once. Third, CRAM allows for the simultaneous processing of multiple independent data in different DPEs. For example, Viadotto can simultaneously apply a convolutional kernel to distinct input feature maps, employing the *tile-level parallelism*.

3.2 MC

To efficiently bridge the gap between general-purpose processing and in-memory computing, Viadotto incorporates a dedicated MC between the MCU and CRAM. Building upon the tile-based architecture introduced in Section 3.1, the MC orchestrates high-level control of CRAM operations, abstracting low-level hardware complexity from the programmer and maintaining robustness even under intermittent power failures discussed in Section 4. The MC enables the MCU to offload data-intensive tasks by accepting macroinstructions (high-level commands) such as add, multiply, or memory copy operations. Internally, the MC translates these instructions into sequences of low-level signals that trigger the corresponding in-memory computations across active DPE tiles.

Figure 3(a) shows the big picture: the MCU communicates with the MC by sending high-level instructions and data. The MC then configures CRAM and manages dataflow without any programmer intervention to handle power failures or synchronization. The main building blocks of the MC and their functionality are as follows. (1) **Register Files** (**RFs**) composed of **Instruction RF** (**IRF**) that holds the high-level operation instructions, **Data RF** (**DRF**) that stores input and output data,

Table 1. High-level Instructions from MCU to MC

Instruction Type	High-level Instructions
Tile activate/deactivate	`cram_tile_enable, cram_tile_disable`
Memory Read/Write	`cram_load_in`, `cram_load_sd`, `cram_unload_out`
Memory Copy	`cram_move_out_in`
Execute	`cram_add`, `cram_mul`, `cram_relu`, `cram_maxpool`, `cram_mac`

and **Configuration Instruction RF** (**CIRF**) that temporarily stores a configuration instruction fetched from memory. (2) **Configuration Memory** (**CM**) stores predefined configuration instruction (microinstruction) sequences for in-memory computations such as addition, multiplication, and MAC, and acts like a lightweight controller firmware that can be easily addressed and triggered. (3) Helper Circuits include the set of trivial logic circuits to ensure smooth MC functioning. (4) **Decoder-Driver** (**DD**) converts configuration instructions into hardware-level control signals for the CRAM tiles and ensures synchronized execution across active DPEs. The blocks in blue are nonvolatile storage (either registers [77, 86] or memory [42]) to enable the MC power failure-resilience. While commercial nonvolatile registers are not available, we rely on published data from [77, 86], which describe nonvolatile register designs for nonvolatile processor prototypes. As presented in Section 5.1, we implement the logic of the MC in VHDL, using the state-of-the-art technique [75] to emulate both nonvolatile memory and registers and estimate timing, power, and area consumption of the MC to include it in our simulation.

Table 1 summarizes the simple set of high-level instructions the MCU uses to control CRAM operations. This abstraction allows developers to program Viadotto using familiar C-style APIs without delving into the complexity of MC management. For example, a single macroinstruction like `cram_mac` triggers a multiply-and-accumulate operation across several DPEs in parallel, handled automatically by the MC through fetching, decoding, and dispatching the correct sequence of hardware operations. As shown in the internal configuration logic of the MC in Figure 3(b), the MCU enables the CM by the `eno` signal and then can point to the configuration instruction set in the CM, corresponding to any macroinstruction from the table by putting the relevant value in the `CMP` (Configuration Memory Pointer). The macroinstruction-related actions on DPEs take place following the proceeding configuration instructions from the CM addressed by incrementing the `CMP`. The MC decodes and drives a single configuration instruction at each clock cycle, i.e., one logic gate is executed in-memory per clock cycle on all active tiles. The simplified schematic of the DD is presented in Figure 3(c). This block is a combinational circuit containing a set of wired logic gates, tri-state buffers, and decoders. The circuitry is responsible for converting incoming signals to another set of signals to trigger word/logic lines and bitline drivers for CRAM computations, as discussed in Section 3.1. In other words, DD directs the necessary set of signals to the required DPEs and their memory sections. As seen in the figure, each output line of the DD is composed of $N \times n$ wires, where N is the number of tiles, while n is the number of columns in a tile. The DD receives these and other values as input from the CIRF.

MC's Operational Flow. The MC follows the **finite state machine** (**FSM**) presented in Figure 4. The MCU uses the `sb` pin (see Figure 3(b)), connected to a status bit register, to check the current status of the MC before requesting any operation. For example, when it is necessary to write data to a DPE, the MCU first ensures the IDLE status of the MC, by checking the state of the status bit register, and writes data to the DRF. Once the DRF is filled, the MCU sends an instruction to the IRF and enables the MC components regarding the instruction type; the CM and its decoder are activated for any instruction type. Then, a configuration instruction pointed out by `CMP` is decoded and brought to the CIRF, which triggers another set of MC components to proceed with

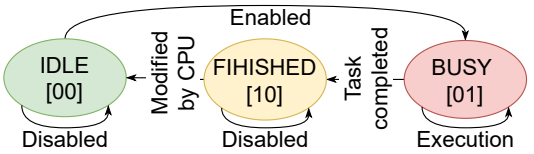


Fig. 4. FSM of the MC. The MCU can check the status of the MC through a status bit register.

the requested operation. Note that the first configuration instruction after enabling the CM always sets the status of the status bit register to BUSY, so the MCU does not interrupt the MC during in-memory operations. The MC stays BUSY until it finishes a requested CRAM operation. After that, the last configuration instruction changes the status of the status bit register to FINISHED and triggers the interrupt signal to the MCU. Finally, the MCU's interrupt handler performs necessary actions (e.g., reads data from the DRF) and triggers the status bit register to IDLE, denoting the MC is ready for the next instruction.

MC's Pipelined Execution. The MC's operation during the BUSY state has three logical stages: the MC fetches a configuration instruction from the CM, decomposes the instruction into the signals for the DD, and executes the operation on CRAM. As seen in Figure 3(b), the MC has units dedicated to these stages: Fetcher, Decomposer, and DD, respectively. We use the register for a fetched configuration instruction and CIRF as communication points between the stages and also as pipeline registers to run all three stages in parallel. Initially, the first configuration instruction is fetched from CM and `CMP` is incremented. In the next clock cycle, the second configuration instruction is fetched while the first one is being decomposed. The next clock cycle makes the pipeline full, executing the first configuration instruction, decomposing the second instruction, and fetching the third one simultaneously. Pipelining increases the CRAM CIM operating frequency, aligning its performance with that of MCUs in batteryless systems.

3.3 MCU-managed Adaptation and Programming Model

3.3.1 Adaptation to Fluctuating Ambient Power. Viadotto dynamically adjusts its tile-level parallelism to match available ambient energy. This mechanism ensures that in-memory computations proceed efficiently without exceeding the system's energy budget, which is especially critical in batteryless environments with intermittent power supply. To manage this, Viadotto monitors the input power level and selectively activates 1, 2, 4, 8, 16, or 32 CRAM tiles. The upper bound is driven by hardware constraints, specifically, area and energy tradeoffs, as activating more than 32 tiles simultaneously would significantly increase peak power consumption and stress the charge retention capabilities of the energy buffer [73]. We chose exponential scaling of tile-level parallelism to simplify runtime decision logic and compiler-generated code paths. This structure enables fast lookup-based configuration selection and efficient batch-based memory partitioning. Supporting arbitrary tile counts (e.g., 6 or 20) would introduce substantial runtime management and synchronization complexity with marginal throughput benefit. While the six fixed configurations are currently hardcoded, the system can be extended to support finer granularity with additional controller logic and compiler support if needed. The selection of the tile count to activate depends on a runtime estimate of available power, enabling Viadotto to adapt its throughput to current energy conditions.

Power Monitoring and Estimation. Viadotto employs a lightweight power estimation technique introduced in [9], which does not rely on ADC measurements or external hardware. Instead, it tracks three voltage thresholds during capacitor charge/discharge cycles: the system powers on and execution begins (V_{on}), the system enters low-power sleep mode (V_{sleep}), and the system performs

Table 2. Viadotto API Functions to Abstract Away Hardware Operations

Function	Description
void collect(char* data_in)	Collect sensor data
void CIM(char* data_in, char* data_out, char* func)	Map the sensor data to free tile(s) for a given function, i.e., add, mul, relu, maxpool, and mac
void transmit(char* data_out)	Transmit inference result

backup and powers down (V_{off}). The time elapsed between the thresholds is measured and used to estimate the ambient input power (P_{amb}):

$$P_{amb_1} = \frac{E_{on-sleep}}{T_{on}}, P_{amb_2} = \frac{E_{sleep-off}}{T_{off}},$$

where $E_{on-sleep}$ is the energy stored between V_{on} and V_{sleep}, $E_{sleep-off}$ is the energy stored between V_{sleep} and V_{off}, T_{on} and T_{off} are time measured from the V_{sleep} trigger to V_{on} and V_{off}, respectively. Note that E is know at design time, while T is measured by the MCU's internal timer. The estimated P_{amb} values are collected and averaged, using the trivial history window approach [6].

