MII: A Multifaceted Framework for Intermittence-Aware Inference and Scheduling

Ziliang Zhang, Cong Liu, *Member, IEEE*, and Hyoseung Kim, *Member, IEEE*

***Abstract*—The concurrent execution of deep neural networks (DNNs) inference tasks on the intermittently-powered batteryless devices (IPDs) has recently garnered much attention due to its potential in a broad range of smart sensing applications. While the checkpointing mechanisms (CMs) provided by the state-of-the-art make this possible, scheduling inference tasks on IPDs is still a complex problem due to significant performance variations across the DNN layers and CM choices. This complexity is further accentuated by dynamic environmental conditions and inherent resource constraints of IPDs. To tackle these challenges, we present MII, a framework designed for the intermittence-aware inference and scheduling on IPDs. MII formulates the shutdown and live time functions of an IPD from profiling the data, which our offline intermittence-aware search scheme uses to find the optimal layer-wise CMs for each task. At runtime, MII enhances the job success rates by dynamically making scheduling decisions to mitigate the workload losses from the power interruptions and adjusting these CMs in response to the actual energy patterns. Our evaluation demonstrates the superiority of MII over the state-of-the-art. In controlled environments, MII achieves an average increase of 21% and 39% in successful jobs under the stable and dynamic energy patterns. In the real-world settings, MII achieves 33% and 24% more successful jobs indoors and outdoors.**

***Index Terms*—Embedded software, energy harvesting, real-time systems, tiny machine learning.**

I. Introduction

INTERMITTENTLY-POWERED batteryless devices (IPDs) offer a promising pathway to zero carbon emissions and maintenance-free operations. Recent advances have enabled them to execute the deep neural network (DNN) inference tasks [1], [2], [3], [4], essential for the smart sensing and IoT applications. These devices harvest ambient energy from the environment and store it in capacitors. Once sufficient energy accumulates, IPD executes tasks using this energy until depletion. IPDs are typically equipped with two types of memory: 1) volatile memory (VM), which is fast but loses the data upon shutdown and 2) nonvolatile memory (NVM), which is slow but retains the data after shutdown [5]. Since, an IPD turns on and off across power cycles, it must store intermediate computation results from VM to NVM before powering off [6], [7], [8], [9].

Existing research on IPDs primarily centers around checkpointing mechanisms (CMs) that preserve the execution progress across the power failures. Broadly, these mechanisms fall into two types: 1) just-in-time checkpointing (JIT) and 2) static checkpointing (ST) using the atomic blocks. JIT [7], [10], [11], [12] checkpoints the system state once at the end of each power cycle, achieving faster speeds but demanding a larger peak memory. On the other hand, ST [1], [3], [8], [9], [12], [13], [14] transforms the program code into smaller atomic blocks of various granularity (e.g., layers, filter, and tiles for DNNs), with the checkpointing code at the end of each block, offering a smaller peak memory but at the cost of speed (Section II-B).

Although the existing studies have laid the groundwork for executing the DNN inference tasks on IPDs, significant challenges persist for the real-world deployment. First, the layer-wise structural distinctions of DNNs lead to performance heterogeneity across the layers, demanding an optimal CM for each layer (Section III-A). However, the limited VM size and intermittent power of IPDs make this particularly challenging due to the inevitable device shutdowns experienced by some layers (the shutdown layers). Second, the real-world environments present varying energy patterns, resulting in different shutdown layers for the inference tasks at runtime. Consequently, CMs choices considered to be optimal for one environment may become the worst in another (Section III-B), necessitating a runtime adaption of CMs.

