SRC: Sustainable Reactive Computing for Battery-free Edge Intelligence

Sepehr Tabrizchi* , Nedasadat Taheri§, Justin Feng‡, Nader Sehatbakhsh‡, David Z. Pan†, and Arman Roohi*,§
*Department of Electrical and Computer Engineering, University of Illinois Chicago, Chicago, IL, USA
§School of Computing, University of Nebraska-Lincoln, Lincoln, NE, USA
‡Department of Electrical and Computer Engineering, New Jersey Institute of Technology, Newark, NJ, USA
†Department of Electrical and Computer Engineering, University of Texas Austin, Austin
aroohi@uic.edu

2024 IEEE 15th International Green and Sustainable Computing Conference (IGSC) | 979-8-3315-0786-2/24/$31.00 ©2024 IEEE | DOI: 10.1109/IGSC64514.2024.00026

***Abstract*—This paper proposes SRC, a novel framework for efficient and reliable inference on battery-free smart Internet of Things (IoT) devices. SRC supports various configurations that follow reactive configuration while using the innovative state machine and a safe_threshold mechanism to proactively halt operations, reducing store/load operations by up to 75%. It strategically stores essential convolutional neural network (CNN) data (layer, kernel, etc.) to optimize input/output feature map management. This reactive design allows seamless task resumption across power cycles, ensuring continuity in unpredictable energy environments. Experiments show significant gains, with SRC achieving on average $\sim 81.85\%$ reduction in read/write operations and approximately 57.18% improvement in sensing compared to conventional reactive methods based on the intermittent Energy Trace 1.**

I. Introduction

The IoT represents a paradigm shift from a cloud-centric approach to a thing-/data-centric perspective, offering the potential to significantly alleviate various challenges, including high latency, limited scalability, quality of service, privacy, and security concerns. Given its immense potential, the IoT market is projected to reach a staggering $4.5 trillion by 2035, with an interconnected network of over one trillion devices spanning across diverse domains such as smart homes, cities, industries, healthcare wearables/implants, and agriculture [1]. Ericsson's research suggests that intelligent IoT systems can reduce carbon emissions by an impressive 3%, equivalent to 63.5 gigatons, by the year 2030 [2]. However, the current reliance on batteries to power IoT devices presents significant challenges, including limited lifespan, cost, maintenance, and environmental sustainability concerns. EnABLES projects that without proactive measures, global battery disposal could reach an alarming 78 million units per day by 2025 [3], highlighting the urgent need for sustainable solutions in the rapidly expanding IoT ecosystem. In contrast, energy harvesting systems represent a critical advancement in the development of sustainable, autonomous computing devices, particularly within the IoT. These systems derive power from environmental sources like solar radiation, thermal gradients, and ambient RF energy. The principle behind energy harvesting is to capture these omnipresent energies and convert them into electrical energy to power electronic devices. This approach enables devices to operate independently of conventional power grids, facilitating deployments in remote or inaccessible areas without regular maintenance. The energy harvesting system needs an intermittent execution approach, where the system must adapt to periods of activity interrupted by power failures. Unlike stable sources, intermittent energy sources can disrupt program execution, causing data loss and unpredictable outcomes. Intermittent computing offers near-zero idle power consumption, instant wake-up, and robustness against power failures [4]. Non-volatile memory (NVM) components prevent the need for a boot-up sequence post-sleep [5]. The advent of CNNs has led to a shift in inference tasks from the cloud to the edge of the IoT, enabling responsive applications with reduced bandwidth and storage requirements. However, IoT edge devices often rely on energy harvesting solutions and are prone to frequent power failures, resulting in intermittent execution. Consequently, intermittent CNN inference has emerged as a critical challenge for modern, self-powered edge devices. Existing intermittent solutions, such as checkpointing and task-based programming models, are not well-suited for hardware-accelerated CNN inference due to performance issues, hardware supportability, energy estimation challenges, and increased programmer burden. Although hardware acceleration makes local CNN inference feasible on lightweight devices, the power consumption of CNN accelerators is still several orders of magnitude higher than what ambient energy harvesters can provide. Traditional checkpoint-based strategies are vulnerable to inconsistencies during power failures, leading to partial execution context retention or inconsistencies between checkpoints. Checkpointing and task-based methods address intermittent program execution but do not directly support intermittent peripheral operation. Recent efforts to dynamically scale atomic peripheral operations based on estimated energy budgets are hindered by the difficulty of accurately predicting available energy at runtime. While preserving the state of simple peripherals is possible, it remains challenging for complex hardware accelerators with inaccessible internal states. Despite extensive research using conventional methods and complex programming paradigms, these techniques often face performance bottlenecks and scalability limitations [6].