Tile Selection Logic. The estimated ambient power P_{amb} is then compared against a table of pre-characterized power consumption values P_{exec} for each tile configuration. This table is provided in a configuration file during deployment. At runtime, Viadotto performs a simple comparison: $P_{amb} \geq P_{exec}(N_{tile})$. It chooses the largest N_{tile} that satisfies this condition. If no estimate is available (e.g., first execution), Viadotto starts conservatively with a 1-tile configuration. As execution proceeds, it updates P_{amb} estimates and adjusts the tile count accordingly.

3.3.2 Programming Interface, Runtime, and Compiler. Viadotto runtime supports a core set of high-level APIs that abstract away hardware operations and are implemented as part of the Viadotto library. These include: `collect()` for sensor data acquisition, `CIM()` to map a computation to one or more free DPEs, and `transmit()` to send the result. Table 2 summarizes the function and purpose of these APIs. Their implementation is backed by the Viadotto runtime system, enabling portable and adaptive execution without requiring developers to manage low-level memory interactions.

Runtime Adaptation. Algorithm 1 presents pseudocode that illustrates how the Viadotto runtime adapts to varying input power levels by selecting the appropriate number of CRAM tiles and executing the corresponding version of the MAC operation. The pseudocode captures the general sequence of operations: gathering input data, enabling the required DPEs, loading operands, executing the MAC operation, unloading results, and finally disabling the tiles. This structure is repeated for different tile configurations (e.g., 1, 2, or more), allowing Viadotto to scale adaptively at runtime.

Compiler. Viadotto compiler consists of two parts. First, the front-end compiler is a source-to-source translator, leveraging the LLVM compiler infrastructure [52]. It facilitates Viadotto APIs through a method involving the instrumentation of microinstructions during compilation. In detail, Viadotto transforms an API function in a given program into a call to a corresponding microinstruction or a series of microinstructions, which becomes part of the program itself. Second, the back-end compiler comes into play by linking the instrumented program with the Viadotto library (see Table 1), resulting in the generation of the final binary executable. In particular, Viadotto compiler ensures the runtime does not overwrite data across tiles (i.e., avoiding data hazards). To achieve the hazard-free tile allocation automatically, the compiler conducts an alias analysis on each tile by leveraging idempotent processing [18, 25, 46, 55, 56, 94, 95]. If there is a dependent tile or no tile available to allocate, the compiler inserts a region boundary and forms another

ALGORITHM 1: Viadotto_MAC_Runtime(NTile)

```
n ← NTile
if n = 1 then
    input0 ← get_sensor_data()
    DPE0 ← True                                          ▷ Enable DPE0
    DPE0_OperandA ← input0
    DPE0_OperandB ← kernel
    DPE0_MAC ← True
    output0 ← DPE0_Result
    DPE0 ← False                                         ▷ Disable DPE0
else if n = 2 then
    input0 ← get_sensor_data()
    input1 ← get_sensor_data()
    DPE0 ← True                                          ▷ Enable DPE0
    DPE1 ← True                                          ▷ Enable DPE1
    DPE0_OperandA ← input0
    DPE1_OperandA ← input1
    DPE0_OperandB ← kernel
    DPE1_OperandB ← kernel
    DPE0_MAC ← True
    DPE1_MAC ← True
    output0 ← DPE0_Result
    output1 ← DPE1_Result
    DPE0 ← False                                         ▷ Disable DPE0
    DPE1 ← False                                         ▷ Disable DPE1
    ...                                                  ▷ Similar pattern for 4, 8, 16, 32 tiles
end if
```

CRAM region thereafter so that each region does not cause data hazard issues. Thus, Viadotto compiler support reduces programming effort and effectively exploits parallelism in different CRAM tiles.

Scalability. Viadotto can enable CIM with a large memory chip, but it may face a scalability challenge. As the memory size increases, the Viadotto runtime needs to hold more versions of CIM code, leading to an increase in code size. To address this issue, Viadotto limits the maximum number of versions to six, resulting in an average binary size overhead of only 5%. Moreover, Viadotto can heuristically filter out the worst-performing version of CIM code and adapt to a more efficient version. This aspect is left for our future work.

3.4 Parallelism Between MCU and CRAM

The MCU in Viadotto orchestrates available resources, loads data-intensive tasks to CRAM, and intermittently executes *simultaneous* general tasks, e.g., data sampling and preprocessing, interfacing peripherals, and handling frequent power failures. The communication between the MCU and MC is interrupt-based, allowing the developer to use the MCU's computational resources for the detached part of the application while executing the data-intensive part in memory in parallel. The MCU also controls the number of active tiles for adaptive execution concerning the input power.

4 Viadotto Power Failure Resilience

Viadotto guarantees the forward progress and memory consistency during intermittent operation.

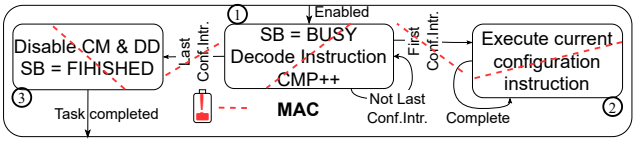


Fig. 5. FSM for the MAC operation under power failures.

CRAM is Inherently Idempotent. Since CRAM cells (MTJs) are nonvolatile, their content persists during power failures, which ensures computation progress. Memory inconsistency, in particular, WAR dependency is also excluded because of the nature of in-CRAM operations. More concretely, the granularity of operations that can be interrupted by a power outage is the gate-level logic operators, which are repeated on power restoration. Even if, for example, the OR operation on two source rows is completed and needs to be re-executed, the result of the operation in the destination row will not change, which makes CRAM inherently idempotent [73].

MC Features Atomic Stages. The forward progress on MC is ensured by utilizing nonvolatile registers proposed in prior work [77, 86]. We assume that the reading of these registers happens in the first half of the clock cycle, while the writing is in the second. However, given the pipelined fashion of the MC organization, keeping data consistent within the MC may seem nontrivial. Figure 5 shows the FSM of the MAC operation split into three stages, where the red dotted lines illustrate places where the signal triggered by the lower voltage threshold can be delivered from the MCU. ① Once the MC is enabled, the SB (status bits) is set to the BUSY state, one configuration instruction is decoded, and the CMP is incremented. ② Then, in the next clock cycle, the decomposed instruction starts executing. Both stages continue to operate in the proceeding clock cycles. ③ When the CMP reaches the last configuration instruction in the set belonging to the MAC instruction, execution switches to the last stage. An interrupt signal at any of the specified points does not affect until the stage is finished, which is guaranteed by considering the energy cost of the most consuming stage in the voltage threshold value.

Backup via JIT Checkpoints. In JIT checkpointing, whenever the voltage level of the capacitor approaches a predefined lower threshold, the MCU must back up the current computational state with the pipeline interrupted. Thus, the remaining energy in the capacitor at this threshold must be enough to perform the backup in a failure-atomic manner. As mentioned in Section 3.3, we adopt a more advanced JIT backup strategy that introduces a pre-checkpoint voltage threshold at which the system transitions to sleep mode with retaining volatile memory and with no backup [9, 60]. The system checks power levels and performs a checkpoint if it's extremely low (i.e., lower than sleep-mode power). Otherwise, the capacitor charges and computation resumes without checkpoints, preventing the re-execution of both the high-level instruction and CRAM primitive. In out implementation, we assume that data sensing and transmitting devices lose their state on power failures. That is, when the V_{sleep} signal interrupts the system, we power gate these devices. The portion of the data sensed and stored in the MCU before a power failure is processed when power restores. This approach significantly reduces the power consumption of the sleep mode and is acceptable since in this work, we do not target real-time mission-critical applications.

Viadotto is explicitly designed to remain operational even under highly constrained energy conditions. When ambient power is low, the platform defaults to minimal tile activation (e.g., 1-tile mode), enabling basic inference to proceed with a near-zero failure rate. As power improves, Viadotto dynamically scales parallelism up to exploit additional energy, thereby optimizing throughput without sacrificing correctness. This ensures a nearly always-on mode of operation: execution may slow, but progress continues.

5 Evaluation and Results

This section presents our evaluation of Viadotto against two state-of-the-art solutions: MOUSE [73] and PiMCo [7].

5.1 Methodology

We develop a simulation infrastructure to simulate Viadotto's DPEs (based on a custom version of CRAM), MC, and compiler-enabled software library. As in MOUSE [73] and PiMCo [7], we can use only simulations to evaluate our system since, to the best of our knowledge, there are no available CRAM-based evaluation boards for real experiments. The simulator components and their parameters are described in this section.