Contributions: We present MII: multifacted framework for intermittence-aware inference and scheduling. MII consists of two parts: 1) offline and 2) online. The offline phase addresses the first challenge which requires co-consideration of both the shutdown layers and peak memory usage. Our offline intermittence-aware search method identifies the optimal CM for each layer under a given environment so that each task's execution time is minimized and the memory constraint is met. The online phase addresses the second challenge, which requires a low-overhead algorithm that quickly captures the environment dynamics and makes adaptations accordingly. MII's online phase makes scheduling decisions dynamically, aligns the task execution with the power cycles, and adapts CMs according to the actual energy supply and usage patterns. MII also introduces a proactive shutdown feature to mitigate the wasted work problem in a mixed JIT and the ST system. Compared to the existing work, MII achieves an optimal

Manuscript received 6 August 2024; accepted 10 August 2024. This work was supported in part by the U.S. NSF under Grant CNS 1943265, Grant CPS 2230969, Grant CNS 2300525, Grant CNS 2343653, and Grant CNS 2312397; in part by the USDA/NIFA SCRI under Grant 2020-51181-32198; and in part by the IITP funded by the Korean government (MSIT) under Grant 2021-0-00360. This article was presented at the International Conference on Embedded Software (EMSOFT) 2024 and appeared as part of the ESWEEK-TCAD special issue. This article was recommended by Associate Editor S. Dailey. *(Corresponding author: Ziliang Zhang.)*

The authors are with the University of California, Riverside, CA 92521 USA (e-mail: zzhan357@ucr.edu; congl@ucr.edu; hyoseung@ucr.edu).

Digital Object Identifier 10.1109/TCAD.2024.3443710

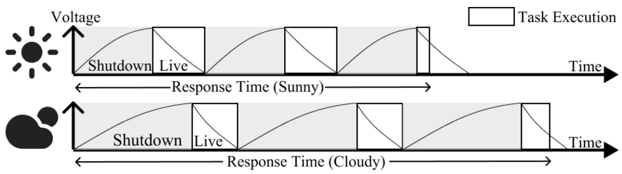


Fig. 1. Same task execution over shutdown and live times. Environment variation results in different response times.

execution of each inference task and adapts it to the runtime environment with its unique layer-wise CM design.

We implemented MII on an Apollo4 Blue Plus board and tested it against eight DNNs trained from the six datasets. We evaluate MII in both the controlled and real-world environments and compare it with the three state-of-the-art methods [2], [3], [10]. MII achieves an average increase of 21% and 39% in successful jobs than the other methods under stable energy patterns and dynamic energy patterns from the controlled environment. MII achieves 33% and 24% more successful jobs under indoor and outdoor real-world environments.

II. Background

A. IPD and Intermittent Inference

An IPD harvests energy from the ambient sources, e.g., solar, wind, radio waves, and vibration. Once sufficient energy is accumulated, the IPD turns on and begins the program execution during the *live time*. Although energy harvesting can continue during this phase, the device typically consumes energy at a faster rate than it accumulates. When the energy is depleted, the IPD powers down, waiting for enough energy to be harvested (*shutdown time*) before restarting this process [6], [7], [8], [9]. Fig. 1 depicts an example of the task execution over the live and shutdown times. Several prior studies [2], [5], [12], [15], [16], [17] have enabled multiprogramming and priority-based scheduling of tasks on IPDs, with a timekeeping ability across power cycles using either the MCU's deep sleep mode or external real-time clock (RTC). In this context, our focus is on the inference tasks that should run periodically for the practical use in areas, such as smart sensing, which periodically samples readings and runs inference tasks for the anomaly or object detection [1], [12], [18].

An intermittent inference task refers to a task executing the forward propagation of a DNN under the intermittent power because of the on-off power cycles, modern IPDs are equipped with both VM and NVM. VM is typically SRAM which is fast but small (tens to hundreds of KB in most MCUs) and loses all the data when powered off. NVM, such as MRAM and FRAM, is larger and slower compared to VM, but it can maintain data when powered off. To fully utilize the speed of VM in DNN inference, the existing work [3], [19] loads all the necessary data to VM, including the input features map (IFM), weights (WEI), and output features map (OFM), and then performs the computations using the VM data. Before shutdown, the calculated OFM from this power cycle is checkpointed to NVM, and once the device reboots, IPD resumes the remaining inference computations by fetching the checkpointed data to VM.