This paper proposes a sustainable reactive intermittent execution paradigm, namely **SRC**, which combines the principles of intermittent computing with a novel reactive execution model to adapt to the dynamic energy conditions inherent in energy harvesting systems. The proposed solutions target off-the-shelf single-core ultra-low-power microcontroller units (MCUs) with limited flexibility and capability. The SRC source codes are publicly available on the Github[1] repository.

[1]https://github.com/iDEALabAcademy/BRIE-IC

979-8-3315-0786-2/24/$31.00 ©2024 IEEE
DOI 10.1109/IGSC64514.2024.00026

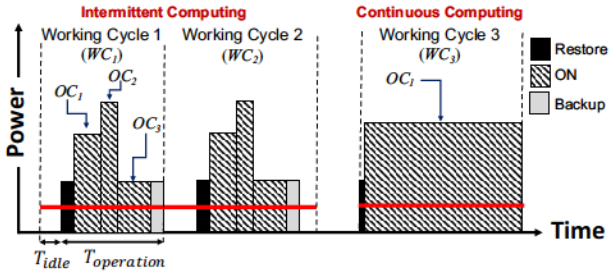

Fig. 1. Two computing paradigms, and key different.

II. Background and Motivation

A. Energy Harvesting System

Energy harvesting systems function intermittently, activating only when there is sufficient environmental energy and entering a state of power failure when the energy is insufficient. Therefore, the operation of energy harvesting systems typically alternates between active periods and power-saving states. Devices are engineered to collect energy slowly, store it in elements like capacitors, and then consume this stored energy rapidly during active phases. This cycle presents unique challenges, especially the fast depletion of energy compared to its collection rate, which can lead to the loss of volatile memory states, e.g., registers, during power outages, although NVM remains unaffected. Figure 1 illustrates the contrasting power management strategies between intermittent and continuous computing in an energy harvesting system. In the intermittent computing regime, the device experiences fluctuating power availability, which results in multiple operational cycles marked by bursts of activity (Operating Cycles OC_1, OC_2, OC_3) followed by inactivity. During each Working Cycle (WC_1,WC_2,WC_3), the system alternates between periods of computational operation (T_{op}) and idle time (T_{idle}), with crucial moments allocated for saving and restoring the system's state to ensure data integrity despite power losses. In contrast, continuous computing is characterized by a stable power input that sustains ongoing device operation, as indicated by a single, extended operating cycle (OC_1). This consistent power ensures that computational tasks can be carried out without interruption. The clear delineation between these modes emphasizes the need for robust power and task management strategies in energy harvesting systems to accommodate the inherent variability of power sources [7].

B. Intermittent Computing

Intermittent computing is characterized by execution that is not steady and unbroken but rather sporadic, resuming operations as energy permits. This approach is crucial in applications where the energy source cannot be controlled or predicted accurately, such as in devices powered by environmental energy. The core challenge is designing systems that can survive frequent power disruptions and operate effectively within these constraints. Several strategies to handle intermittent computing in energy-limited settings have been explored:

1) Static: Checkpointing is a widely used technique in intermittent computing to ensure progress across power failures. When a power failure is imminent, the system state (registers, memory contents) is saved to NVM. Upon power restoration, the system resumes from the last checkpoint [8], [9]. However, they suffer from high overhead due to frequent checkpoints of large system states, which may lead to inconsistencies if a power failure occurs during checkpointing and may result in wasted energy if checkpoints are taken too frequently.

2) Task-based: Task-based approaches divide the application into atomic tasks that can be executed within the available energy budget. Checkpoint is performed only at task boundaries, reducing the overhead compared to checkpointing-based approaches. Nevertheless, they require programmer intervention to define tasks and specify what data to save, which may lead to wasted energy if a power failure occurs in the middle of a task [10], [11].

