Fine-grained Hardware Acceleration for Efficient Batteryless Intermittent Inference on the Edge

LUCA CARONTI, University of Trento, Italy
KHAKIM AKHUNOV, University of Trento, Italy
MATTEO NARDELLO, University of Trento, Italy
KASIM SINAN YILDIRIM, University of Trento, Italy
DAVIDE BRUNELLI, University of Trento, Italy

Backing up the intermediate results of hardware-accelerated deep inference is crucial to ensure the progress of execution on batteryless computing platforms. However, hardware accelerators in low-power AI platforms only support the one-shot atomic execution of one neural network inference without any backups. This paper introduces a new toolchain for MAX78000, which is a brand-new microcontroller with a hardware-based convolutional neural network (CNN) accelerator. Our toolchain converts any MAX78000-compatible neural network into an intermittently executable form. The toolchain enables finer checkpoint granularity on the MAX78000 CNN accelerator, allowing for backups of any intermediate neural network layer output. Based on the layer-by-layer CNN execution, we propose a new backup technique that performs only necessary (urgent) checkpoints. The method involves the batteryless system switching to ultra-low-power mode while charging, saving intermediate results only when input power is lower than ultra-low-power mode energy consumption. By avoiding unnecessary memory transfer, the proposed solution increases the inference throughput by 1.9× for simulation and by 1.2× for real-world setup compared to the coarse-grained baseline execution.

CCS Concepts: • **Computer systems organization** → **Embedded software**; • **Computing methodologies** → **Machine learning**.

Additional Key Words and Phrases: Intermittent computing, convolutional neural networks, edge computing, energy harvesting, hardware accelerator, checkpointing.

ACM Reference Format:
Luca Caronti, Khakim Akhunov, Matteo Nardello, Kasım Sinan Yıldırım, and Davide Brunelli. 2023. Fine-grained Hardware Acceleration for Efficient Batteryless Intermittent Inference on the Edge. *ACM Trans. Embedd. Comput. Syst.* 1, 1, Article 1 (January 2023), 19 pages. https://doi.org/10.1145/3608475

1 INTRODUCTION

Intermittent computing is a paradigm in which resource-constrained devices operate without batteries and execute programs during active periods interleaved with frequent power outages. In this paradigm, devices utilize energy harvesters to charge their short-term energy storage, such as capacitors, by harnessing environmental power, which may vary rapidly over time. When a device is powered on and actively computing, its energy storage frequently depletes, resulting in power failures that shut down the device and clear the volatile computational state, including the contents of main memory and registers. The device can turn on again once the energy storage is fully recharged. Given the intermittent nature of operation, it is essential to back up the computational state when a power failure is imminent to

Authors' addresses: Luca Caronti, luca.caronti@unitn.it, University of Trento, via Sommarive 9, Trento, TN, Italy, 38123; Khakim Akhunov, khakim.akhunov@unitn.it, University of Trento, via Sommarive 9, Trento, TN, Italy, 38123; Matteo Nardello, matteo.nardello@unitn.it, University of Trento, via Sommarive 9, Trento, TN, Italy, 38123; Kasım Sinan Yıldırım, kasimsinan.yildirim@unitn.it, University of Trento, via Sommarive 9, Trento, TN, Italy, 38123; Davide Brunelli, davide.brunelli@unitn.it, University of Trento, via Sommarive 9, Trento, TN, Italy, 38123.

Manuscript submitted to ACM

preserve computational progress. Embedded non-volatile memory (e.g., FRAM (Ferroelectric-RAM) [20]) in de facto microcontrollers (MCUs) in intermittent computing platforms (e.g., TI MSP430FR series [21]) retains data after power failures. It enables backup and recovery of computational state to ensure the progress of energy harvesting applications running intermittently [6, 26, 40, 44, 45].

As the trend towards enabling ultra-low-power intelligence at the edge grows [13], standard intermittent computing platforms become obsolete and inefficient to execute parallelizable and data-intensive machine learning loads on batteryless edge devices [2, 16, 25, 28]. Meanwhile, several low-power MCU-based platforms facilitating AI at the edge are emerging. Compared to the TI MSP series MCUs, these platforms offer more computational power and advanced hardware acceleration, but they do not have embedded non-volatile memory. For example, MAX78000 [22] is a new breed of microcontroller with two low-power cores (ARM Cortex-M4 and RISC-V) and a hardware-based convolutional neural network (CNN) accelerator that consumes only micro-joules of energy for a single inference. The only way to make this MCU suitable for intermittent computing is to use external non-volatile memory (e.g., SPI-based FRAM), which allows backup and power-failure recovery during hardware-accelerated intermittent execution of complex machine learning loads. However, this setting introduces extra overhead in terms of energy and time due to serial communication during non-volatile memory access (i.e., SPI overhead).

Problem Statement. In addition to the volatile computational state of the MCU, the hardware accelerators in low-power AI platforms also include volatile state elements that lose their computational state upon power failures. Moreover, their energy consumption during inference depends on the neural network (NN) size, which is application dependent. For example, the CNN accelerator on MAX78000 assumes a continuous input power and only supports the one-shot atomic execution of one neural network inference without any backup. In this case, power failures might lead to a significant amount of wasted energy due to repeated hardware reconfiguration, e.g., reloading weights from the non-volatile memory into the volatile buffer of the accelerator. Moreover, if the size of the energy storage capacitor is not big enough to execute the whole inference at once, power failures might lead to non-termination. Without backing up intermediate results and recovering them after reboot, the hardware-accelerated deep inference might never be completed. On the other hand, backup and recovery can also be costly due to the communication overhead of the external nonvolatile memory, diminishing the efficiency of the hardware-accelerated deep inference.

Contributions. In this paper, we introduce a new toolchain called LbLTT[1] (**L**ayer-**b**y-**L**ayer **T**ransient **T**oolchain) that converts any neural network model for the CNN accelerator in MAX78000 platform into a layer-by-layer executable form. The conversion makes the inference execution granularity finer, shrinking the atomically executed unit size to the layer size of the neural network. This reduction allows for flexibility in matching the intermittent inference execution to the limited energy capacity since it is possible to back up any intermediate layer output (weights, biases, and feature maps) in external non-volatile memory. Using this toolchain, we transform various neural networks into intermittently-executable forms and evaluate different backup strategies by considering their time, energy, and memory costs. Furthermore, we propose a novel backup strategy that eliminates redundant checkpoints and reduces the frequency of SPI-based non-volatile memory access, allowing more time and energy on computation. Our comparison against state-of-the-art solutions shows that the proposed backup strategy improves the inference throughput by 1.9×.

The rest of this article is organized as follows. Section 2 presents background on intermittent computing and hardware-accelerated and low-power deep inference. In Section 3 we present our approach to executing intermittently

[1]Available at https://git.iotn.it/lucacaronti/LbLTT

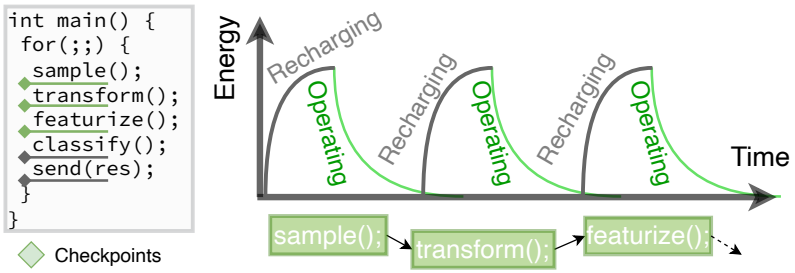


Fig. 1. An example of intermittent computation. A batteryless edge device performs a series of computations and then stores the result, alternating this process with periods of inactivity where the device shuts off and gathers energy to charge its energy storage.

hardware-accelerated convolutional neural networks in a layer-by-layer manner. Results and evaluation are then discussed in Section 4. Finally, Section 5 concludes the paper with some final remarks and future works.