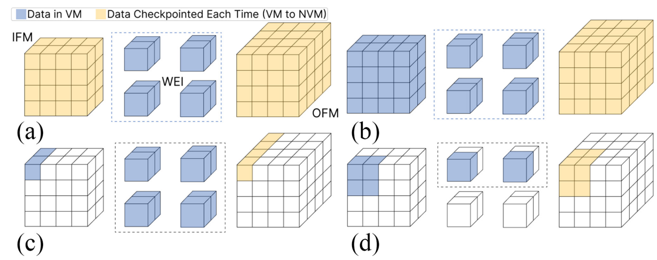


Fig. 2. Four types of CMs applicable to DNNs: blue is the data read back to VM and yellow is the data checkpointed each time. (a) JIT checkpointing. (b) ST-Layer (ST-L). (c) ST-Filter (ST-F). (d) ST-Tiled (ST-T).

DNN inferences keep a large memory footprint during the execution and have a magnitude more data than needed for checkpointing [1], [3], [19]. For example, a tiny seven-layer DNN performing a 32×32 pixel colored image classification needs to checkpoint 9216 output features to NVM for the largest layer, whereas the noninference tasks, such as thermometer sensing and alarm, only need to checkpoint less than ten outputs [9], [12], [14], [15], [16]. Despite the large memory footprint, loading all the corresponding data (including WEI) to VM during the inference is necessary, as it significantly reduces the NVM accesses and results in up to 51% less response time and 39% longer live time for the same seven-layer DNN compared to the direct read and write in NVM [1], [3].[1]

B. Checkpointing Mechanisms

State-of-the-art CMs fall into two categories: 1) JIT and 2) ST. JIT [7], [10], [11] makes a checkpoint of the entire system's states to NVM when the shutdown is imminent. The device's energy level, i.e., the capacitor voltage is constantly polled and compared with a predefined voltage threshold (the JIT threshold) that guarantees a successful checkpointing [11], [12]. When the capacitor voltage falls below the JIT threshold, JIT checkpoints the system states to NVM so that the IPD can be safely shut down without losing its progress [10]. Although JIT enjoys fast execution speed by checkpointing only once per power cycle, it demands a substantial amount of the memory. JIT needs to checkpoint both the IFM and OFM of the current layer since the previous checkpoint may not have saved the previous layer's OFM (the current layer's IFM). A detailed memory access pattern of JIT is shown in Fig. 2(a).

ST entails transforming the original task into the atomic blocks and performing a checkpointing at the end of each atomic block [1], [3], [9], [12], [13], [14]. If a shutdown occurs in the middle of a block, the IPD resumes from the last checkpoint upon reboot and re-executes the block. Since, any code with write-after-read (WAR) can disrupt idempotency, methods have been studied to construct the atomic blocks to guarantee the memory consistency and correct execution [8], [13], [20]. In the context of DNNs, an inference task can be divided into

[1] This holds for IPDs that run CPU and SRAM at higher clock rates than MRAM or FRAM, such as our platform. For others like MSP430, loading WEI may be considered optional.

the atomic blocks of various granularities shown in Fig. 2(b)–(d).[2]

1) *ST-L (Layer):* Due to its explicit IFM and OFM structures, each layer of a DNN can be naturally modeled as an atomic block, achieving ST at the layer-level granularity. ST-L is generally the fastest among all the STs, but it needs to load all the IFM, WEI, and OFM of that layer, resulting in the largest peak memory size among all the STs.
2) *ST-F (Filter):* In convolution layers, a filter is convolved across the IFM to compute a feature vector output. By rewriting each filter convolution into a separate atomic block, ST is attained at the filter granularity [1], [2]. ST-F is generally slower than ST-L due to its finer-grain block size; however, ST-F requires less memory than ST-L by loading partial IFM and OFM. Note that, ST-F still needs to load the entire WEI of the layer for its filter-wise computation.
3) *ST-T (Tile):* Inference can be further broken down by reorganizing into a tiled structure [3]. ST-T can be achieved by converting each tile's execution into an atomic block. Hence, unlike ST-F, ST-T can do computation with only a portion of WEI, thereby further reducing the peak memory with a potentially longer execution time.