3) Reactive: Reactive intermittent computing avoids the drawbacks of static and task-based approaches by reacting to changes in energy availability [12]. It typically uses voltage detection circuits to monitor the supply voltage and triggers a checkpoint only when power failure is imminent. Reactive approaches minimize unnecessary checkpoints, improve energy efficiency, avoid code re-execution and memory inconsistencies, are compatible with existing software with minimal modifications, and are portable across hardware platforms. However, saving and restoring the entire system state can be expensive, handling failure-atomic sections (FASEs) is more challenging, and they require a minimum amount of energy buffering to guarantee successful checkpointing.

C. Energy Harvesting CNN Accelerator

By harnessing environmental energy, energy harvesting CNN accelerators revolutionize edge intelligence, realizing complex on-device inference and reducing latency, bandwidth, and cloud dependence.

1) Hardware Approaches: Hardware accelerators, such as Processing in-Memory (PIM) architectures, can significantly improve CNN inference's performance and energy efficiency on resource-constrained devices [13], [14]. They offer high performance and energy efficiency, reducing data movement overhead, and are suitable for computation-intensive workloads like deep learning inference. However, they require specialized hardware support, may incur high write energy and latency for reconfiguring NVM cells, and volatile state elements in hardware accelerators lose computational state upon power failures.

2) Software Approaches: Software approaches complement the hardware strategies by optimizing the neural network models to fit the energy harvesting paradigm. Optimization techniques, such as model compression, quantization, and pruning, can reduce the computational requirements and memory footprint of deep learning models, making them more suitable for intermittent execution [15]–[17]. However, they may lead to some accuracy degradation, require retraining of the compressed/quantized models, and not all models are amenable to aggressive compression or quantization.

3) Adaptive Inference: Adaptive inference techniques, such as ePerceptive [18], dynamically adjust the computational complexity of CNNs based on available energy, trading off accuracy versus energy efficiency. This can be achieved through techniques such as early exit, resource scaling, or approximate computing. Adaptive inference allows for graceful degradation of accuracy under low-energy conditions and enables the system to progress even with limited energy availability, e.g., SONIC [19] and HAWAII [20]. However, training adaptive models may lead to variable inference latency and accuracy, demanding runtime monitoring of energy availability.

III. Proposed SRC Approach

Motivated by the abovementioned challenges, we propose a lightweight and generic intermittent paradigm, namely SRC, to optimize CNN inference and maximize forward progress with minimal checkpointing and programming overheads. SRC combines the principles of intermittent computing with a reactive execution model to adapt to the dynamic energy conditions inherent in energy harvesting systems. Our approach is inspired by the HAWAII approach, leveraging the footprinting concept to save the inference footprints (i.e., layer (L), kernel (K), and input feature map's (ifmap) ($H \times W$) indices) and the intermediate output feature map (ofmap) results. The key distinction from HAWAII lies in storing only the remaining portion of ifmaps based on the last completed ofmap's index. In contrast, HAWAII necessitates copying the entire ifmap to NVM, leading to more NVM checkpoints. Furthermore, HAWAII must ensure that the capacitor's energy budget is adequate for the accelerator's most demanding suboperation. However, SRC processes each layer independently, considering its unique parameters and features. Herein, we ensure atomic execution at the suboperation level, which is designed to be completed within a few clock cycles and, if disrupted by a power failure, will be re-executed. This approach allows us to relax the atomicity constraints significantly, eliminating the need for task partitioning based on application-level energy estimations. Unlike checkpointing methods, SRC saves the outputs of suboperations to preserve forward progress without suspending the application, substantially reducing the overhead associated with state persistence. Inaccurate energy cost or availability estimation can cause task-partitioning approaches to fail, requiring repeated execution in subsequent power cycles. Conversely, our reactive approach captures and utilizes processor footprints to resume interrupted operation, enabling completion over multiple power cycles. Notably, the recovery cost decreases with each power cycle, as only the necessary input data for remaining suboperations is fetched during inference state recovery.