CRAM Structure. Our CRAM design has a *byte-addressable* tiled architecture. Following the common recommendation of subarray size for nonvolatile memories [30], we set the size of each combined tile to 128 KB, which is an array of 1024×1024 bits. Within each DPE, we allocate 512×512 bits for one operand and the same amount for the result, located in the upper-right and lower-right corners of a DPE, respectively. This size is sufficient to store a relatively large output of a CNN layer.

DPE Parallelism. We design a DPE so that it can simultaneously process nine (refers to 3×3 kernel) 8-bit values. We reserve a 128×128 bits array in the upper-left corner for the computation, dedicating 72 (9×8) columns to operand alignment. We limit the number of columns for an operand to avoid massive column activation that consumes a significant amount of energy (about 13 mW for simultaneously active 1,024 columns). We estimate that even for complex 8-bit arithmetic operations (e.g., multiplication) on CRAM, only 25% of the columns need to stay active in parallel. In our design, a single DPE can have 32 (128×0.25) simultaneously activated columns in the worst case. However, if the power budget allows, the programmer can enable additional DPEs, thereby exploiting the tile-level parallelism. The entire CRAM occupies 4 MB of memory. Hence, our Viadotto design has 32 independent DPEs, which together can hold 1,024 active columns in parallel. We assign each tile's remaining part (62 KB) to the second operand.

DPE Energy Requirements. Using parameters presented in [39] for *Modern* MTJ, we calculate the DPEs' energy consumption and execution time by adding the contributions of all the steps involved in the computation: energy and time to preset output values, to execute bitwise logical operations, and to transfer bits to the necessary direction. Additionally, we grasped numbers from NVSim [30] to consider the time and energy consumption for the bit line drivers of CRAM.

MC. Due to the custom design of the memory controller, we model it in VHDL for timing and power estimation. We use Xilinx ISE [84] and target a Spartan 6 lower power XC6SLX16L FPGA with a 45 nm process. The design also implements the SPI-based connectivity between the MC, MCU, and CRAM. We conservatively account for the full FPGA footprint in our area analysis (see Section 5.4), even though only 2% of flip-flops and LUTs are actually used. Importantly, the MC is entirely powered down during Viadotto's deep sleep states, and does not contribute to idle leakage. In low-power active mode, its pipelined structure allows fast task offload and return to sleep, minimizing its energy footprint compared to systems that require longer or repeated wake cycles for equivalent work.

Software Library. As the MCU part of Viadotto, we use the Apollo4 [10] board. The board features an ARM Cortex-M4 processor operating at a maximum of 192 MHz, 2 MB of emerging nonvolatile memory (MRAM), and ultra-low active-mode current consumption (12.3 µA/MHz). Compared to the de facto MCU for intermittent systems (MSP340FR), Apollo4 consumes 10× less current in active mode, is 12× faster, and has 16× larger nonvolatile memory. The efficiency of the board in intermittent computing has already been shown in prior art [47]. We use the datasheet of the board

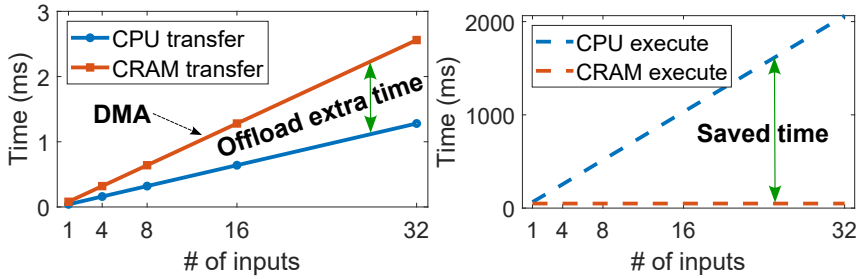
to extract the time and energy consumption of the device booting and the application executing the *viadotto* library and the code during CRAM computation.

Energy Harvesting and Supply. We exploit a single power supply to power Viadotto. Viadotto assumes that the energy buffer must store enough energy to complete at least one atomic in-memory operation plus the associated MCU interaction and backup. This requirement defines the system's minimum energy budget, which corresponds to few nanojoules. However, for the sake of practicality, we suggest extending this minimum energy budget to the need for a reliable execution of a full inference step, including sensing, CRAM computation, results transmission, and backup. Based on profiling of Viadotto's execution with 1-tile configurations (the lowest-power mode), we estimate <5 mJ energy consumption. Therefore, the capacitor must charge till V_{on} to ensure this minimum is met. While Viadotto uses a simple threshold-based heuristic to select tile count, it always begins with a conservative estimate (e.g., 1-tile) if no prior history is available. This ensures safe fallback behavior under sudden drops in ambient energy. We use three prerecorded irradiance power traces from a specific community dataset [49]: night [0.01–0.33 mW], car ride on a sunny day [1–21 mW], and daily indoor-outdoor routine [0–35 mW]. We set the capacitance of the energy buffer to 1 mF, V_{on} to 3.3V, V_{sleep} to 1.45 V, and V_{off} to 1.35 V. While the upper limit is set to ensure that harvested energy (circa 5.5 mJ) is enough to meet the minimum energy budget, other voltage levels are selected empirically and are sufficient to implement the three-threshold monitoring approach (see Section 3.3) in Viadotto. Notice that the levels of V_{sleep} and V_{off} do not need to be dynamic since the backup overhead does not change with the number of active tiles in CRAM.

Putting All Together. Viadotto is designed and evaluated as a tightly integrated system combining simulation models for CRAM-based DPEs, a VHDL-implemented MC, and the Apollo4-based runtime executing compiled application logic. The MCU executes the high-level control logic, including adaptive decisions based on energy conditions, and interacts with the MC through memory-mapped registers and interrupts. The MC offloads data-intensive computations to CRAM, which is modeled using the simulator informed by MTJ parameters. The compiler and runtime coordinate tile allocation and parallelism, while energy behavior is evaluated across the entire platform, including sensing, computation, and communication. This integration ensures that energy and performance results reflect realistic system-level operation.

5.2 Baselines and Benchmarks

MOUSE Simulation. We parameterize MOUSE by the values reported in the article [73]. The programming model of MOUSE is radically different from the one proposed in our work. The program binary stays in CRAM and obeys a dedicated instruction set architecture. The MC of MOUSE is responsible for reading instructions from the CRAM instruction tiles and decoding them into appropriate signals to drive the CRAM operations. However, while implementing MOUSE, we observed significant limitations in exploiting the tile and column parallelism. More specifically, MOUSE applies the bulk addressing technique [78] for the columns within a tile while keeping the instruction formats explicitly specifying a single tile and executing instructions sequentially. Moreover, only five columns can be active at a time since a single instruction can specify up to five column addresses [73, see Sec. IV.B]. Therefore, we follow the design of the extended version of MOUSE [74], where the authors have spread the bulk addressing to tiles with the possibility of activating all the available tiles (see Section 3.4.2) and apply bit mask to activate multiple columns simultaneously (see Section 3.3). However, the addressing tile groups are static and must be defined at the design stage [78], which limits the system's flexibility and scalability. Another crucial point to mention about MOUSE implementation is that its authors shift the responsibility of laying out the data and computation flow in CRAM onto programmers. This approach can lead



(a) Offload cost. (b) Saved time.

Fig. 6. High-level evaluation of the data offload overhead.

to inefficient design decisions we encounter. More specifically, one option to organize the flow in MOUSE is to dedicate different tiles to different functionalities. For example, some entire tiles are reserved for input data, some for only computation, and some for output data. This layout allows for uploading the whole CNN model to CRAM at once. However, given a single data buffer in the MC [73], data transfer between tiles cannot be performed for different input data simultaneously, which serializes execution and degrades the overall performance, nullifying the benefits of parallelism. In contrast, Viadotto avoids inter-tile communications and explicitly splits each tile into dedicated interconnected sections, preventing programmers from misleading. For the fairness of the comparison, we implement the MOUSE's CRAM identically to ours in Viadotto.

PiMCo Simulation. Compared to MOUSE, PiMCo proposes a higher-level programming model and comprises an MCU. However, the programming model has no compiler support and remains burdensome, forcing programmers to manually allocate data in CRAM and keep track of enabled and available DPEs. For the CRAM part, we use the same parameters as for MOUSE, while the layout has a negligible difference compared to Viadotto. For the MCU part, we also use Apollo4 running at a lower frequency, because, like MOUSE, PiMCO introduces no pipelining for the MC operations. To simulate the MC, we follow Section 3.2 in [7].