2 BACKGROUND AND RELATED WORK

Energy harvesting batteryless embedded devices operate intermittently (see Figure 1) due to frequent power failures, which occur since energy sources (e.g., sunlight, radio waves, or vibration) are transient and energy storage capacitors can store a finite and small amount of energy. Each power failure resets the device and clears the contents of its volatile hardware state elements, leading the program control flow to return back to the beginning of the application entry point. To progress computation and keep memory consistent [12], prior works proposed various software solutions for intermittent computing that mainly deal with when to backup and what to backup.

2.1 Intermittent Computing Approaches

Employing checkpoints and task-based programming are the two state-of-the-art intermittent computing solutions [15]. Checkpoints back up the current computational state into non-volatile memory only at the source code locations specified by the programmer or compiler [2, 6, 9, 14, 26, 31, 45]. At a power restore, the computation continues from the last successfully performed checkpoint. The just-in-time checkpoints [11, 32] require additional dedicated hardware that monitors the capacitor voltage level to signal when the voltage level drops below a predefined voltage threshold. At this point, the program must back up the current computational state, turn off the system, and wait for charging. In task-based approaches, programmers split the application into idempotent atomic subtasks that can fit into the capacitor and define a task-based execution flow [5, 12, 42, 44]. The backup operations are performed only at task boundaries. If a power failure happens in the middle of a task execution, the intermediate results are discarded, and the computation re-starts from the beginning of the interrupted task. In this work, we consider checkpoint-based backup approaches since they do not require programmer intervention and are simpler to employ.

2.2 Machine Learning on Intermittent Power

With the increasing number of IoT devices, the amount of sensed data from the environment is growing at an exponential rate. This data is collected by cloud servers, where advanced machine learning (ML) techniques are employed for processing, and the relevant information is then sent back. However, in many cases, these remote servers are located quite far from the sensing device, resulting in increased latency and communication expenses. Besides, some of

the data contain redundant and not very useful information, which wastes communication bandwidth and energy. For energy-harvesting devices, the energy required for communication is significantly higher compared to local sensing and computation. Hence, it is more reasonable to shift ML inference to an edge device or to a sensor itself. Recent work has proposed solutions to enable and enhance inference on resource-constrained embedded systems, e.g., [1, 3, 24, 33, 36], and especially on intermittent computing systems by various means: software frameworks [16], hardware accelerators [25, 27], new architectures [17], and even emerging technologies [39]. As an example, Sonic [16] is the first framework that demonstrated deep inference on TI MSP series MCUs by also exploiting its low-energy accelerator (LEA) to perform vector-based operations efficiently, such as matrix-vector multiplication.

2.3 Hardware-accelerated Deep Inference on Low-power

Several brand-new ultra-low-power MCUs with advanced hardware accelerators are appearing in the market. One notable example is Syntiant's Tiny Machine Learning Development Board [43], which features a low-power ML accelerator. The board is equipped with the ultra-low-power NDP101 Neural Decision Processor and packs native neural network computation in the lowest power envelope. NDP101 achieves higher performance by exploiting inherent deep learning parallelism and computing only at the required numerical precision. Another noteworthy product is the MAX78000 [35], which belongs to a new line of ultra-low-power microcontrollers. This microcontroller incorporates a low-power CNN accelerator designed to facilitate machine learning tasks. Remarkably, it can execute inference operations while consuming merely micro-joules of energy, making it an ideal choice for batteryless edge devices.

These MCUs have not yet been adapted to intermittent computing, as there is currently a lack of strategies to support the backup of computational tasks executed on these advanced accelerators. An example of this is evident in the MAX78000's CNN accelerator. The issue is that, depending on the NN model, distinct NN layers have different energy and computational requirements. To execute inference on the MAX78000's CNN accelerator intermittently, it becomes essential to understand the energy requirements of each layer and to enable more fine-grained backups (e.g., layer-by-layer). To the best of our knowledge, when, what, and how to back up during hardware-accelerated intermittent inference remains unexplored, which makes these issues the focus of this article.

FPGAs (Field-Programmable Gate Arrays) are also highly promising technologies for accelerating inference on edge devices. Furthermore, several studies have been conducted to evaluate the utilization of FPGAs in intermittent computing scenarios, specifically focusing on backup strategies [34, 37, 41] and the implementation of intermittent computing [19, 46, 47]. Nevertheless, it is important to acknowledge that FPGAs do have certain limitations stemming from their power consumption and the memory technology they utilize. Additionally, it is worth noting that non-volatile FPGAs have not yet been widely adopted in commercial products, making FPGAs still not a very suitable technology for intermittent computing. Therefore, in this article, we target resource-constrained MCUs equipped with neural accelerators, in particular, the MAX78000.

3 LAYER-BY-LAYER ACCELERATED DEEP INFERENCE ON INTERMITTENT POWER

In this paper, we exploit MAX78000, a new AI microcontroller with an on-chip CNN accelerator that enables battery-powered applications to execute AI inferences while consuming only micro-joules of energy. MAX78000 supports a wide variety of both CNN and DNN models, has a comprehensive Software Development Kit (SDK) and active development community. It is equipped with two CPUs, one of which is an ultra-low-power RISC-V processor. The CNN accelerator features 64 parallel processors split into four equal quadrants, as shown in Figure 2. The accelerator has dedicated memory for corresponding weights (440KB), biases (2KB), and data (512KB). The accelerator also has a connection

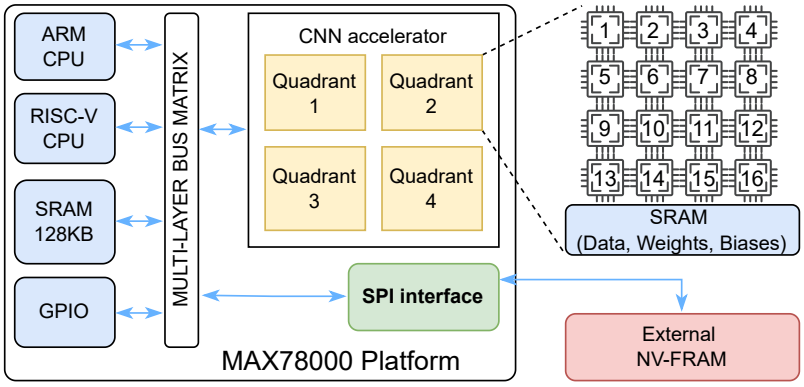


Fig. 2. MAX78000 platform architecture.

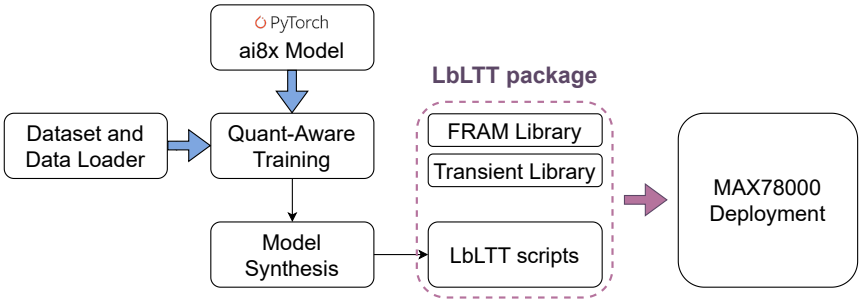


Fig. 3. Proposed MAX78000 extended toolchain for intermittently executable inference on the CNN accelerator.

to the multi-layer bus matrix shared with CPUs, platform memory, and other peripherals. All the internal memory components of MAX78000 are volatile (blue blocks in Figure 2).

The MAX78000 SDK [23] eases training, synthesizing, and deploying NN models on the accelerator. As shown in Figure 3, the training is performed with the help of the PyTorch library for machine learning, data loader, and corresponding dataset. The synthesis and deployment stages are responsible for converting the trained NN model to human-readable C code. Once the model is deployed on the device, the developer follows the steps shown in Figure 5 (left). First, the accelerator is initialized by setting the SRAM control bits and the number of CNN layers. Then, weights and biases are loaded from the general-purpose SRAM to the dedicated memory. These steps are followed by the CNN configuration, which sets the number of rows and columns for each acceleration quadrant (i.e., the group of 16 processors) and specifies corresponding memory pointers. Finally, input data is loaded into the accelerator memory, and the inference starts.