A. Architectural States and Transitions

Figures 2(a)-(b) show finite state machine (FSM) of intermittent computing, conventional reactive, and SRC schems, respectively. The architectural state of SRC includes eight states `Off`, `Stop` (Sp), `Sense` (Se), `Compute` (Cp), `Transmit` (Tr), `Store` (Str), `Load` (Ld), and `Standby` (Stb). Herein, the core operations of an IoT node are categorized into three essential functions: sense, computation, and transmission, collectively referred to as `Application` (App) state. A 2-bit register stores the next state of the system, which should be one of the `App` states, and a single bit indicates whether the NVM has been validated, referred to as `next_state` and `NVM_flag`, respectively. When the system has sufficient energy to activate, it begins from the **`Stb`** state. Based on `NVM_flag`, the system either proceeds to **`Ld`** if necessary or moves directly to **`Sp`**. In the **`Sp`** state, depending on `next_state`, the state may change to **`Se`**, **`Cp`**, or **`Tr`**. The system remains in these states unless the system energy falls out of the `safe_zone` or the entire operation is completed, necessitating a state change. As depicted in Fig. 2(b), there are no direct connections between states within `App`; state changes are only mediated through the **`Sp`** state. If the energy falls below the `store_threshold` (as explained later), all required values and registers are stored in NVM. If the store operation completes successfully, the system returns to **`Sp`**; otherwise, it transitions to **`Stb`**. The system remains in this state until the power is completely lost or restored to the operational threshold. It should be noted in the **`Stb`** state that the values in SRAM and registers will be lost. In the event of a power failure after transmission has been done, there is no need to store anything. The proposed SRC has an additional threshold voltage, termed `safe_threshold`, which determines when the MCU is Stopped. When the system energy decreases to this level, it halts current tasks and transitions to **`Sp`**, where the power consumption of MCU is much less than in operational mode. In the conventional reactive approach, the system continues its tasks until the system reaches the backup threshold. At this point, the application will suspend and perform **`Str`**, and thereafter, the system enters **`Stb`** state. The concept of a `safe_zone` reduces the system's store and load operations.

After the CNN architecture has been specified, the pro-

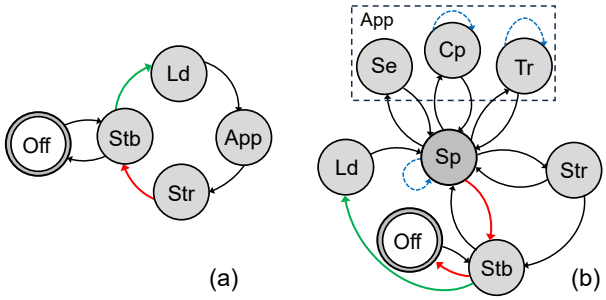

Fig. 2. State machine representations of (a) reactive, and (b) SRC schemes.

are loaded instantly, and the corresponding MCU operation is initiated. Once the end-to-end inference process is complete, SRC signals the user program to indicate its completion. The following sections describe each component of SRC:

`Stop (Sp):` Herein, the aim is to maximize power savings by primarily using the *Stop* mode, bypassing *Sleep* due to higher power consumption. In *Stop*, all SRAM and register values remain valid, and the Real-Time Clock (RTC) stays functional, allowing the microprocessor to periodically wake up, check status and power, and transition to *App* states.

`Standby (Stb):` Whenever the system's energy falls below V_{Stb}, the microprocessor transitions to the **`Stb`** state. In this mode, all data in the SRAM and registers will be lost. It should be noted that an external power management unit is responsible for handling the *standby* and *wake-up* operations of the MCU.

`Sense (Se):` In SRC architecture, the sense operation is performed atomically. After initiating this state, the system will return to the **`Sp`** state, following a predetermined delay time and power consumption. Users have the flexibility to implement more complex sensing functions by modifying the SRC code.

`Compute (Cp):` SRC supports CNNs with up to 256 layers and 1024 kernels. It assumes a minimum energy for 1,000 computations to ensure at least one ofmap generation. The optimal configuration is the input stationary approach, where all kernels are initially applied to the input, computing ofmaps. Thus, after applying all kernels, the input need not be stored in case of power failure.

`Transmit (Tr):` In the **`Tr`** state, the MCU operates, and the transceiver chip is reactivated. After computation, all `state_register` values reset to zero. Each transmitted element increments the corresponding indices. In SRC, the last network layer represents the final result, so the first 8 bits of the `state_register` are unused, with the rest used similarly to the **`Cp`** state.