Benchmarks. We implement an 8-bit-quantized CNN model for two datasets: MNIST [27] (handwritten digits recognition) containing 28×28-pixel gray-scale images of handwritten digits; and ESC-10 [69] (environmental sound classification) containing 5 s audio clips sampled at 44.1 KHz. For ESC-10, we use the 1s audio downsampled five times and convert it to a spectrogram, presented as a 32×32-pixel RGB image, before feeding it to the CNN model. The model structure is as follows: 3 convolutional layers with ReLU and MaxPool functions and 1 fully connected layer followed by the Softmax of 10 neurons.

5.3 Internal vs. External Data Movement

Viadotto motivates developers to offload data-intensive computation to highly parallelizable CRAM, which leads to additional data movement and might seem unreasonable as compared to the execution of the same computation by using a CPU rather than CRAM. To assess this overhead, we use the ESC-10 benchmark, which has 55×10^3 parameters that are assumed to be stored in NVM. In Figure 6(a), we compare the time to transfer the parameters from NVM to CPU registers (for CPU-based execution) and from NVM to CRAM (for the execution via Viadotto). For Viadotto, we considered offloading the entire CNN model and several input images (x-axis) to DPEs of CRAM. Similarly, we also considered the data traffic when the same number of images are processed on the CPU. The figure shows that offloading data to CRAM can take several milliseconds of extra time due to the intermediate link (i.e., MC) between NVM and CRAM. However, the offloading

job can be assigned to the DMA, which frees the CPU for other useful tasks. Moreover, offloading model parameters to CRAM is performed only once. Then the same parameters are reused for all subsequent input images (whose size is only 32×32×3 bytes). Conversely, the CPU brings the model parameters to its registers at each inference computation. Figure 6(b) shows the time saved by parallelizing the inference on CRAM tiles, reaching thousands of milliseconds released for the CPU. Given the modest power consumption of MTJ cells, offloading data to CRAM also saves energy.

5.4 Viadotto vs. MOUSE vs. PiMCo

For this comparison, we consider two evaluation targets: (1) only CRAM with MC (i.e., the MCU part is excluded), and (2) the entire platform. For the prior, we assume that MC receives already preprocessed data, therefore, we use the MNIST dataset, whose images can directly be fed to our CNN model. For the latter, we use the environmental sound classification (ESC-10) application, which is aligned with the acute need for animal-borne sensors [31] and importance of animal-surrounding sounds classification [67]. Increasing the throughput of such batteryless applications by enabling architectural adaptation can help in catching a broader range of sounds, given the dynamism of animals. The application benefits from multiple levels of parallelism: data- and operation-level parallelism on CRAM (see Section 3.1); and task-level parallelism where we generate the next spectrogram on the MCU while the previous one is being inferred on CRAM. Unlike prior art, our platform energy consumption includes the energy cost of sensing audio with MAX4466 microphone [1], transferring inference results with BG22 BLE tag [51], and peripheral setup at each power restoration.

The MC is the main driver for CRAM. According to the ISE [84], the best case achievable clock cycle period for our MC is about 9 ns, which is 111MHz, with a power consumption of 0.46 mW, while the maximum frequency for modern MOUSE is only 30.3 MHz, as a result of non-pipelined execution. Since the authors of MOUSE give no details on their MC implementation, we use our MC, lowering the operating frequency 3× while keeping the energy consumption at the same level. This assumption considers the increased power consumption of our pipelined MC solution, where each clock-cycle fetch, decode, and execute are performed. We apply the same 30.3 MHz, frequency to PiMCo since no pipelining support and frequency improvement are reported in [7]. PiMCo proposes to implement the MC based on a low-power 8-bit MCU, but even ATmega328P requires an order of magnitude more power than Viadotto's MC. Thus, for fairness, we extrapolate the power of PiMCo's MC to one order of magnitude less, assuming that ATmega328P would require 10× less to power the operations necessary for the MC functionality. A traditional MCU is an integral part of Viadotto, enabling architectural adaptation and a conventional programming model with compilation support. Conversely, MOUSE does not feature an MCU and completely shifts the responsibility for hard-coded low-level programming to the developer. PiMCo, in turn, comprises an MCU but does not benefit from a compiler. Like in Viadotto, we set the communication between the MCU and MC in PiMCo to interrupt-based to use the MCU simultaneously with CRAM.

(1-) CRAM Comparison. For CRAM evaluation, we excluded the Viadotto's adaptation (since the MCU is not included in this setup) and set four CRAM configurations with a fixed number of active tiles. We perform this comparison only against the CRAM pioneer, MOUSE. We selected two edge cases, 1- and 32-tile CRAM, for each system: MOUSE 1 (M1), MOUSE 32 (M32), Viadotto 1 (V1), and Viadotto 32 (V32). As seen in Figure 7, we ran 10-minute experiments under four types of power traces, including constant power, and counted the number of images processed. Constant power introduces no power failures, so Viadotto benefits from the faster MC in terms of performance. On the other hand, with the pipelined MC, the energy consumption of CRAM increases over 2×. However, these parameters are neutralized under transient power sources. For example, during a car ride, ambient power is still strong to maintain uninterrupted M1 and V1, but M32 and V32

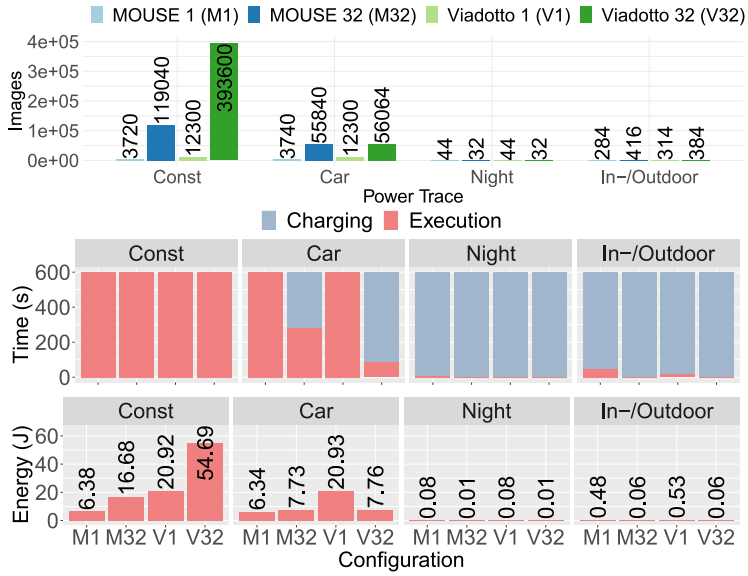


Fig. 7. Performance comparison of the Viadotto's CRAM against that of MOUSE. Results for the MNIST benchmark.

suffer multiple power failures, which almost equalize the energy and performance of M32 and V32. The difference is only in the number of power failures and the time spent on charging. Another key point of this experiment is showing that under different power conditions, different CRAM configurations outperform others, e.g., even if M1 and V1 encounter no power failures during a car ride, M32 and V32 with multiple interruptions perform much better. With the night power, the single-tile setup is 1.4× faster because the multi-tile setup spends more time charging. With the mixed power (in-/outdoor activity), we have average results between high and low power. However, increasing the system's power consumption when input power allows and decreasing it when input power is modest would significantly reduce the number of failures.

(2-) Platform Comparison. For this evaluation, we enable the architectural adaptation of Viadotto running at low (30 MHz, VL) and high (111 MHz, VH) frequencies and compare them with PiMCo implementations Pi1 and Pi32, which are non-adaptive. For making adaptation decisions in Viadotto, we estimate the power consumption of V1, V2, V4, V8, V16, and V32 and deliver this information through the configuration file. In decision making, we compare the estimated P_{amb} (see Section 3.3) with each configuration, starting from V32. For the first match of $P_{amb} \geq P_{exec}$, we select the configuration holding the condition or, if available, the next higher configuration. This approach is based on our observations that showed that in most cases, even if, for example, V8 holds the condition, V16 performs better compensating power failures by speedup. However, V32 cannot outperform V8 in this case, given the significantly increased number of power failures. Figure 8 shows that MOUSE (M) cannot compete with Viadotto because the interface between MCU and MOUSE is not provided by design. The figure also shows that with the MCU and under constant power, VL consumes an insignificant amount of time for decision-making and performs almost the same as Pi32. However, the superior frequency of VH allows Viadotto to outperform Pi1 and Pi32. Note that under this power condition, adaptive Viadotto always selects the V32 configuration. For the energy-harvesting scenarios, both Viadotto setups VL and VH adapt to corresponding configurations, outperforming Pi1 and Pi32 by up to 18.9× (Car) and up to 1.48× (In-/Outdoor),

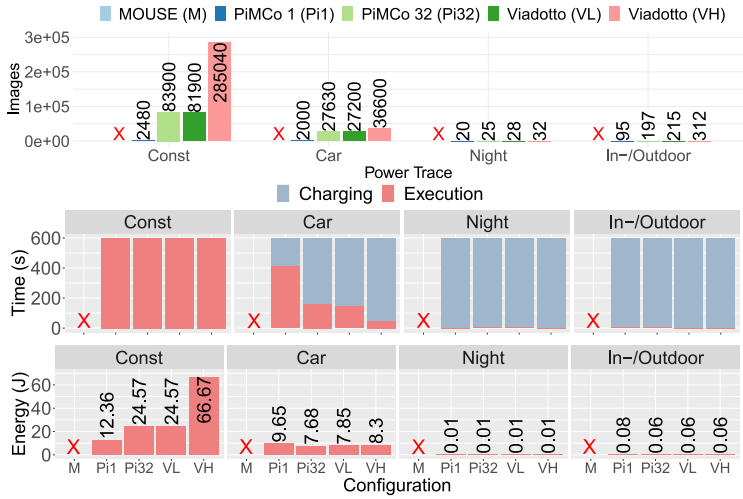


Fig. 8. Performance comparison of the Viadotto entire platform against MOUSE and PiMCo. Results for the ESC-10 benchmark.