3.1 Challenges of Intermittent Inference on MAX78000

Inference operation on MAX78000 CNN accelerator is a single atomic task, i.e., if the execution is interrupted by a power failure, the inference loses all intermediate computational results, starting the computation from the initialization phase. To exclude the occurrence of non-terminating execution of intermittent inference, the capacitor of a device must be large enough to store at least the energy for restoration, single inference, and backup. However, enlarged energy

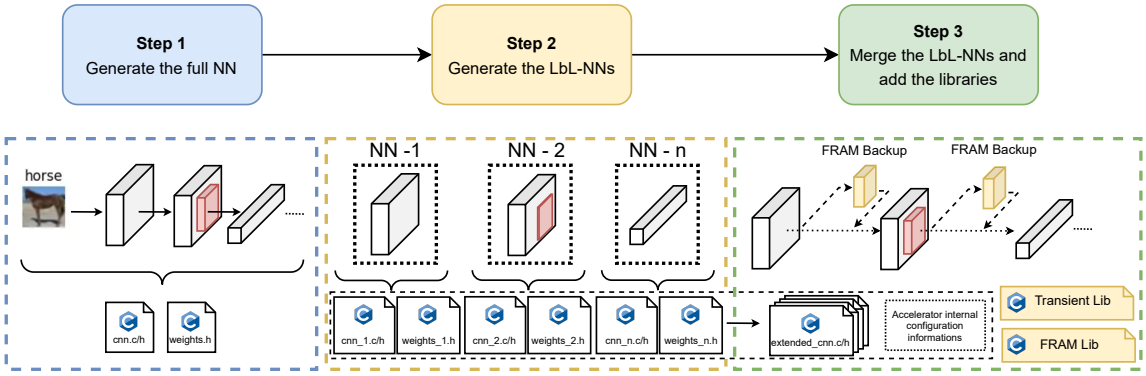


Fig. 4. LbLTT workflow.

storage increases the charging time as well as the size and weight of the device. Moreover, to ensure execution progress, the platform requires an expensive SPI-based connection to external non-volatile memory (red block in Figure 2). With frequent backups, such extraordinary memory access can annihilate the energy and performance efficiency of the CNN accelerator.

3.2 Layer-by-Layer Transient Toolchain (LbLTT)

We address the mentioned challenges by developing a new toolchain, named Layer-by-Layer Transient Toolchain (LbLTT). The toolchain extends the MAX78000 software stack, allowing developers to control inference execution layer-by-layer. As shown in Figure 3, the proposed toolchain (pink dashed block) converts the neural network synthesized into an intermittently executable form before deploying it to the network accelerator. This allows the platform to work in an intermittent scenario, where the atomic tasks are represented by the evaluation of every single layer of the neural network. It is then possible to eliminate the need for capacitors large enough to sustain the whole inference since only the energy for restoration, single-layer evaluation, and checkpointing is needed. LbLTT workflow is organized in three steps, presented in Figure 4. The process starts from the synthesized C code using the *ai8xize.py* Python script. This step consists of the full neural network project generation. It converts the Pytorch NN model into C code. Two output files are generated: *cnn.c* contains all the functions to load, start, and get the output of the NN; and *weights.c* contains the NN weights. The second step generates each individual layer using LbLTT (it actually provides the segmentation). Here, a standalone NN consisting of input, a layer, and output is created for each layer of the original NN. This allows the generation of the configuration files used to configure the CNN accelerator at the beginning of the computation of each layer. Finally, in the third step, all layers are merged and linked into one combined NN execution. The previously generated neural networks are interpolated and linked together by adding a series of functions that allow the intermediate output to be extracted, saved in the FRAM and reloaded when necessary. In this step also, the intermittent functionalities – such as energy monitoring and checkpointing strategy – and the code for interfacing the external FRAM are merged into the project. The resulting C code is then ready to be deployed on the MAX78000 CNN accelerator, supporting intermittent execution. To ease this process, a Python script automates all the steps in one command line. The command takes as an argument a single configuration file in JSON format, which contains parameters such as the number of layers to split, the location of the original neural network to be converted, and the threshold for the checkpointing mechanism. Figure 5 presents the CNN execution flowchart on runtime. On the left,

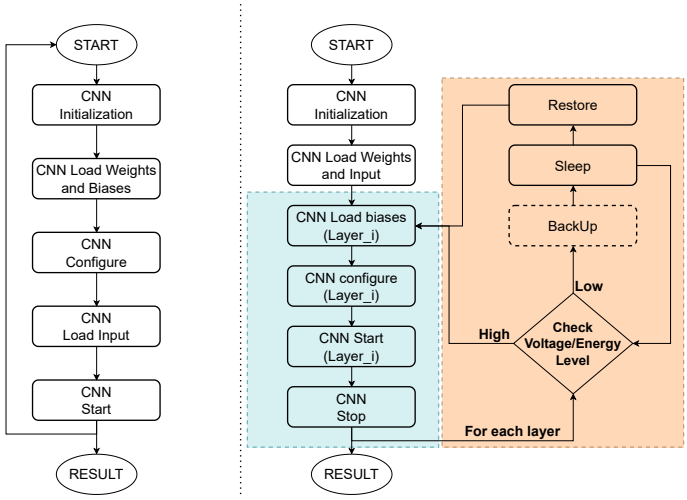


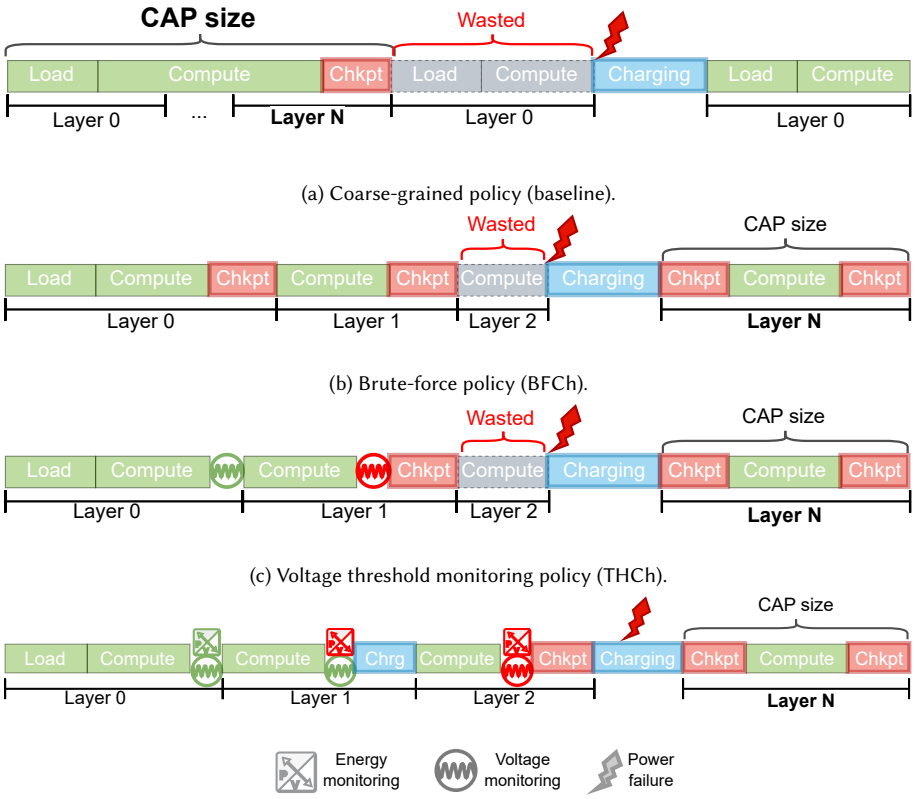
Fig. 5. CNN execution runtime flow comparison. On the left, the standard approach. On the right, the proposed LbLTT combines layer-by-layer evaluation (blue square) and energy monitoring (orange square). The backup operation is executed only one time after the end of a layer if the remaining energy is too low for evaluating the next layer.

the classic (i.e., the whole network) implementation is presented, while the proposed LbLTT implementation is on the right. After the CNN initialization, input data for the first layer is loaded into the accelerator memory. Then, the biases for the first layer are loaded, and the accelerator is configured and started. These steps are repeated for each layer of the original NN model. This loop makes the inference execution fine-grained and opens access to the inter-layer computational results that can be backed up in the external FRAM in case of a power failure. Notice that the weights of the network are loaded all together at the beginning of the execution, while the biases are loaded on a per-layer basis. This was possible thanks to configuring the parameters of the original *ai8xize.py* script that allows specifying a memory offset for the biases. This is necessary because each single layer is considered as a standalone neural network by the Maxim toolchain, leading to a memory overwrite problem. In fact, the original script maps all the NNs starting from a fixed address (e.g., 0x0000). However, the script does not provide the option to specify a memory offset where to store weights, thus the need to load them all at once. Anyhow, it is worth noting that the weights' magnitude is smaller than the biases, so reloading weights at each layer does not add any significant overhead.