C. Environmental Effects

In the real-world scenarios, the dynamics of the environment lead to significant changes in the energy harvesting rate, resulting in different response times for the same task on an IPD. Fig. 1 illustrates this phenomenon under the solar energy. Compared to the case of the sunny light conditions, the shutdown time under the cloudy conditions is obviously longer due to the lower harvesting rate. The live time is shorter because the device still harvests energy while it is executing the inference task but the harvesting rate is lower.

The variation of the environment is therefore the key challenge in scheduling tasks on an IPD. Existing work addresses this in two categories: 1) energy prediction and 2) workload reduction. Energy prediction methods [15], [16], [17] assume a priori knowledge of the future energy patterns or predict based on the previous patterns. However, the prediction can never be perfect due to the sporadic nature of the environment, and the use of more complex models increases overhead. Conversely, workload reduction [2], [14], [23], [24], [25] reacts to environmental changes by reducing the workload (e.g., skipping some layers of DNNs, called "early termination" [2]) as the harvesting rate reduces. Its limitations include the degradation in output quality, and the extra efforts and overhead to enable early termination. More importantly, in the DNN inference, the entire layer OFM has to be loaded in VM for an early-exit classifier or model to begin the execution [2]. This makes it unable to keep the peak memory usage smaller than the layer OFM, potentially limiting the IPD from running the multiple inference tasks.

D. System Model

We consider an IPD equipped with a fixed-size capacitor and a solar panel for energy harvesting, and with an RTC for timekeeping. The system has m periodic DNN inference tasks. Each task τ_i releases a job according to its period, and each job performs one inference of the task's DNN with the η_i layers. Due to the nature of intermittent computing, we do not aim to execute all the jobs of tasks; instead, we focus on maximizing the number of jobs that successfully complete their execution. If a job cannot finish before the start of the next period, it continues executing in the next period and the next period's job is skipped to prevent the overloads. The system may have other noninference tasks, e.g., the sensor and peripheral tasks, but they are not the main focus of this article and their CMs are statically determined as done in the prior work [12], [16], [17], [18], [24].

The system has V_{ipd} KB of memory in VM for the inference tasks. In practice, the DNN models for MCUs dynamically allocate and free memory from the heap space for each layer's execution. Thus, V_{ipd} essentially indicates the heap area size, and for each task, the layer that uses the most heap memory determines the peak memory usage of that task.

III. Motivation

A. Checkpointing Tradeoff on Intermittent Inference

To understand the effect of CMs on the execution time and memory usage of a DNN inference job, we set up an experiment evaluating both the JIT and all the granularities of ST (ST-L, ST-F, and ST-T) with three DNNs on the MNIST and CIFAR10 datasets. Each DNN name is given by "dataset name - # of layers." We chose the DNNs and datasets following the prior work [1], [2], [3] as they are used in the real-world applications like the wildlife monitoring. We tested the DNNs on an Apollo4 Blue Plus board due to its sufficient VM size to run these DNNs under all the four CMs.

We first applied each CM to the entire DNN, as done in the prior work [1], [2], [3] under the continuous power from the USB. As shown in the top of Fig. 3, JIT yields shorter execution times and consumes larger memory than ST, whereas ST uses a small memory size at the cost of longer execution time. This tradeoff occurs due to their inherent differences in checkpointing. JIT loads the entire layer's IFM, OFM, and weights to the VM during execution. This results in the peak memory size matching that of the largest layer within the system. On the contrary, ST only fetches a portion of the layer data as described in Fig. 2 so that it can maintain a small memory footprint. However, it needs to checkpoint the calculated results to NVM after each block. As the granularity of ST decreases from the layer level to the tile level, we observe an increase in inference latency and a decrease in peak memory usage.