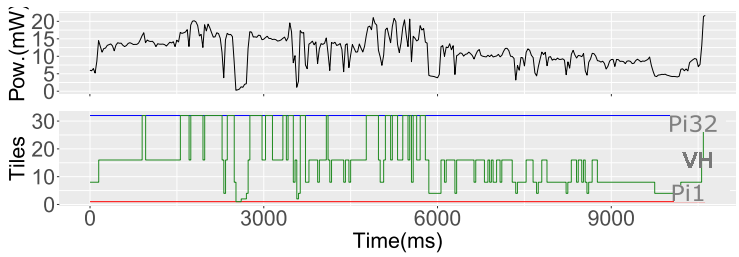


Fig. 9. Viadotto adaptation to a given power trace.

respectively. In Figure 9, we demonstrate the track of the Viadotto adaptation decisions along the car ride power trace. As seen, Viadotto adjusts the number of active CRAM tiles according to the ambient power at runtime, while the degree of the tile-level parallelism of MOUSE is fixed at the design stage by embedding the code into the CRAM substrates.

Note that the three-voltage-levels JIT backups (see Section 3.3) eliminate the re-execution overhead for all scenarios, while only a few checkpoints and reboots happen at night when input power is less than that of the sleep mode. Moreover, after the system voltage reaches V_{on}, Viadotto performs a cold start, which includes MCU boot-up, reinitialization of runtime state, and resumption of CRAM controller operations. Based on Apollo4's datasheet [10] and measured startup behavior, the MCU boot and library reinitialization take approximately 1.7 ms and consume roughly 7.1 µJ. This recovery includes setting up peripheral states and restoring Viadotto's software stack. Since CRAM operations are failure-atomic and nonvolatile, no memory-level state restoration is needed, which significantly reduces overhead compared to volatile systems. The MC waits for the MCU to send macroinstructions again and resumes operation without explicit reboot logic. Overall, Viadotto's post-power-up recovery is fast and energy-efficient, allowing it to maintain forward progress with minimal interruption.

(3-) Area Overhead. Viadotto occupies 88 mm^2 space contributed by the MCU (22 mm^2 [10]), the MC (64 mm^2 [85]), and CRAM (2 mm^2 [30]). For MOUSE, we grasp the largest area reported

Table 3. Comparison of the Viadotto Parameters Against MOUSE and PiMCo

Parameter	Viadotto	MOUSE	PiMCo
Operating frequency, MHz	111 ✓	30.3 ✗	30.3 ✗
Area overhead, %	73 ✓	0	103 ✗
Programming model	High-level ✓	Gate-level ✗	High-level ✓
Compiler support	Yes ✓	No ✗	No ✗
MCU-integrated	Yes ✓	No ✗	Yes ✓
Architectural adaptation	Yes ✓	No ✗	No ✗

in [73, Tab.IV], which is 51 mm^2. Note that the MCU, which occupies 25% of Viadotto area, is a must component of any existing intermittent system. Therefore, integrating Viadotto adds only 66 mm^2 to the total area consumption. Note that although the MC consumes only 2% of flip-flops and look-up tables of the FPGA, we pessimistically consider the entire area of the FPGA since the design requires almost 85% of input-output blocks. For PiMCo, we include 22 mm^2 of the MCU, 81 mm^2 of MC [11], and 2 mm^2 of CRAM. The comparison parameters are summarized in Table 3.

5.5 Viadotto vs. CGRA

Coarse-Grained Reconfigurable Array (**CGRA**) is an attractive combination of the reconfigurability of FPGAs and the custom design of ASICs [54, 70]. CGRAs offer hardware-like efficiency and software-like programmability. Viadotto also benefits from software-based reconfigurability of underlying hardware, but unlike the state-of-the-art CGRA compiler, RipTide [35], Viadotto's compiler leverages existing ISAs and hardware resources without requiring any modifications or additional support. More importantly, CGRAs are not designed for intermittent computing, preventing us from one-to-one comparison against Viadotto. Only a few studies [24, 34, 45, 68] have proposed the design of ultra-low-power CGRAs. Arguably the most energy-efficient (<1 mW) CGRA-based system, Snafu-Arch [34], comprises a scalar RISC-V core, CGRA fabric, and banked SRAM memory. To save energy, Snafu-Arch features a bufferless Network-on-Chip connecting **processing elements** (**PEs**), which implement traditional ALU and keep intermediate results in their internal memory. Moreover, each PE contains a configuration cache storing currently used configurations of the PE. This memory hierarchy is totally volatile, which makes Snafu-Arch suffer from the non-termination problem under frequent power failures. Even if all the memory elements are replaced by nonvolatile versions (which increases energy and timing overhead), the hierarchy requires nontrivial solutions [22, 40, 93, 96] to maintain memory consistency under intermittent power. Besides, the integration of CGRA fabric into existing intermittent systems inevitably requires an external PE array with auxiliary circuits and connections. Conversely, CRAM can be mounted in place of nonvolatile memory with minimal modifications (additional logic drivers and lines). In essence, the design of Viadotto is orthogonal to CGRAs and, unlike CGRAs, mainly addresses batteryless devices.

6 Limitations and Future Work

In this section, we discuss the limitations of Viadotto and future research directions.

Practicality. Currently, the evaluation of such emerging platforms as Viadotto is limited to custom system-level simulations since there is no manufactured CRAM prototype in the market. While flexible, fast, and less resource-demanding, this approach perfectly works for early-stage system evaluation but sacrifices implementation precision necessary for detailed platform exploration. We envision several ways to enhance the practicality of the evaluation setup in future. First, cycle-accurate and validated full-system simulation (like gem5 [57]) would improve the evaluation

granularity, allowing for more detailed system exploration. Second, mimicking the CRAM behavior based on traditional CMOS technology, using FPGA or ASIC, would give the opportunity to interface CRAM with real-life MCU, MC, sensors, and other periphery. Finlay, we expect memory manufacturers and big research groups in the near future to release a PCB-based platform comprising the CRAM chip with all the necessary connections to assemble the full Viadotto system, analogous to NeuRRAM [81].

Programmable CIM. Programmable in-memory computing has been a desirable feature since the invention of the CIM paradigm [33]. Researchers are still trying to invent an optimal solution, targeting DRAM-featured devices mostly with complex software stack and memory hierarchy [5, 32, 87]. Viadotto targets resource-constrained devices with no operating system support and caching mechanism, providing high-level programmability to emerging in-memory computing on transiently powered devices for the first time.

Scalability and Portability of the System. The fixed design of the MC makes Viadotto less flexible in terms of ISA extension. While the software library is easy to augment with new instructions, the MC parts require effort partial redesign. However, by covering most macroinstructions widely used in modern applications, we make Viadotto versatile since the configuration instructions are reusable. Furthermore, the content of CM does not depend on the MCU part, the selection of which is limited by the application needs. Although the MCU-to-CRAM interface in Viadotto is universal, the system designer should care about potential bottlenecks, which can be caused by any discrepancy in their operating frequencies.

Scalability of Code Versioning. Viadotto currently stores six precompiled versions of each CRAM routine, corresponding to 1, 2, 4, 8, 16, and 32 tile configurations. Each version includes tile-specific operand alignment, loop unrolling, and configuration logic. In our current design, each CRAM routine occupies 0.8–1.2 KB per version, resulting in a total footprint of around 6 KB for each operation type. While this overhead is acceptable for platforms with 128–256 KB of flash (like Apollo4), it may become limiting as memory size or tile count increases. If the system were scaled to 64 or 128 tiles, naive versioning would require significant flash or CRAM storage. In such cases, strategies like partial code synthesis at runtime [20], parameterized CRAM kernels [44], or runtime-controlled loops [90] could reduce footprint without sacrificing adaptability. These strategies would still rely on compiler support: instead of statically emitting multiple unrolled CRAM kernel versions, the Viadotto compiler would generate parameterized code templates with runtime-controlled bounds or loop bodies. This allows the system to retain its high-level programming model while significantly reducing code footprint. We identify this as a future optimization path as Viadotto targets larger-scale CIM systems.