3.3 LbLTT Backup Policies

Without LbLTT, the only option for intermittent computing on the CNN accelerator is backing up the final result of the entire inference. As shown in Figure 6a, such coarse-grained checkpoint policy requires a relatively large energy buffer to ensure computational progress and suffers from a significant amount of wasted time and energy if a power failure interrupts the inference.

Applying LbLTT enables two other backup policies used in modern intermittent systems. (i) The first one is a brute-force (*BFCh*, Figure 6b) solution, when the checkpoint is performed at the end of each layer's computation, avoiding the necessity of additional hardware for voltage level monitoring and execution profiling. Furthermore, this policy can guarantee execution progress with a smaller capacitor. However, the cost of this backup strategy is redundant memory access and lost computational results if a power outage occurs during the layer execution. (ii) The second



(a) Coarse-grained policy (baseline).

(b) Brute-force policy (BFCh).

(c) Voltage threshold monitoring policy (THCh).

(d) Capacitor energy monitoring policy (CMPCh).

Fig. 6. Overview of backup policies. We refer to *CAP size* as the energy-wise dimension of the energy storage used.

solution for backups is setting a voltage threshold for checkpoints (*THCh*, Figure 6c). This strategy involves using dedicated hardware for incoming voltage monitoring at layer borders. If at the end of any layer execution, the incoming voltage reaches a specified threshold, the system backs up the intermediate result. Some unnecessary checkpoints are eliminated with such backups, but the system still loses computational results at mid-layer power failures. Moreover, voltage monitoring circuitry requires additional energy and area resources.

3.4 Eliminating Redundant Backups

The adoption of *THCh* still requires checkpoints unused by recovery, i.e., those after which no power failure appeared. These checkpoints happen in two cases when at the end of the layer execution, the capacitor voltage reaches a single specified threshold: (i) the input ambient power is strong enough to maintain the computation of the next layer; (ii) the energy currently stored in the capacitor is sufficient to continue the computation of the next layer without interrupts. We propose an optimized backup policy for intermittent layer-by-layer execution (*CMPCh*) by extending the functionality of the voltage monitoring circuitry of *THCh*. The proposed policy tries to keep an intermittent system alive, performing only inevitable checkpoints and eliminating unnecessary expensive external memory transfers. The energy monitoring circuit is composed of two main parts: (i) the internal voltage comparator of the MAX78000 MCU, and (ii) an analog

Table 1. Backup policies and corresponding states of the art.

Backup policy	Related state of the art
Baseline	Protean [4]
BFCh	Mementos [38], DINO [29], Alpaca [30], InK [44],
THCh	Hibernus [7], Hibernus++ [6], Samoyed [32], AdaMICA [2]
CMPCh	**This work**

frontend connected to the energy storage. The measured power contribution of the Low-Power Voltage comparators of the MCU is equal to around 15 nW, and thus considered negligible during the evaluation. On the other hand, the analog frontend, used to generate the voltages to be applied to comparators, comprises a voltage divider, a digital switch, and an operational amplifier. Thanks to the use of nano-power components, the overall power overhead of the analog front end is equal to 0.66 nW when the digital switch is off and to around 5 µW when the whole circuit is powered. This energy overhead was considered during the evaluation.

Figure 6d shows that at the end of each layer, *CMPCh* monitors the energy left in the capacitor and decides on checkpoints depending on the comparison against the energy consumed by the upcoming layer execution. The computation is continued with no checkpoint if the energy currently stored in the capacitor is enough to compute the next layer. Otherwise, the system transitions to an ultra-low-power (µW) mode retaining volatile memory content. While staying in µW mode, the system continues to charge the energy buffer. With input power equal to or greater than the µW mode power consumption, the *CMPCh* strategy allows a batteryless system to stay alive, maintaining the capacitor energy persistent or increasing, respectively. The system performs a checkpoint only when the voltage threshold reaches the lower value and resumes when the upper voltage threshold is reached. This strategy gains additional benefits from having voltage monitoring circuitry, reducing the number of checkpoints. Additionally, the *CMPCh* backup policy eliminates any power failures during computation and avoids data loss. Preliminary profiling of execution is necessary since the system must know in advance the energy consumption of each layer. This profiling introduces insignificant burdens with the proposed LbLTT. Multiple options for the backup strategy with LbLTT give flexibility to the developers of machine learning applications on the emerging MAX78000 device.

Table 1 compares the backup techniques described above to several states of the art. Protean [4] is the only work that proposes an intermittent multisensor platform based on MAX78000. The platform is decoupled with adaptive runtime that implements task-based intermittent computing. Only the entire inference in Protean's CNN accelerator can be treated as an atomic task with no intermediate results checkpoint. We consider this approach as a baseline. Once a CNN model is split into layers, any task-based checkpointing technique (e.g., Alpaca [30], InK [44]) can be applied to wrap the layers execution into separate atomic tasks. Another option is to manually define the checkpoint locations with the help of, for example, Mementos [38] or DINO [29] and explicitly back up at each layer complete. These approaches correspond to the BFCh policy. Finally, in Hibernus [7], Hibernus++ [6], Samoyed [32], and AdaMICA [2], the authors use a voltage monitoring mechanism similar to that of the THCh policy. In the preceding sections, our proposed backup policy, CMPCh, is compared against all others in the table.

4 EVALUATION AND RESULTS

We evaluated and characterized the proposed LbLTT toolchain by using three different CNN models provided by Maxim Integrated [23]: the `AI85Net5` NN model trained on **MNIST** dataset; the `AI85Net20` NN model trained on **KWS**

(Key Word Spotting) dataset; and the `ai85simplenetwide2x` NN model trained on the **CIFAR-100** dataset. We have developed custom inference routines to evaluate the LbLTT policies presented in Section 3. To this end, since the MAX78000 platform does not integrate an internal non-volatile FRAM for checkpointing, we used a 4 Mbit external FRAM connected through an SPI communication working at 20 Mbit/s.

4.1 Neural Inference Execution Time Evaluation

We evaluate the execution time of the two primary tasks, namely *Inference* and *Backup* using the **CIFAR-100**, **KWS**, and **MNIST** models. Table 2 shows the execution and backup time for each layer. Furthermore, for the sake of comparison of the execution time, the time breakdown for the **MNIST** model is also presented in Figure 7. Notice that the overhead introduced by the external FRAM dominates the execution time. External memory access can takes up to 27× more time than computation (e.g., Layer 0 in Figure 7), making the entire inference unreasonably long. Thus, we have decided to evaluate the backup time in the case the MAX78000 would have a *theoretical internal FRAM*. To this end, we timed the FRAM access in an MSP430FR5994 microcontroller [21] – one of the few MCUs with integrated FRAM – operating at 1 MHz, and then interpolated it to the 100 MHz, to match the operation frequency of the MAX78000 ARM core. The measurements are presented in the last column of Table 2, showing that internal FRAM access requires on average 4× less time than inference.

4.2 Power Consumption Evaluation

We analyzed the average current consumption of the MAX78000 operating in various states to assess the energy consumption of the proposed method. Table 3 presents the current consumption for the four main states: **1)** Idle; **2)** Normal-NN (uninterrupted inference); **3)** LbL-NN (layer-by-layer inference); and **4)** External FRAM data backup.