`Store (Str):` In **`Str`**, all results and system states must be stored in NVMs. A 48-bit `state_register` tracks the progress of multiply-accumulate (MAC) operations in CNN layers, allocated as: 8 bits for layers (l), 10 bits for kernels (k), and 15 bits each for width (w) and height (h) of the input. If a power failure occurs during computation, calculated ofmaps and unused ifmap are saved in the NVM. Backup points are based on $\lceil \frac{w}{kernel\_size} \rceil$ and $\lceil \frac{h}{kernel\_size} \rceil$ for w and h, respectively. For data transmission failures, only the intended data is recorded in the NVM. SRC stores data from the most to least significant bits, maintaining functionality with approximate data during power failures.

`Load (Ld):` When transitioning from **`Stb`**, the system may load data from NVM. If in **`Sp`** awaiting a new sample, no NVM data retrieval is needed, setting `NVM_flag` to false. If `NVM_flag` is true, `next_state` and required input values are loaded into shared memory based on `state_register`. SRC retrieves output from NVM only after **`Cp`** completion. In **`Ld`**, only the remaining ifmap is loaded, significantly reducing NVM read/write operations in **`Str`** and **`Ld`** states.

IV. Results Analysis

A. Experimental Setup

We developed a system-level framework to validate our approach, integrating it with the proposed FSM and evaluating performance using our cross-layer framework. We simulated an intermittent power source, cycling through a predefined sequence of voltage levels to mimic battery behavior, which accumulates energy during availability and depletes it during outages. We monitored critical parameters such as power levels, availability, and the status of the virtual energy store. The system's design includes a capacitance of 2mF and operates at 5V, capable of storing up to 25mJ of energy. A series of experiments on STM32F107vc and BlueNRG1 for Bluetooth Low Energy (BLE) communication (hereafter Platform). Detailed specifications for the configuration are presented in Table I. We considered that various microcontrollers employ different power management strategies, primarily through low-power states. For instance, based on STM32F107vc datasheets, the connectivity line features three low-power modes designed to achieve an optimal balance between low power consumption, fast startup times, and a range of available wake-up sources. In *Sleep* mode, only the CPU is halted. All peripherals remain operational and can awaken the CPU upon interrupt or event occurrence. Conversely, in *Stop* mode, all clocks cease, yet the contents of SRAM and registers within the 1.8 V are preserved. In *Standby* mode, the entire 1.8 V domain powers down, resulting in the loss of all register values. For this analysis, registers and RAM values are assumed to persist in *Stop* mode but are lost in *Standby* mode.

Figure 3 shows an example of SRC implementation, where (a) depicts the energy (E_{Batt}) stored in the capacitor, including all the considered thresholds. The yellow circles in Fig. 3(a) highlight instances where the `safe_threshold` prevents unnecessary backups, even in cases where energy returns after a backup, eliminating the need to load data from NVM. Figure 3(b) and (c) show the system's charging rate and a small portion of the system's state changes, respectively. Note that Fig. 3(c) represents the state transitions themselves, not the duration of each state. The system begins in the **`Sp`** state, progressing through **`Se`** and **`Cp`**. In the **`Cp`** state, a power drop below the `store_threshold` triggers a backup of all required values to NVM, followed by a transition to **`Stb`**. Consequently, data must be loaded from NVM (**`Ld`**) when energy is restored. The system frequently alternates between the **`Sp`** and **`Cp`** states during the remaining transient phases. The SRC

TABLE I
Specification of the experimental platform.

Hardware Model		Examined Platform	
MCU	CLK		72 MHz
	Memory		64 to 256 Kbytes of Flash, 64 Kbytes of SRAM
	Current	Sleep	49 mA
		Stop	33 µA
		Standby	3.8 µA
		Compute	68 mA
Transmitter	Current	Sleep	3.500 nA
		Receive	7.7 mA
		Send	15.1 mA
	Data Rate		2 Mbps