Advanced Decision-making Strategy. While Viadotto employs a lightweight voltage-threshold-based mechanism to estimate ambient power and adjust tile-level parallelism, we acknowledge that more advanced power prediction strategies could further optimize adaptation decisions. Techniques such as historical energy profiling, machine learning-based estimators, or hybrid capacitor charge models could potentially improve responsiveness to rapid energy fluctuations. Additionally, the decision-making for adaptation can be enhanced by integrating an adaptive runtime monitoring, considering properties of intermittent computing, such as ensuring progress despite power failures and avoiding stale operations from charging delays [89]. However, incorporating such mechanisms would require nontrivial additional computation, memory overhead, and sensing infrastructure, which are beyond the intended lightweight and platform-agnostic design of Viadotto. We identify this as a promising direction for future work, particularly in contexts where prediction accuracy could meaningfully improve task scheduling and throughput.

Other Checkpoint Policies. Given the diversity of checkpoint strategies, Viadotto can adopt any of them by certain extensions. In task-based intermittent systems, the decision-making process

needs to be integrated into existing tasks or allocated to a separate task and interfaced with others. Moreover, the energy budget of existing tasks must be revised due to the extra load introduced by CRAM. For statically placed checkpoints, the developer needs to reallocate the backup points concerning the CRAM energy consumption and the interrupt exchange between the MCU and the MC. However, the complexity of these extensions even increases with the architectural adaptation, which affects the task's and checkpoints region's energy budget at runtime.

Advanced Compiler Support. Another promising future direction is the development of a more sophisticated compiler. Recent study [29] has shown that loading computation closer to memory is not always beneficial, e.g., compute-intensive applications might better match multicore CPUs. Advanced compiler techniques could support developers in assigning application subtasks to the most performant computational units of the heterogeneous system (MCU vs. CRAM), considering application specifications, platform restrictions, and the unique properties of intermittent computing.

Architectural Gains vs. Technology Baseline. While Viadotto uses simulation-based evaluation for the MC and CRAM logic, all measurements related to the MCU (Apollo4) are grounded in its publicly available datasheet and documentation. Importantly, our reported improvements in energy and throughput are not solely a result of newer MCU technology. The primary gains stem from architectural features, namely, the offloading of memory-bound operations to failure-atomic CRAM, the pipelined MC design, and adaptive tile-level execution. For fair comparison, all baselines (e.g., MOUSE and PiMCo) are normalized to match similar technology nodes where possible. Moreover, Viadotto's improvements persist even when adjusting for the MCU energy profile. We acknowledge that full-system hardware integration may introduce additional minor overheads (e.g., I/O buffering or interconnect latency), but our architectural contributions remain valid and essential to the observed improvements.

7 Conclusion

In this article, we presented Viadotto, a new programmable CRAM-based in-memory computing platform that tolerates frequent power failures. Viadotto performs adaptive in-memory operations highly efficiently and provides the programmability and versatility for intermittent CIM operations. Viadotto stands on three pillars: a software library with compiler support that exposes interfaces for high-level operations to be performed in-memory, an adaptive CRAM substrate, and a faster MC that facilitates data and computation mapping for CIM. According to our evaluations under intermittent conditions, Viadotto achieves a 48% performance improvement over the state-of-the-art intermittent CIM system.

References

[1] Adafruit. 2023. *Electret Microphone Amplifier - MAX4466 with Adjustable Gain*. Technical Report. Adafruit. Retrieved from https://www.adafruit.com/product/1063

[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, et al. 2020. Battery-less zero-maintenance embedded sensing at the mithræum of circus maximus. In *Proceedings of the 18th Conference on Embedded Networked Sensor Systems*. 368–381.

[3] Saad Ahmed, Bashima Islam, Kasim Sinan Yildirim, Marco Zimmerling, Przemysław Pawełczak, Muhammad Hamad Alizai, Brandon Lucia, Luca Mottola, Jacob Sorber, and Josiah Hester. 2024. The internet of batteryless things. *Communications of the ACM* 67, 3 (2024), 64–73.

[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 *Proceedings of the EWSN*. 97–107.

[5] Junwhan Ahn, Sungjoo Yoo, Onur Mutlu, and Kiyoung Choi. 2015. PIM-enabled instructions: A low-overhead, locality-aware processing-in-memory architecture. In *Proceedings of the 2015 ACM/IEEE 42nd Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 336–348.

[6] 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.
[7] Khakim Akhunov and Kasım Sinan Yıldırım. 2023. CRAM-based acceleration for intermittent computing of parallelizable tasks. *IEEE Transactions on Emerging Topics in Computing* 12, 1 (2023), 48–59.
[8] Khakim Akhunov and Kasim Sinan Yildirim. 2023. LUTIC: A CRAM-based architecture for power failure resilient in-memory computing. In *Proceedings of the 2023 26th International Symposium on Design and Diagnostics of Electronic Circuits and Systems (DDECS)*. IEEE, 69–72.
[9] Khakim Akhunov, Eren Yildiz, and Kasim Sinan Yildirim. 2023. Enabling efficient intermittent computing on brand new microcontrollers via tracking programmable voltage thresholds. In *Proceedings of the 11th International Workshop on Energy Harvesting and Energy-Neutral Sensing Systems*. 16–22.
[10] ambiq. 2023. *Apollo4 Datasheet*. Technical Report. ambiq.Retrieved from https://ambiq.com/apollo4/
[11] Atmel. 2015. *ATmega328P*. Technical Report. Atmel. Retrieved from https://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-7810-Automotive-Microcontrollers-ATmega328P_Datasheet.pdf
[12] Abu Bakar, Rishabh Goel, Jasper de Winkel, Jason Huang, Saad Ahmed, Bashima Islam, Przemysław Pawełczak, Kasım Sinan Yıldırım, and Josiah Hester. 2022. Protean: An energy-efficient and heterogeneous platform for adaptive and hardware-accelerated battery-free computing. In *Proceedings of the 20th ACM Conference on Embedded Networked Sensor Systems*. 207–221.
[13] 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.
[14] Abu Bakar, Tousif Rahman, Rishad Shafik, Fahim Kawsar, and Alessandro Montanari. 2022. Adaptive intelligence for batteryless sensors using software-accelerated tsetlin machines. In *Proceedings of the 20th ACM Conference on Embedded Networked Sensor Systems*. 236–249.
[15] 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.
[16] Domenico Balsamo, Anup Das, Alex S. Weddell, Davide Brunelli, Bashir M. Al-Hashimi, Geoff V. Merrett, and Luca Benini. 2016. Graceful performance modulation for power-neutral transient computing systems. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 35, 5 (2016), 738–749.
[17] Writam Banerjee. 2020. Challenges and applications of emerging nonvolatile memory devices. *Electronics* 9, 6 (2020), 1029.
[18] Jaeseok Choi, Hyunwoo Joe, Changhee Jung, and Jongouk Choi. 2024. Defending against emi attacks on just-in-time checkpoint for resilient intermittent systems. In *Proceedings of the 2024 57th IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 121–135.
[19] Jongouk Choi, Hyunwoo Joe, Yongjoo Kim, and Changhee Jung. 2019. Achieving stagnation-free intermittent computation with boundary-free adaptive execution. In *Proceedings of the 2019 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*. IEEE, 331–344.
[20] Jongouk Choi, Larry Kittinger, Qingrui Liu, and Changhee Jung. 2022. Compiler-directed high-performance intermittent computation with power failure immunity. In *Proceedings of the 2022 IEEE 28th Real-Time and Embedded Technology and Applications Symposium (RTAS)*. IEEE, 40–54.
[21] Jongouk Choi, Qingrui Liu, and Changhee Jung. 2019. CoSpec: Compiler directed speculative intermittent computation. In *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*. 399–412.
[22] Jongouk Choi, Jianping Zeng, Dongyoon Lee, Changwoo Min, and Changhee Jung. 2023. Write-light cache for energy harvesting systems. In *Proceedings of the 50th Annual International Symposium on Computer Architecture*. 1–13.
[23] 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.
[24] Satyajit Das, Davide Rossi, Kevin J. M. Martin, Philippe Coussy, and Luca Benini. 2017. A 142 mops/mw integrated programmable array accelerator for smart visual processing. In *Proceedings of the 2017 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE, 1–4.
[25] Marc A. De Kruijf, Karthikeyan Sankaralingam, and Somesh Jha. 2012. Static analysis and compiler design for idempotent processing. In *Proceedings of the 33rd ACM SIGPLAN Conference on Programming Language Design and Implementation*. 475–486.
[26] Jasper De Winkel, Vito Kortbeek, Josiah Hester, and Przemysław Pawełczak. 2020. Battery-free game boy. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies* 4, 3 (2020), 1–34.
[27] Li Deng. 2012. The mnist database of handwritten digit images for machine learning research [best of the web]. *IEEE Signal Processing Magazine* 29, 6 (2012), 141–142.