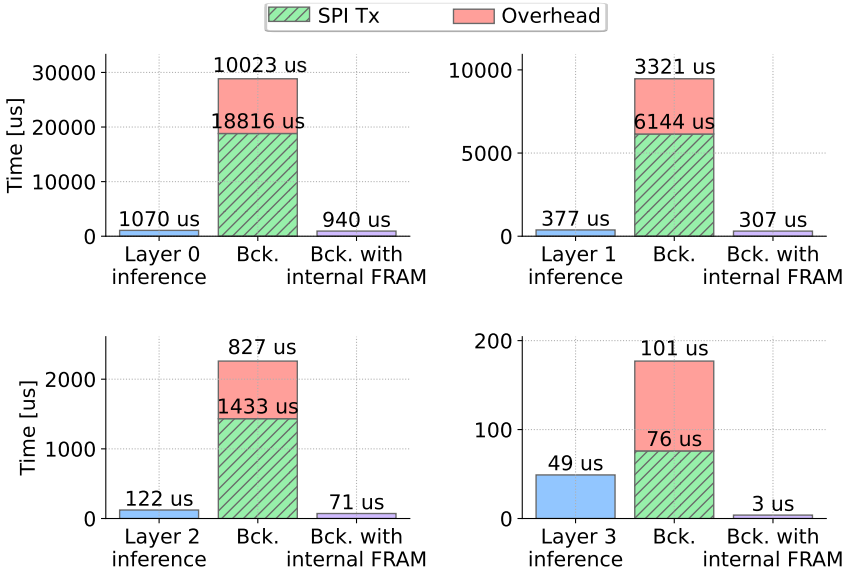


Fig. 7. Task time breakdown for the MNIST dataset. The first column presents per-layer inference time. The second column shows the time associated with the SPI FRAM operation. The third column is the calculated backup time in the case of internal FRAM.

Table 2. Tasks execution time breakdown for the thee CNN models evaluated. The last column on the right presents the backup time in the case of internal FRAM

Network	Layer	Inf. Time [µs]	Backup time [µs]			
			External FRAM			Internal FRAM
			SPI TX	Overhead	TOT	(theoretical)
KWS	Conv	1697	5120	10081	15201	160
	Conv	541	4838	9486	14324	151
	Conv + Pool	216	1612	3206	4818	50
	Conv	106	1171	2344	3515	36
	Conv + Pool	87	768	1559	2327	24
	Conv	101	1075	2146	3221	33
	Conv + Pool	107	560	1146	1706	17
	Conv + Pool	60	102	256	358	3
CIFAR100	Conv	853	9830	9830	19660	307
	Conv	759	13107	13107	26214	409
	Conv	759	13107	13107	26214	409
	Conv	759	13107	13107	26214	409
	Conv + Pool	248	3276	3276	6552	102
	Conv	323	3276	3276	6552	102
	Conv	404	6553	6553	13106	204
	Conv + Pool	156	1638	1638	3276	51
	Conv	149	1638	1638	3276	51
	Conv + Pool	126	819	819	1638	25
	Conv + Pool	267	819	307	1126	25
	Conv	224	307	76	383	9
	Conv + Pool	112	76	76	152	2
MNIST	Conv	1070	18816	10023	28839	940
	Conv	377	6144	3321	9465	307
	Conv	122	1433	827	2260	71
	Conv	49	76	101	177	3

Table 3. Current consumption for the four main tasks

	Idle	Normal-NN	LbL-NN	FRAM Backup
Current [mA]	12.11	22.27	21.13	16.11

Notably, the current consumption of LbL-NN is lower than that of Normal-NN. The reason for this reduction is two-fold: (i) the interruptions between layers pause the energy consumption of the CNN accelerator for inter-layer management, and (ii) for different layers, a varying number of the processors of the CNN accelerator is activated.

4.3 Backup Policy Evaluation

The energy consumption profiles are also evaluated using the LbLTT backup techniques outlined in Section 3. As a *baseline*, we consider the entire inference process (i.e., the evaluation of all layers without interruption) with checkpoints

Table 4. Comparison between baseline and the proposed LbLTT policies. Experiments are based on the *night* power trace from [18] evaluating the MNIST model.

	Baseline	BFCh	THCh	CMPCh
Number of resets	37386	37130	37345	2351
Number of restores	0	5582	2818	115

Table 5. Comparison between executed inferences and number of executed backups for the MNIST model using the night power trace from [18]. In the last two lines, we present the normalized number of inferences and the normalized number of backups w.r.t baseline.

	Baseline	BFCh	THCh	CMPCh
Number of Inferences	79254	47809	75077	147802
Number of Backups	79254	239045	104430	342306
Normalized Inferences	1	0.65	0.94	1.86
Normalized #Backups	1	5	1.39	2.31

only executed at the end of each inference (Figure 6a). Three distinct and realistic power traces [18] are used to assess the platform under varying input power levels: *car ride* (primarily high power), *daylight activity* (alternating between high, medium, and low power), and *night* (primarily low power).

4.3.1 ***Simulation Results.*** To perform a controllable and repeatable evaluation of an intermittent CNN accelerator, we developed a simulator of the target device, using the parameters presented in Sections 4.1 and 4.2. We fix the capacitor size to 250 µF so that the energy stored is enough to execute one complete inference (including input load, computation, and results storing) for the MNIST dataset.

As discussed in Section 3, the *THCh* and *CMPCh* policies exploit voltage thresholds to trigger specific operations. In the case of the *THCh* policy, the threshold is used to determine whether to back up the result of a layer. In the simulation, this threshold is set to 2.1V and the layer is backed up if the energy storage voltage falls below this value at the end of the inference, regardless of its layer size. The *CMPCh* policy uses two different thresholds: one for triggering the inference on a layer and another for determining whether to perform a backup. For the backup threshold, we also set 2.1V, while using a dynamic threshold (depending on the layer size) for the beginning of different layers. By considering the size and the expected energy consumption of each layer and the remaining energy in the capacitor, the simulation can calculate whether the inference can be completed. Furthermore, the energy for backup is also included in the budget to ensure that no information is lost. This approach allows the device to remain in a low-power consumption mode until the capacitor has been sufficiently charged to carry out the inference safely.

As can be noted in Table 4, the proposed *CMPCh* backup strategy reduces the number of system resets caused by power failures. Compared to the other checkpoint policies, we can achieve up to 16× fewer resets. We have also compared the number of inter-layer restores. The baseline solution presents no restoration operations. Power failures in this case force the re-execution of the entire inference starting from the input layer. The highest number of restores are executed with the *BFCh* solution, while the voltage monitoring mechanism in *THCh* helps to reduce this number almost twice. Finally, the energy-aware *CMPCh* strategy further reduces the number of restores, achieving 48× fewer restores than *BFCh*.