We further break down the overall inference execution time into individual layers and characterize the layer-wise performance. Fig. 3 bottom shows the layer-wise relative

[2]This approach of leveraging the DNN structure is motivated by iNAS [3], which offers benefits over the general programming language-based [21] and compiler-assisted [22] methods by ensuring that the blocks fit into the device memory without requiring the extensive manual effort and code changes.

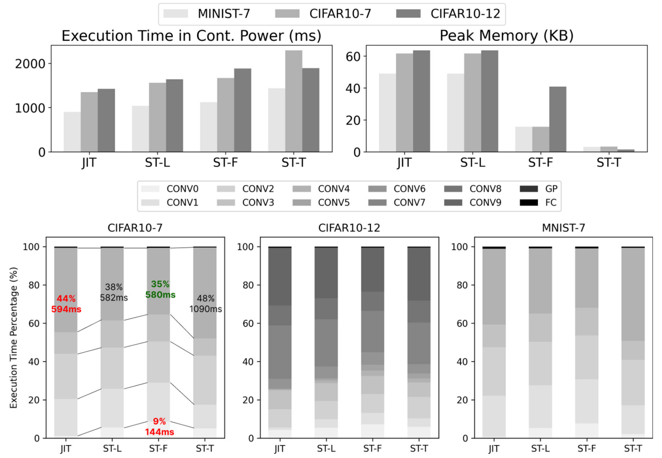


Fig. 3. Top: JIT versus STs of various granularity tradeoff space. Bottom: layer performance heterogeneity under JIT and STs.

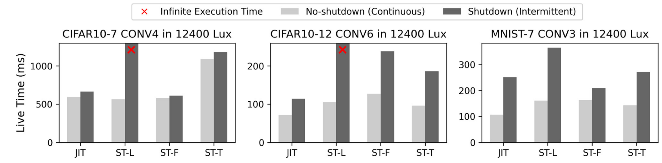
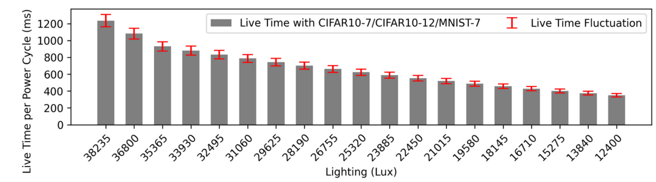


Fig. 4. Total live time of layers in the presence of shutdown.



CIFAR10-7 Shutdown Layers

CM	12400 Lux	38235 Lux
JIT	2,4	2,4
ST-L	1,2,3,4	3
ST-F	1,2,4	2
ST-T	1,2,4	2,4

CIFAR10-12 Shutdown Layers

CM	12400 Lux	38235 Lux
JIT	6	7,9
ST-L	2,3,6	6,8
ST-F	1,3,6,7,9	6,9
ST-T	2,6,7,8,9	7,9

MNIST-7 Shutdown Layers

CM	12400 Lux	38235 Lux
JIT	2,4	4
ST-L	2,3,4	4
ST-F	1,3,4	4,5
ST-T	2,3,4	2,4,5

Fig. 5. Top: live time per power cycle when running CIFAR10-7 in various light conditions (CIFAR10-12 and MNIST-7 have the same pattern). Bottom: the shutdown layers of all three DNNs in two light conditions.

execution time of the DNN inference under different CMs for all three DNNs. As depicted in the figure, each layer experiences a different execution time depending on the CM used, which is not consistent with the general expectation that JIT is faster than ST. For instance, when executing CONV4, which has a large IFM and small OFM, JIT takes longer execution time and yields worse performance compared to ST-F and ST-L (the left-most red text) because JIT has to checkpoint the entire IFM of CONV4. ST-F achieves the best performance (the top green text) due to the small OFM of CONV4, which reduces the checkpointing overhead of ST-F. However, if we choose ST-F as the CM across all the layers of the job, it gives the worst performance for the layer CONV0, which has a large OFM and small WEI (the bottom red text). This is because the large OFM of CONV0 results in ST-F having a greater checkpointing overhead than the other CMs.

Obs. 1: For DNN inference tasks, relying on a single CM, as done in the prior work may result in suboptimal performance, thereby requiring an optimal CM choice for each layer.