[28] Harsh Desai, Matteo Nardello, Davide Brunelli, and Brandon Lucia. 2022. Camaroptera: A long-range image sensor with local inference for remote sensing applications. *ACM Transactions on Embedded Computing Systems* 21, 3 (2022), 1–25.

[29] Alexandar Devic, Siddhartha Balakrishna Rai, Anand Sivasubramaniam, Ameen Akel, Sean Eilert, and Justin Eno. 2022. To pim or not for emerging general purpose processing in ddr memory systems. In *Proceedings of the 49th Annual International Symposium on Computer Architecture*. 231–244.

[30] Xiangyu Dong, Cong Xu, Yuan Xie, and Norman P. Jouppi. 2012. Nvsim: A circuit-level performance, energy, and area model for emerging nonvolatile memory. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 31, 7 (2012), 994–1007.

[31] Diego Ellis-Soto, Martin Wikelski, and Walter Jetz. 2023. Animal-borne sensors as a biologically informed lens on a changing climate. *Nature Climate Change* 13, 10 (2023), 1042–1054.

[32] João Dinis Ferreira, Gabriel Falcão, Juan Gómez-Luna, Mohammed Alser, Lois Orosa, Mohammad Sadrosadati, Jeremie S. Kim, Geraldo F. Oliveira, Taha Shahroodi, Anant Nori, and Onur Mutlu. 2021. pLUTo: In-DRAM lookup tables to enable massively parallel general-purpose computation. arXiv:2104.07699. Retrieved from https://arxiv.org/abs/2104.07699

[33] Saugata Ghose, Amirali Boroumand, Jeremie S. Kim, Juan Gómez-Luna, and Onur Mutlu. 2019. Processing-in-memory: A workload-driven perspective. *IBM Journal of Research and Development* 63, 6 (2019), 3–1.

[34] 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 *Proceedings of the 2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 1027–1040.

[35] Graham Gobieski, Souradip Ghosh, Marijn Heule, Todd Mowry, Tony Nowatzki, Nathan Beckmann, and Brandon Lucia. 2022. A programmable, energy-minimal dataflow compiler and architecture. In *Proceedings of the 2022 55th IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 546–564.

[36] Graham Gobieski, Brandon Lucia, and Nathan Beckmann. 2019. Intelligence beyond the edge: Inference on intermittent embedded systems. In *Proceedings of the 24th International Conference on Architectural Support for Programming Languages and Operating Systems*. 199–213.

[37] 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.

[38] Xinge Guo, Luwei Wang, and Chengkuo Lee. 2025. Battery-less outdoor farming IOTll sensing system using multifunctional hydrogel enabled direct-current powering and self-powered leaf monitoring capability. In *Proceedings of the 2025 IEEE 38th International Conference on Micro Electro Mechanical Systems (MEMS)*. IEEE, 315–318.

[39] Zongxia Guo, Jialiang Yin, Yue Bai, Daoqian Zhu, Kewen Shi, Gefei Wang, Kaihua Cao, and Weisheng Zhao. 2021. Spintronics for energy-efficient computing: An overview and outlook. *Proceedings of the IEEE* 109, 8 (2021), 1398–1417.

[40] Noureldin Hassan, Byounguk Min, Changhee Jung, Yan Solihin, and Jongouk Choi. 2025. WarmCache: Exploiting STT-RAM cache for low-power intermittent systems. In *Proceedings of the 52nd Annual International Symposium on Computer Architecture*. 586–600.

[41] 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.

[42] Texas Instruments. 2022. *FRAM - New Generation of Non-Volatile Memory*. Technical Report. Retrieved from https://www.ti.com/lit/ml/szzt014a/szzt014a.pdf

[43] Bashima Islam and Shahriar Nirjon. 2020. Scheduling computational and energy harvesting tasks in deadline-aware intermittent systems. In *Proceedings of the 2020 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*. IEEE, 95–109.

[44] Bashima Islam and Shahriar Nirjon. 2020. Zygarde: Time-sensitive on-device deep inference and adaptation on intermittently-powered systems. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies* 4, 3, 1–29.

[45] Changmoo Kim, Mookyoung Chung, Yeongon Cho, Mario Konijnenburg, Soojung Ryu, and Jeongwook Kim. 2014. Ulp-srp: Ultra low-power samsung reconfigurable processor for biomedical applications. *ACM Transactions on Reconfigurable Technology and Systems* 7, 3 (2014), 1–15.

[46] Hongjune Kim, Jianping Zeng, Qingrui Liu, Mohammad Abdel-Majeed, Jaejin Lee, and Changhee Jung. 2020. Compiler-directed soft error resilience for lightweight GPU register file protection. In *Proceedings of the 41st ACM SIGPLAN Conference on Programming Language Design and Implementation*. 989–1004.

[47] 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.

[48] 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 25th International Conference on Architectural Support for Programming Languages and Operating Systems*. 85–99.

[49] David Kotz, Tristan Henderson, Ilya Abyzov, and Jihwang Yeo. CRAWDAD dataset dartmouth/campus (v. 2009-09-09). (n. d.). Retrieved from https://crawdad.org/columbia/enhants/20110407

[50] Roberto La Rosa, Catherine Dehollain, Mario Costanza, Angelo Speciale, Fabio Viola, and Patrizia Livreri. 2022. A battery-free wireless smart sensor platform with bluetooth low energy connectivity for smart agriculture. In *Proceedings of the 2022 IEEE 21st Mediterranean Electrotechnical Conference (MELECON)*. IEEE, 554–558.

[51] Silicon Labs. 2023. *EFR32BG22 Thunderboard Kit*. Technical Report. Silicon Labs. Retrieved from https://www.silabs.com/development-tools/thunderboard/thunderboard-bg22-kit?tab=overview

[52] Chris Lattner and Vikram Adve. 2004. LLVM: A compilation framework for lifelong program analysis and transformation. In *Proceedings of the International Symposium on Code Generation and Optimization (CGO '04)*. IEEE Computer Society, Washington, DC, USA, 75–86.

[53] 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.

[54] Leibo Liu, Jianfeng Zhu, Zhaoshi Li, Yanan Lu, Yangdong Deng, Jie Han, Shouyi Yin, and Shaojun Wei. 2019. A survey of coarse-grained reconfigurable architecture and design: Taxonomy, challenges, and applications. *ACM Computing Surveys* 52, 6 (2019), 1–39.

[55] Qingrui Liu, Joseph Izraelevitz, Se Kwon Lee, Michael L. Scott, Sam H. Noh, and Changhee Jung. 2018. iDO: Compiler-directed failure atomicity for nonvolatile memory. In *Proceedings of the 2018 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 258–270.

[56] Qingrui Liu, Changhee Jung, Dongyoon Lee, and Devesh Tiwari. 2016. Compiler-directed lightweight checkpointing for fine-grained guaranteed soft error recovery. In *SC'16: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 228–239.

[57] Jason Lowe-Power, Abdul Mutaal Ahmad, Ayaz Akram, Mohammad Alian, Rico Amslinger, Matteo Andreozzi, Adrià Armejach, Nils Asmussen, Brad Beckmann, Srikant Bharadwaj, et al. 2020. The gem5 simulator: Version 20.0+. arXiv:2007.03152. Retrieved from https://arxiv.org/abs/2007.03152

[58] Brandon Lucia, Brad Denby, Zachary Manchester, Harsh Desai, Emily Ruppel, and Alexei Colin. 2021. Computational nanosatellite constellations: Opportunities and challenges. *GetMobile: Mobile Computing and Communications* 25, 1 (2021), 16–23.

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

[60] Giedrius Lukosevicius, Alberto Rodriguez Arreola, and Alex S. Weddell. 2017. Using sleep states to maximize the active time of transient computing systems. In *Proceedings of the 5th ACM International Workshop on Energy Harvesting and Energy-Neutral Sensing Systems*. 31–36.

[61] Kaisheng Ma, Xueqing Li, Srivatsa Rangachar Srinivasa, Yongpan Liu, John Sampson, Yuan Xie, and Vijaykrishnan Narayanan. 2017. Spendthrift: Machine learning based resource and frequency scaling for ambient energy harvesting nonvolatile processors. In *Proceedings of the 2017 22nd Asia and South Pacific Design Automation Conference (ASP-DAC)*. IEEE, 678–683.

[62] 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.

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

[64] 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.