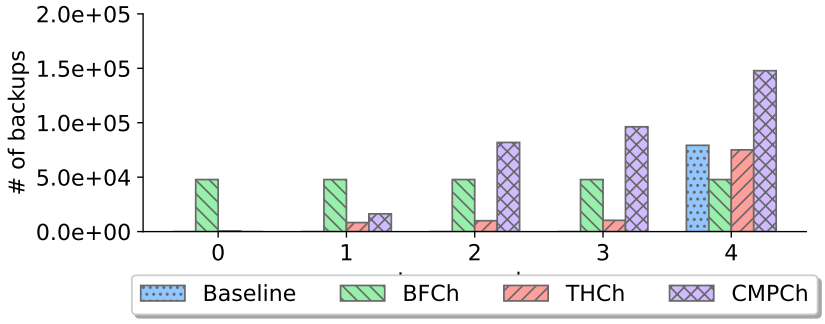
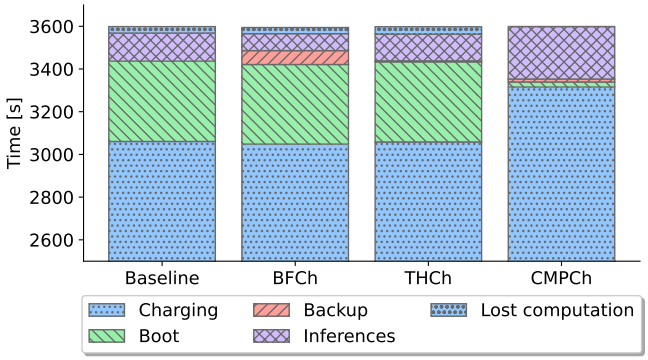
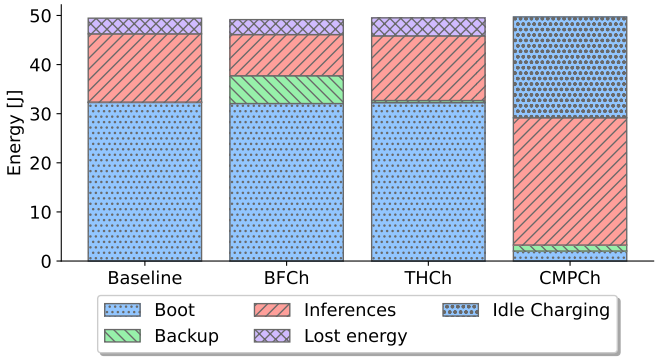


Fig. 8. Number of backups executed among the different layers of the MNIST model.



(a) Time cost distribution.



(b) Energy cost distribution.

Fig. 9. Time and energy overheads distribution for different backup policies.

We have thus analyzed the number of checkpoints executed. Figure 8 presents the distribution of the checkpoints among CNN layers for the MNIST model. Notice that the baseline policy only saves the result of the last layer due to the uninterrupted inference. *BFCh* checkpoints at the end of each layer, while *THCh* skips some unnecessary checkpoints thanks to the voltage monitoring mechanism. *CMPCh* totally avoids checkpoints for the first layer and only backs up the intermediate results to avoid unnecessary backup operations. To further evaluate the proposed policies, Table 5 presents the total number of inferences achieved using the night power trace. The baseline policy performs only one backup per inference, while *BFCh* performs 5× more checkpoints. *THCh* and *CMPCh*, on average, execute 1.4 and 2.3 backups per inference, respectively. Even if *CMPCh* executes one more backup per inference than that of *THCh*, the throughput of the proposed *CMPCh* outperforms three others, achieving 1.9× more inferences. Compared to *BFCh* and *THCh*, *CMPCh* executes 3.1× and 2× faster, respectively.

The improvement of the throughput introduced by the *CMPCh* strategy is due to the ability to exploit better the available energy. As can be seen in Figure 9a that presents the time cost distribution for the main tasks, *Baseline*, *BFCh*, and *THCh* spend most of the time for recharging and booting operations. Since we do not have a control mechanism for the available energy, the platform cannot complete the booting phase when the harvested energy during the charging phase is insufficient. On the contrary, *CMPCh* spends more time on recharging, waiting to have collected enough energy before starting the operations. Comparing the energy contribution of the different working phases, presented in Figure 9b, we can see that *CMPCh* allows the allocation of available energy better. More than half of the total energy is used for inference, while 35% of energy is used for keeping the device in a low-power mode during the charging phase, avoiding expensive booting and backup operations.

4.3.2 ***Real-world Implementation Results.*** We have replicated the simulation using real hardware to validate the effectiveness of the proposed toolchain. Two different applications were implemented. Figure 10 presents the experimental setup. It uses a MAX78000FTHR Evaluation Board, an external SPI FRAM, a supercapacitor used as an energy storage, and an *Analog Discovery 2* (AD2) used as a programmable source and measurement unit. The AD2 was used to emulate the voltage traces presented in [18], to provide the energy storage status of charge to the MAX78000, and to analyze and save the results. For both tests, we used the same amount of energy and time windows of the simulation, respectively equal to 50 J and 60 minutes.

MNIST. In this experiment, we have characterized the platform by implementing the MNIST network. We used the power traces of the simulations and the low-power comparator built into the MAX78000 to trigger the different operations. The comparison between the different backup policies is presented in Figure 11. Notice that in this experiment, we adopted the policy named *CMPCh**. We used a fixed energy threshold to schedule the various operations because of the implementation constraints of the evaluation board. This threshold was selected considering the worst-case scenario, the energy needed for the biggest atomic operation. On the contrary, *CMPCh* dynamic policy adapts the threshold based on the energy needed to complete the current task. This allows a better allocation of the available energy, leading to more inferences. This approach makes *CMPCh* slightly more efficient. Nevertheless, the real test confirmed the validity of the proposed *CMPCh* policy, achieving a 1.2× improvement in the number of inferences.

VISUAL SURVEILLANCE SYSTEM. In the second test, we implemented an autonomous visual sensing system trained on the CIFAR100 dataset. Although the setup is the same already presented in Figure 10, we have also used images from the integrated RGB camera of the MAX78000FTHR board. The system was configured to take a photo, downsample it to 32×32 pixels and then process the information using the integrated neural accelerator. The result of this experiment is presented in Figure 12. We highlight that the number of inferences is much lower than in the previous tests because

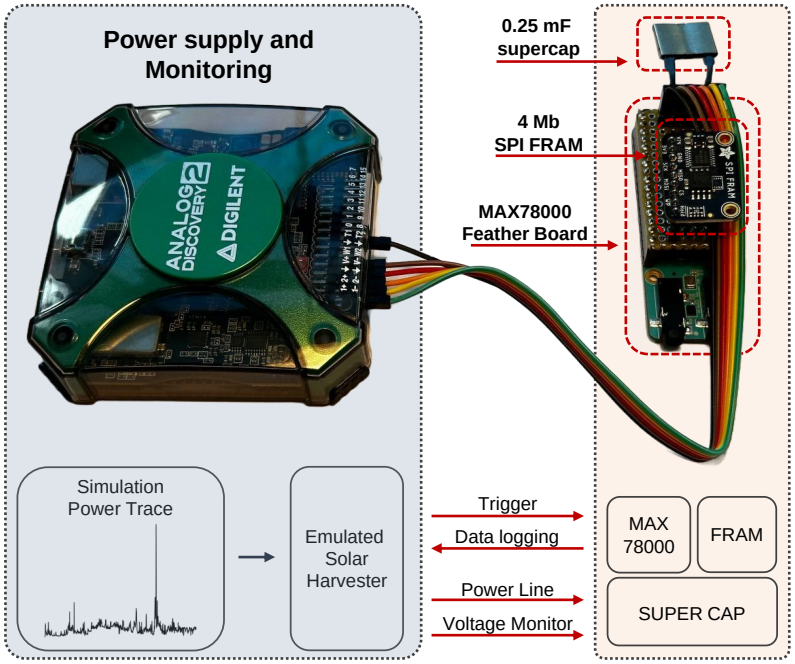


Fig. 10. Real hardware implementation test-bed. On the left, the Analog Discovery is used for emulating the energy harvester and for logging data. On the right, the MAX78000 Feather board with on top the SPI FRAM and the super-capacitor-based energy storage.

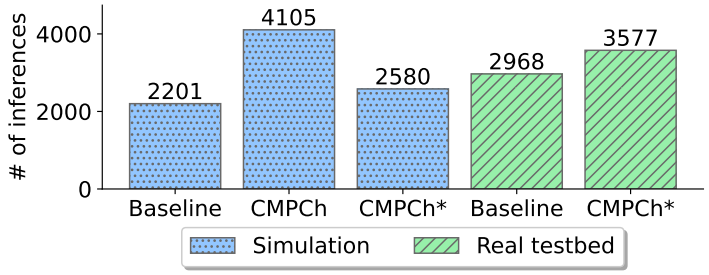


Fig. 11. Throughput comparison between simulation and real-world implementation. *Indicates the policy with a static threshold for backup and inferences.

of the energy consumption of the RGB sensor, which dominates the energy budget of the application. We have thus less energy for computing tasks. Still, the proposed *LbLTT* can provide a 1.16× time improvement in the number of inferences, proving the validity of the approach.