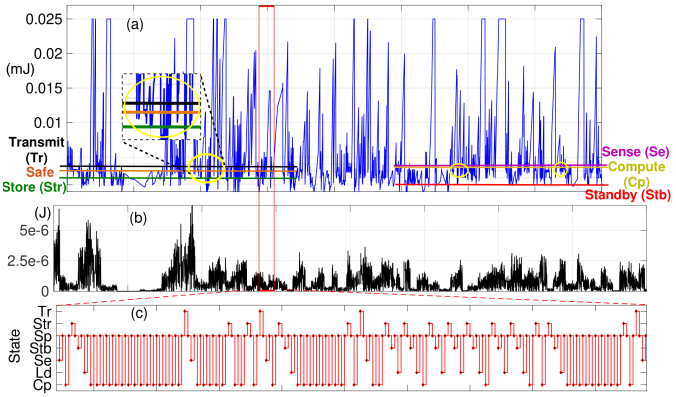

Fig. 3. (a) The system energy behavior and (b) the charging rate. (c) A small portion of the system's states.

mechanism demonstrates how these transitions temporarily halt computations before a critical energy drop, reducing the overall number of stores and loads. The system transitions to **`Tr`** upon computation completion, followed by a return to **`Se`** to capture another sample. The fluctuation period between **`Cp`** or **`Tr`** and **`Sp`** depends on the system's available energy.

B. Performance Evaluation

To evaluate the effectiveness of the proposed SRC approach, we tested it under two different intermittent energy traces (ET1 and ET2), using two CNNs (AlexNet and VGG11). The ET1 models a mild condition with minimal power loss, whereas ET2 represents a harsh condition. All the battery energy patterns and the systems' states of PLATFORM are depicted in Figs. 4 (a-b) and subfigures (c-d), respectively. Two captured energy traces from the environment are shown in Figs. 4(e-f). Therefore, the results shown in subfigures (a) and (c) are based on ET1 (e). As previously elaborated, in Fig. 4(c), ❶, the system initially transitions to a **`Sp`**, **`Str`**, then to **`Stb`**, and finally moves to **`Ld`** to retrieve data. If the system's energy repeatedly falls below the `store_threshold` and quickly recovers, data is stored but not reloaded, as it is not lost in this step. The influence of ET2 (Fig. 4(f).), which has a lower charging rate, is directly illustrated by ❷ in Fig. 4(d). Thus, the lack of energy disrupts the uniformity of the sensing distances. Moreover, as shown in ❸ and ❹, within the same time window, the number of senses in ❸ exceeds that in ❹, indicating greater computational energy availability for the platform. Furthermore, as depicted in ❹, energy scarcity disrupts computation, causing frequent system state transitions to `Stb`, `Str`, and `Ld`.

A further testbench used to evaluate SRC involves changing the network architecture. Running VGG11 and AlexNet requires 9.2 billion and 1.5 billion MAC operations, respectively. Given a selected microprocessor with 90 MIPS, performing the inference for a single input (i.e., 224×224) takes at least 172 seconds. SRC is tailored for processors with limited capabilities and IoT devices, typically resulting in longer processing times. The results are listed in Table II, showcasing the performance and efficiency across different networks, along with the normalized number of frames for each system. The results indicate that larger networks consume more energy to store outcomes compared to smaller ones, which is a logical outcome considering their computational demands. The frame rate further evidences the SRC's efficacy; larger networks, requiring more energy for both computation and transmission, yield a lower frame rate.

TABLE II
ENERGY CONSUMPTION (J) FOR TWO CNNS UNDER ET1.

Network	`Stb`	`Sp`	`Se`	`Cp`	`Tr`	`Ld`	`Str`	Frame
VGG11	0.088	0.42	0.0116	1.962	0.008	5.24e-06	5.24e-05	11.8
AlexNet	0.117	0.78	0.0149	2.172	0.003	5.24e-06	5.24e-05	16.6

As discussed earlier, SRC minimizes unnecessary standby, store, and load operations. To demonstrate the significance of our approach, we evaluated it with the conventional reactive intermittent computing scheme. The simulation results are presented in Table III. Additionally, the energy savings achieved enhance the frame processing capability, as shown by the increased number of **`Se`** operations. This effectiveness of thresholding technique can be attributed to the energy consumption pattern in PLATFORM, where most of the energy is utilized in computation that can be efficiently segmented. The results demonstrate that this technique significantly reduces the average number of read and write operations by 88.1% and 79.1%, respectively. Moreover, by comparing the **`Ld`** and **`Str`**