[65] 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.

[66] Onur Mutlu, Saugata Ghose, Juan Gómez-Luna, and Rachata Ausavarungnirun. 2019. Processing data where it makes sense: Enabling in-memory computation. *Microprocessors and Microsystems* 67, C (2019), 28–41.

[67] Takuji Noda, Takuya Koizumi, Naoto Yukitake, Daisuke Yamamoto, Tetsuro Nakaizumi, Kotaro Tanaka, Junichi Okuyama, Kotaro Ichikawa, and Takeshi Hara. 2024. Animal-borne soundscape logger as a system for edge classification of sound sources and data transmission for monitoring near-real-time underwater soundscape. *Scientific Reports* 14, 1 (2024), 6394.

[68] Nobuaki Ozaki, Yoshihiro Yasuda, Mai Izawa, Yoshiki Saito, Daisuke Ikebuchi, Hideharu Amano, Hiroshi Nakamura, Kimiyoshi Usami, Mitaro Namiki, and Masaaki Kondo. 2011. Cool mega-arrays: Ultralow-power reconfigurable accelerator chips. *IEEE Micro* 31, 6 (2011), 6–18.

[69] Karol J. Piczak. 2015. ESC: Dataset for environmental sound classification. In *Proceedings of the 23rd ACM international conference on Multimedia*. 1015–1018.

[70] Artur Podobas, Kentaro Sano, and Satoshi Matsuoka. 2020. A survey on coarse-grained reconfigurable architectures from a performance perspective. *IEEE Access* 8 (2020), 146719–146743.

[71] Keni Qiu, Nicholas Jao, Mengying Zhao, Cyan Subhra Mishra, Gulsum Gudukbay, Sethu Jose, Jack Sampson, Mahmut Taylan Kandemir, and Vijaykrishnan Narayanan. 2020. ResiRCA: A resilient energy harvesting ReRAM crossbar-based accelerator for intelligent embedded processors. In *Proceedings of the 2020 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 315–327.

[72] Benjamin Ransford and Brandon Lucia. 2014. Nonvolatile memory is a broken time machine. In *Proceedings of the Workshop on Memory Systems Performance and Correctness*. 1–3.

[73] Salonik Resch, S. Karen Khatamifard, Zamshed I. Chowdhury, Masoud Zabihi, Zhengyang Zhao, Husrev Cilasun, Jian-Ping Wang, Sachin S. Sapatnekar, and Ulya R. Karpuzcu. 2020. MOUSE: Inference in non-volatile memory for energy harvesting applications. In *Proceedings of the 2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 400–414.

[74] Salonik Resch, S. Karen Khatamifard, Zamshed I. Chowdhury, Masoud Zabihi, Zhengyang Zhao, Husrev Cilasun, Jian-Ping Wang, Sachin S. Sapatnekar, and Ulya R. Karpuzcu. 2022. Energy-efficient and reliable inference in nonvolatile memory under extreme operating conditions. *ACM Transactions on Embedded Computing Systems* 21, 5 (2022), 1–36.

[75] Simone Ruffini, Luca Caronti, Kasım Sinan Yıldırım, and Davide Brunelli. 2022. NORM: An FPGA-based non-volatile memory emulation framework for intermittent computing. *ACM Journal on Emerging Technologies in Computing Systems* 18, 4 (2022), 1–18.

[76] 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.

[77] Abdullah Ash Saki, Sung Hao Lin, Mahabubul Alam, Sandeep Krishna Thirumala, Sumeet Kumar Gupta, and Swaroop Ghosh. 2019. A family of compact non-volatile flip-flops with ferroelectric FET. *IEEE Transactions on Circuits and Systems I: Regular Papers* 66, 11 (2019), 4219–4229.

[78] Vivek Seshadri, Donghyuk Lee, Thomas Mullins, Hasan Hassan, Amirali Boroumand, Jeremie Kim, Michael A. Kozuch, Onur Mutlu, Phillip B. Gibbons, and Todd C. Mowry. 2017. Ambit: In-memory accelerator for bulk bitwise operations using commodity DRAM technology. In *Proceedings of the 2017 50th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 273–287.

[79] Weining Song, Stefanos Kaxiras, Luca Mottola, Thiemo Voigt, and Yuan Yao. 2023. Silent stores in the battery-less internet of things: A good idea?. In *Proceedings of the EWSN*. 40–45.

[80] Thang Viet Tran, Nam Trung Dang, and Wan-Young Chung. 2017. Battery-free smart-sensor system for real-time indoor air quality monitoring. *Sensors and Actuators B: Chemical* 248 (2017), 930–939.

[81] Weier Wan, Rajkumar Kubendran, Clemens Schaefer, Sukru Burc Eryilmaz, Wenqiang Zhang, Dabin Wu, Stephen Deiss, Priyanka Raina, He Qian, Bin Gao, et al. 2022. A compute-in-memory chip based on resistive random-access memory. *Nature* 608, 7923 (2022), 504–512.

[82] Jian-Ping Wang and Jonathan D. Harms. 2015. General structure for computational random access memory (CRAM). (dec 2015). US Patent 9,224,447.

[83] Yilun Wu, Byounguk Min, Mohannad Ismail, Wenjie Xiong, Changhee Jung, and Dongyoon Lee. 2024. {IntOS}: Persistent embedded operating system and language support for multi-threaded intermittent computing. In *Proceedings of the 18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24)*. 425–443.

[84] Xilinx. 2022. *ISE Design Suite*. Technical Report. Xilinx. Retrieved from https://www.xilinx.com/support/download/index.html/content/xilinx/en/downloadNav/vivado-design-tools/archive-ise.html

[85] Xilinx. 2022. *Spartan-6 FPGA Packaging and Pinouts*. Technical Report. Xilinx. Retrieved from https://docs.amd.com/v/u/en-US/ug385

[86] Xueqing Li, Sumitha George, Yuhua Liang, Kaisheng Ma, Kai Ni, Ahmedullah Aziz, Sumeet Kumar Gupta, John Sampson, Meng-Fan Chang, Yongpan Liu, Huazhong Yang, Suman Datta, and Vijaykrishnan Narayanan. 2018. Lowering area overheads for FeFET-based energy-efficient nonvolatile flip-flops. *IEEE Transactions on Electron Devices* 65, 6 (2018), 2670–2674.

[87] Xu Yang, Yumin Hou, and Hu He. 2019. A processing-in-memory architecture programming paradigm for wireless Internet-of-Things applications. *Sensors* 19, 1 (2019), 140.

[88] 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.

[89] Eren Yildiz, Khakim Akhunov, Lorenzo Antonio Riva, Arda Goknil, Ivan Kurtev, and Kasim Sinan Yildirim. 2024. Adaptable runtime monitoring for intermittent systems. In *Proceedings of the 19th European Conference on Computer Systems*. 1175–1191.

[90] Eren Yıldız, Lijun Chen, and Kasim Sinan Yıldırım. 2022. Immortal Threads: Multithreaded event-driven intermittent computing on {Ultra-Low-Power} microcontrollers. In *Proceedings of the 16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22)*. 339–355.

[91] Shimeng Yu, Xiaoyu Sun, Xiaochen Peng, and Shanshi Huang. 2020. Compute-in-memory with emerging nonvolatile-memories: Challenges and prospects. In *Proceedings of the 2020 IEEE Custom Integrated Circuits Conference (CICC)*. IEEE, 1–4.

[92] Masoud Zabihi, Zamshed Iqbal Chowdhury, Zhengyang Zhao, Ulya R. Karpuzcu, Jian-Ping Wang, and Sachin S. Sapatnekar. 2018. In-memory processing on the spintronic CRAM: From hardware design to application mapping. *IEEE Transactions on Computers* 68, 8 (2018), 1159–1173.

[93] Jianping Zeng, Jongouk Choi, Xinwei Fu, Ajay Paddayuru Shreepathi, Dongyoon Lee, Changwoo Min, and Changhee Jung. 2021. Replaycache: Enabling volatile cachesfor energy harvesting systems. In *Proceedings of the MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture*. 170–182.

[94] Jianping Zeng, Tong Zhang, and Changhee Jung. 2024. Compiler-directed whole-system persistence. In *Proceedings of the 2024 ACM/IEEE 51st Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 961–977.

[95] Yida Zhang and Changhee Jung. 2022. Featherweight soft error resilience for gpus. In *Proceedings of the 2022 55th IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 245–262.

[96] Yuchen Zhou, Jianping Zeng, Jungi Jeong, Jongouk Choi, and Changhee Jung. 2023. Sweepcache: Intermittence-aware cache on the cheap. In *Proceedings of the 56th Annual IEEE/ACM International Symposium on Microarchitecture*. 1059–1074.

Received 31 October 2024; revised 2 June 2025; accepted 22 August 2025