4.4 Discussion

For evaluating the proposed toolchain, three distinct real-world power traces were used. The traces are derived from [18], a dataset of radiant light energy measurements collected by Columbia University's EnHANTs (Energy Harvesting

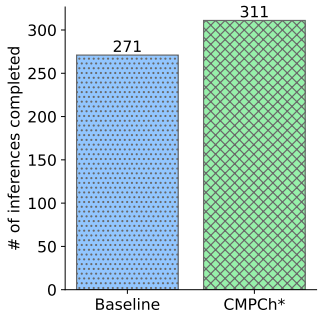


Fig. 12. Number of inferences comparison between the baseline (no LbLTT) and the proposed *CMPCh**. *Indicates the policy with a static threshold for backup and inferences.

Active Networked Tags) project. The authors conducted their study in New York City office buildings to assess energy availability. During the study, they collected long-term measurements of irradiance in several indoor locations and a set of shorter-term indoor/outdoor mobile device measurements. By using this dataset, it was possible to evaluate the proposed toolchain using a real-world power trace. As can be noted from the simulation and the real-world evaluations, the proposed toolchain can improve the number of achievable inferences by 1.16× to 1.9×. As highlighted by Figure 9, this improvement is possible thanks to a better energy allocation for the different tasks executed by the platform. The improvements are thus not limited to throughput increase. In fact, it allows the achievement of different goals that would otherwise not be possible. Focusing on the transient and sustainable computing scenario, the throughput increase can be traded with the following aspects:

- **Sensor size.** Allowing a higher number of inferences means that we can achieve the same results with a smaller and more compact sensor that has a more compact harvesting subsystem (i.e., solar, thermal, or vibration). For instance, in a solar-powered solution, the solar panel area and the size of the supercapacitor can be reduced, making the whole device smaller and cheaper. Moreover, having a smaller energy storage element means a faster cold start since we need to harvest less energy to reach the operating voltage level. This feature also opens the ability to exploit nano-power energy sources that could not charge up big energy storage elements.
- **Application Execution.** In scenarios with severely limited energy budgets, the available energy may not be sufficient to complete a single inference, as the mean harvested energy alone is typically inadequate to sustain the entire computation. When relying solely on fluctuating power sources, it becomes crucial to effectively utilize the typically short periods of relatively stable input power for computation. In this case, LbLTT enables the complete and correct execution of the application, which is unfeasible with the other approaches.

To better highlight the improvements brought by LbLTT, Figure 13 presents a comparison between the number of inferences using the baseline approach, and the proposed *CMPCh* backup strategy, for two different portions of power traces from [18]. Note that LbLTT allows the completion of the inference when the baseline approach fails due to the extremely low energy input.

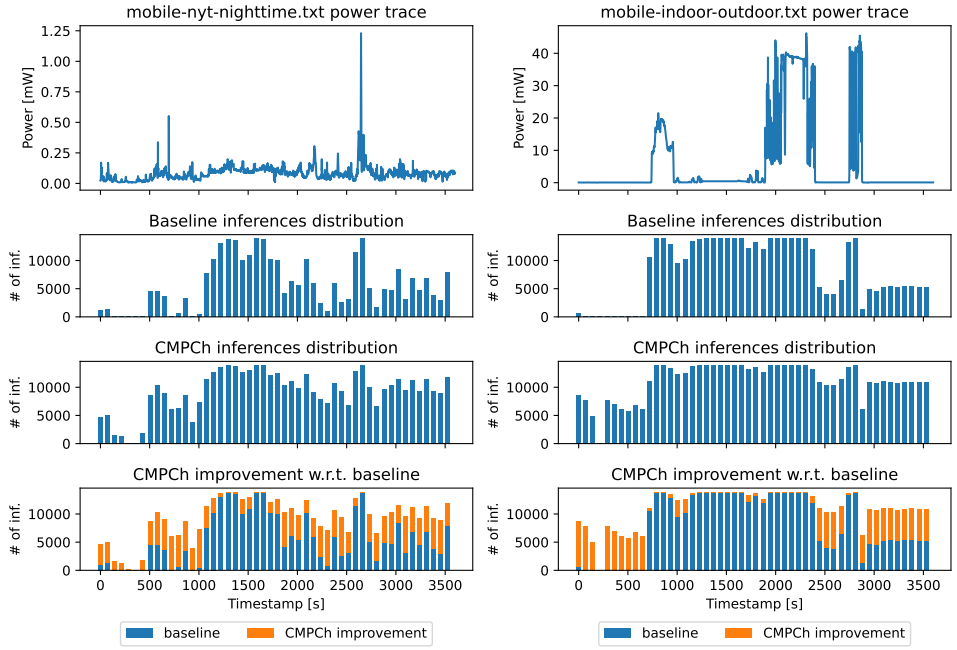


Fig. 13. Throughput comparison using the CIFAR-100

5 CONCLUSION AND FUTURE WORKS

In this paper, we presented *LbLTT*, a new toolchain for improving the intermittent execution of deep neural networks on recent convolutional neural network accelerators. We introduced, explored, and evaluated by means of time, energy, and implementation costs different layer-by-layer backup strategies in non-volatile memory to progress inference despite power failures. The results showed that the proposed solution increases the inference throughput by 1.9× for simulation and by 1.2× for real-world setup compared to the baseline execution.

In future research, we will focus on studying more complex deep neural networks to expand the scope of our proposed technique, which should significantly increase the achievable number of inferences on intermittent platforms. Additionally, we plan to enhance our toolchain by considering Hardware-Aware Neural Architecture Search (HA-NAS) [8, 10]. Specifically, we aim to generalize and integrate LbLTT functionalities as parameters in state-of-the-art HA-NAS approaches. Furthermore, we will investigate the integration of non-volatile memory and CNN accelerators to speed up backup and recovery operations.

ACKNOWLEDGMENTS

This work was partially supported by iNEST (interconnected NordEst innovation Ecosystem, Project ID: ECS00000043) PNRR project (Mission 4.2, Investment 1.5) Spoke 3 “Green and digital transition for advanced manufacturing technology”, funded by the European Commission under the NextGeneration EU programme.
Further, this work has been supported by the GEMINI (“Green Machine Learning for the IoT”) national research project, funded by the Italian MUR under the PRIN 2022 programme (Contract 20223M4HZ4).

REFERENCES

[1] JunIck Ahn, Daeyong Kim, Rhan Ha, and Hojung Cha. 2023. Controlling Action Space of Reinforcement Learning-based Energy Management in Batteryless Applications. *IEEE Internet of Things Journal* (2023), 1–1. https://doi.org/10.1109/JIOT.2023.3234905

[2] 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.

[3] Andrea Albanese, Matteo Nardello, Gianluca Fiacco, and Davide Brunelli. 2023. Tiny Machine Learning for High Accuracy Product Quality Inspection. *IEEE Sensors Journal* 23, 2 (2023), 1575–1583. https://doi.org/10.1109/JSEN.2022.3225227

[4] 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.

[5] 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.

[6] Domenico Balsamo, Alex S Weddell, Anup Das, Alberto Rodriguez Arreola, Davide Brunelli, Bashir M Al-Hashimi, Geoff V Merrett, and Luca Benini. 2016. Hibernus++: a self-calibrating and adaptive system for transiently-powered embedded devices. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 35, 12 (2016), 1968–1980.

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

[8] Hadjer Benmeziane, Kaoutar El Maghraoui, Hamza Ouarnoughi, Smail Niar, Martin Wistuba, and Naigang Wang. 2021. A comprehensive survey on hardware-aware neural architecture search. *arXiv preprint arXiv:2101.09336* (2021).

[9] Naveed Anwar Bhatti and Luca Mottola. 2017. HarvOS: Efficient code instrumentation for transiently-powered embedded sensing. In *2017 16th ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN)*. IEEE, 209–220.

[10] Han Cai, Ligeng Zhu, and Song Han. 2018. Proxylessnas: Direct neural architecture search on target task and hardware. *arXiv preprint arXiv:1812.00332* (2018).