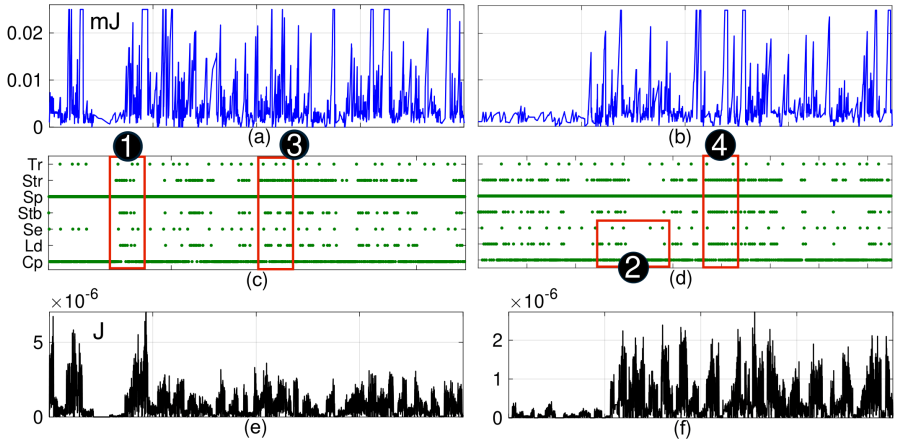

Fig. 4. Subfigures (a) and (b) show energy profiles for PLATFORM across two energy traces (e) and (f), respectively. Their operational states are detailed in (c) and (d), respectively.

TABLE III
COMPARISON BETWEEN THE REACTIVE SCHEME AND SRC.

Design	Power Trace	Number of Occurrences							Improvement(%)*			
		Stb	Sp	Se	Cp	Tr	Ld	Str	Stb	Se	Ld	Str
Reactive	ET1	898	4407	27	2041	26	898	898	-	57.8	-	-
Reactive	ET2	1530	6422	28	2380	33	1530	1530	-	22.2	-	-
SRC	ET1	108	1715	64	1155	63	106	220	87.9	-	88.2	75.5
SRC	ET2	90	1073	36	686	45	83	143	94.1	-	94.6	90.65

*The higher (lower) the **Se** (**Stb**, **Ld**, **Str**), the better performance.

counts, the efficacy of our method is evident. For instance, in ET1, the number of **Str** is improved by approximately 75%.

C. Consideration & Limitations

The proposed SRC implementation requires extra code to adjust the MCU's states based on the system energy. This can be combined with hardware acceleration and software optimization for further improvements. We did not consider any machine learning compression methods, such as pruning, quantization, etc., which are beyond the scope of this paper.

Compute: In computing, the user needs to take care of the indices. As mentioned, SRC is input stationary friendly, but the values of `state_register` should be updated after calculating each output feature map.

Transmit: To support the approximation data in case of power loss, users need to modify transmission algorithms to send the most significant bits first. This operation requires more computation steps than conventional methods, but its overhead is negligible due to the high power consumption of transmitting (about 1000 times more than computation). In this state, the user also needs to keep the `state_register` updated. While in transmission, users do not need to update the number of layers.

Store: storing data in both computation and transmitting starts from the most significant bits. Users need to add the store and load functions in their code.

Interrupts: Two external interrupts are triggered by intelligent power management scheme. The first one changes the system's state to `Store` to hold data in NVM. The second one is related to stopping the process. Since all the applications have different threshold voltages, handling them using one common external power management is complicated. For this reason, only the stop points, which are the same among all of them (`safe _threshold`), are implemented using smart power management. By happening this interrupts, the application will be stopped, and the system's state changes to **Sp**.

RTC: The intermittent system is most of the time in a **Sp** or `Standby` states. As a result, users need to define an RTC to wake up the system regularly. The period of this timer can vary based on the application. In the case of a smaller value, the system consumes more power and runs faster. Depending on the system's energy, the system may or may not change the state to an application state after waking up.