[11] Jongouk Choi, Larry Kittinger, Qingrui Liu, and Changhee Jung. 2022. Compiler-directed high-performance intermittent computation with power failure immunity. In *2022 IEEE 28th Real-Time and Embedded Technology and Applications Symposium (RTAS)*. IEEE, 40–54.

[12] Alexei Colin and Brandon Lucia. 2016. Chain: Tasks and Channels for Reliable Intermittent Programs. In *Proc. OOPSLA*. ACM, Amsterdam, Netherlands, 514–530.

[13] Shuiguang Deng, Hailiang Zhao, Weijia Fang, Jianwei Yin, Schahram Dustdar, and Albert Y. Zomaya. 2020. Edge Intelligence: The Confluence of Edge Computing and Artificial Intelligence. *IEEE Internet of Things Journal* 7, 8 (2020), 7457–7469. https://doi.org/10.1109/JIOT.2020.2984887

[14] Harsh Desai, Matteo Nardello, Davide Brunelli, and Brandon Lucia. 2022. Camaroptera: A Long-Range Image Sensor with Local Inference for Remote Sensing Applications. *ACM Trans. Embed. Comput. Syst.* 21, 3, Article 32 (may 2022), 25 pages. https://doi.org/10.1145/3510850

[15] Cağlar Durmaz, Kasim Sinan Yıldırım, and Geylani Kardas. 2022. Virtualizing Intermittent Computing. *IEEE Internet of Things Journal* 9, 21 (2022), 20869–20878. https://doi.org/10.1109/JIOT.2022.3176587

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

[17] 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.

[18] Maria Gorlatova, Michael Zapas, Enlin Xu, Matthias Bahlke, Ioannis (John) Kymissis, and Gil Zussman. 2022. CRAWDAD columbia/enhants. (2022). https://doi.org/10.15783/C77P47

[19] Matthew Hicks. 2017. Clank: Architectural Support for Intermittent Computation *(ISCA '17)*. Association for Computing Machinery, New York, NY, USA, 228–240. https://doi.org/10.1145/3079856.3080238

[20] Texas Instruments. 2023. *FRAM FAQs*. Retrieved February 16, 2023 from https://www.ti.com/lit/pdf/slat151

[21] Texas Instruments. 2023. MSP430FR5969 LaunchPad Development Kit. Retrieved February 16, 2023 from http://www.ti.com/tool/MSP-EXP430FR5969

[22] Maxim Integrated. 2023. *Artificial Intelligence Microcontroller with Ultra-Low-Power Convolutional Neural Network Accelerator*. Retrieved July 03, 2023 from https://www.analog.com/en/products/max78000.html#product-overview

[23] Maxim Integrated. 2023. *Maxim Integrated AI Development*. Retrieved July 03, 2023 from https://github.com/MaximIntegratedAI

[24] Tianyu Jia, Yuhao Ju, Russ Joseph, and Jie Gu. 2020. Ncpu: An embedded neural cpu architecture on resource-constrained low power devices for real-time end-to-end performance. In *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 1097–1109.

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

[26] Vito Kortbeek, Kasim Sinan Yildirim, Abu Bakar, Jacob Sorber, Josiah Hester, and Przemysław Pawełczak. 2020. Time-sensitive Intermittent Computing Meets Legacy Software. In *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*. 85–99.

[27] Seulki Lee, Bashima Islam, Yubo Luo, and Shahriar Nirjon. 2019. Intermittent learning: On-device machine learning on intermittently powered system. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies* 3, 4 (2019), 1–30.

[28] 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.

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

[30] Kiwan Maeng, Alexei Colin, and Brandon Lucia. 2017. Alpaca: Intermittent execution without checkpoints. *Proceedings of the ACM on Programming Languages* 1, OOPSLA (2017), 96.

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

[32] Kiwan Maeng and Brandon Lucia. 2019. Supporting peripherals in intermittent systems with just-in-time checkpoints. In *Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation*. 1101–1116.

[33] Vicent Sanz Marco, Ben Taylor, Zheng Wang, and Yehia Elkhatib. 2020. Optimizing deep learning inference on embedded systems through adaptive model selection. *ACM Transactions on Embedded Computing Systems (TECS)* 19, 1 (2020), 1–28.

[34] Azalia Mirhoseini, Ebrahim M. Songhori, and Farinaz Koushanfar. 2013. Idetic: A high-level synthesis approach for enabling long computations on transiently-powered ASICs. In *2013 IEEE International Conference on Pervasive Computing and Communications (PerCom)*. 216–224. https://doi.org/10.1109/PerCom.2013.6526735

[35] Arthur Moss, Hyunjong Lee, Lei Xun, Chulhong Min, Fahim Kawsar, and Alessandro Montanari. 2023. Ultra-Low Power DNN Accelerators for IoT: Resource Characterization of the MAX78000. In *Proceedings of the 20th ACM Conference on Embedded Networked Sensor Systems* (Boston, Massachusetts) *(SenSys '22)*. Association for Computing Machinery, New York, NY, USA, 934–940.

[36] Matteo Nardello, Harsh Desai, Davide Brunelli, and Brandon Lucia. 2019. Camaroptera: A Batteryless Long-Range Remote Visual Sensing System. In *Proceedings of the 7th International Workshop on Energy Harvesting & Energy-Neutral Sensing Systems* (New York, NY, USA) *(ENSsys'19)*. Association for Computing Machinery, New York, NY, USA, 8–14. https://doi.org/10.1145/3362053.3363491

[37] Sebin Shaji Philip, Roberto Passerone, Kasim Sinan Yildirim, and Davide Brunelli. 2023. Intermittent Computing Emulation of Ultralow-Power Processors: Evaluation of Backup Strategies for RISC-V. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 42, 1 (2023), 82–94. https://doi.org/10.1109/TCAD.2022.3169108

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

[39] 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.

[40] Alberto Rodriguez Arreola, Domenico Balsamo, Anup K. Das, Alex S. Weddell, Davide Brunelli, Bashir M. Al-Hashimi, and Geoff V. Merrett. 2015. Approaches to Transient Computing for Energy Harvesting Systems: A Quantitative Evaluation. In *Proceedings of the 3rd International Workshop on Energy Harvesting & Energy Neutral Sensing Systems* (Seoul, South Korea) *(ENSsys '15)*. Association for Computing Machinery, New York, NY, USA, 3–8. https://doi.org/10.1145/2820645.2820652

[41] Simone Ruffini, Luca Caronti, Kasım Sinan Yıldırım, and Davide Brunelli. 2022. NORM: An FPGA-Based Non-Volatile Memory Emulation Framework for Intermittent Computing. *J. Emerg. Technol. Comput. Syst.* 18, 4, Article 73 (oct 2022), 18 pages. https://doi.org/10.1145/3517812

[42] Muhammad Moid Sandhu, Sara Khalifa, Raja Jurdak, and Marius Portmann. 2021. Task Scheduling for Energy-Harvesting-Based IoT: A Survey and Critical Analysis. *IEEE Internet of Things Journal* 8, 18 (2021), 13825–13848. https://doi.org/10.1109/JIOT.2021.3086186

[43] Syntiant. 2022. Syntiant's Tiny Machine Learning Development Board. https://www.syntiant.com/_files/ugd/799fc8_856de1ac190c425c95dcde276f3752cb.pdf

[44] 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.

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

[46] Xinyi Zhang, Clay Patterson, Yongpan Liu, Chengmo Yang, Jingtong Hu, and Chun Jason Xue. 2020. Low Overhead Online Data Flow Tracking for Intermittently Powered Non-Volatile FPGAs. 16, 3, Article 26 (jul 2020), 20 pages. https://doi.org/10.1145/3371392

[47] Xinyi Zhang, Clay Patterson, Yongpan Liu, Chengmo Yang, Chun Jason Xue, and Jingtiong Hu. 2018. Low Overhead Online Checkpoint for Intermittently Powered Non-volatile FPGAs. In *2018 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*. 238–244. https://doi.org/10.1109/ISVLSI.2018.00052

Received 17 February 2023; revised 28 April 2023; accepted 5 June 2023