V. CONCLUSION

SRC is a lightweight, adaptive framework that optimizes CNN inference on battery-free devices. Its reactive computing is combined with a dynamic energy-aware model to enhance efficiency in energy harvesting systems. SRC uses a `safe_threshold` voltage and suboperation-level atomic execution to reduce memory operations and relax atomicity constraints. We evaluated SRC via the STM32F107 platform, on different CNN models, and under various intermittent energy traces. SRC achieves on average $\sim 81.85\%$ reduction in read/write operations and $\sim 57.18\%$ improvement in sensing compared to conventional reactive methods. The simplicity, efficiency, and software compatibility make SRC a promising approach, enabling robust and sustainable edge intelligence.

ACKNOWLEDGMENT

This work is supported in part by the National Science Foundation (NSF) under grant numbers 2447566, 2303115, 2303116, and 2339193.

REFERENCES

[1] S. Liu *et al.*, "Energy-aware mac protocol for data differentiated services in sensor-cloud computing," *Journal of cloud computing*, vol. 9, pp. 1–33, 2020.
[2] Ericsson, "Climate action - ericsson," https://tinyurl.com/47zhzhpz, 2024, accessed: 2024-05-01.
[3] European Commission. (2024) Up to 78 million batteries will be discarded daily by 2025, researchers warn. Accessed: 2024-05-01. [Online]. Available: https://tinyurl.com/yh38efk9
[4] N. Taheri *et al.*, "Intermittent-aware design exploration of systolic array using various non-volatile memory: A comparative study," *Micromachines*, vol. 15, no. 3, p. 343, 2024.
[5] A. Roohi and R. F. DeMara, "Nv-clustering: Normally-off computing using non-volatile datapaths," *IEEE Transactions on Computers*, vol. 67, no. 7, pp. 949–959, 2018.
[6] S. Tabrizchi *et al.*, "Diac: Design exploration of intermittent-aware computing realizing batteryless systems," in *DATE*. IEEE, 2024, pp. 1–6.
[7] M. M. Sandhu *et al.*, "Task scheduling for energy-harvesting-based iot: A survey and critical analysis," *IEEE Internet of Things Journal*, vol. 8, no. 18, pp. 13 825–13 848, 2021.
[8] B. Ransford *et al.*, "Mementos: System support for long-running computation on rfid-scale devices," in *ASPLOS*, 2011, pp. 159–170.
[9] D. Balsamo *et al.*, "Hibernus: Sustaining computation during intermittent supply for energy-harvesting systems," *IEEE Embedded Systems Letters*, vol. 7, no. 1, pp. 15–18, 2014.
[10] A. Colin and B. Lucia, "Chain: tasks and channels for reliable intermittent programs," in *PACMPL*, 2016, pp. 514–530.
[11] K. Maeng *et al.*, "Alpaca: Intermittent execution without checkpoints," *Proceedings of the ACM on Programming Languages*, vol. 1, no. OOPSLA, pp. 1–30, 2017.
[12] S. T. Sliper *et al.*, "Efficient state retention through paged memory management for reactive transient computing," in *DAC*, 2019, pp. 1–6.
[13] L. Song *et al.*, "Pipelayer: A pipelined reram-based accelerator for deep learning," in *HPCA*. IEEE, 2017, pp. 541–552.
[14] X. Qiao *et al.*, "Atomlayer: A universal reram-based cnn accelerator with atomic layer computation," in *DAC*, 2018, pp. 1–6.
[15] S. Lee and S. Nirjon, "Neuro zero: a zero-energy neural network accelerator for embedded sensing and inference systems," in *SenSys*, 2019, pp. 138–152.
[16] M. Rastegari *et al.*, "Xnor-net: Imagenet classification using binary convolutional neural networks," in *ECCV*. Springer, 2016, pp. 525–542.
[17] C.-C. Lin *et al.*, "Intermittent-aware neural network pruning," in *60th ACM/IEEE DAC*. IEEE, 2023, pp. 1–6.
[18] A. Montanari *et al.*, "eperceptive: energy reactive embedded intelligence for batteryless sensors," in *SenSys*, 2020, pp. 382–394.
[19] G. Gobieski *et al.*, "Intermittent deep neural network inference," in *SysML Conference*, 2018, pp. 1–3.
[20] C.-K. Kang *et al.*, "Everything leaves footprints: Hardware accelerated intermittent deep inference," *IEEE TCAD*, vol. 39, no. 11, pp. 3479–3491, 2020.