We therefore advocate a layer-wise adoption of different CMs. Although it may seem straightforward to choose the optimal CM for each layer, solving such a problem under an intermittent power condition is challenging. Fig. 4 compares the cumulative live time of each layer, i.e., the time that the device is live for inference and checkpointing, under the intermittent power with a 2 mF capacitor and 12 400 lux lighting condition. As depicted, the optimal CM choice under the continuous power ("no-shutdown" in the legend) becomes often suboptimal under the intermittent power which causes the device to shut down at least once ("shutdown"). Also, the optimal CM chosen under the continuous power can make a layer trapped in the endless loop of re-execution. For instance, ST-L is the optimal CM for the CONV4 layer of CIFAR10-7 under the continuous power. However, ST-L makes it unable to checkpoint before shutdown under the intermittent power resulting in infinite execution time.

Obs. 2: Due to shutdown, the optimal CM choice for each layer of an interference task under the intermittent power may diverge from the one made under the continuous power.

B. Intermittent Inference Under Environment Variations

To find out how the change in energy harvesting patterns affects the inference performance, we first focus on the live time of an IPD, which directly affects the response time of an inference task as discussed in Section II-C. We use the same three DNNs and run them each separately in various light conditions. The top part of Fig. 5 shows the average live time per power cycle when running the CIFAR10-7 model under different CMs and light conditions. From the results, we find that live time varies significantly with the light conditions, and that under the same light condition, the live time is only marginally affected by CMs and tasks. For example, under 38 325 lux lighting, the live time in each power cycle for all three DNNs is 1236ms. On the other hand, if we change the lighting to 29625 Lux, the live time per power cycle changes to 743 ms. Some fluctuations may be observed depending on the CMs or tasks, but they are within the 6% and 7% range of the live time.

To further explore the effect of CMs and environment conditions on the intermittent inference, we characterize the *shutdown layers*, which are the subset of layers of an inference job that experience the shutdowns during their execution. The tables at the bottom of Fig. 5 depict the shutdown layers of each DNN job in two representative light conditions. The shutdown layers of each DNN vary significantly with the CM used and the given light condition. Recall our discussion in Section III-A and with Fig. 4. The optimal CM when the layer does not shut down becomes often suboptimal or even the worst if a shutdown occurs for that layer. Vice versa, if we choose the optimal CM assuming that the layer always experiences a shutdown, such a CM will likely perform worse when there is no shutdown.

From the above two experiments on the live time and shutdown layers, the following observation can be made.

Obs. 3: While both live time and shutdown layers play significant roles in determining the optimal CMs, they can vary

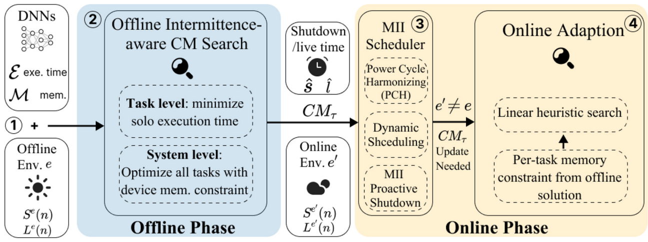


Fig. 6. Overview of the proposed MII framework.

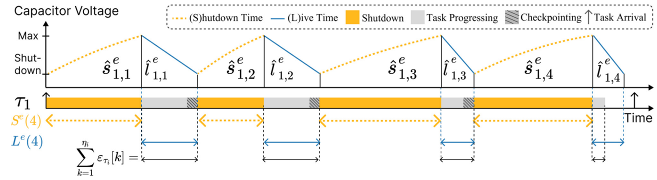


Fig. 7. Cumulative shutdown time $S^e(4)$ and live time $L^e(4)$ of an intermittent inference task τ_1.

drastically with environmental conditions. Consequently, there is a strong need for runtime adaptation with low overhead.

IV. MII Design Overview

Fig. 6 presents the overview of our **MII** framework, designed to address the two key challenges elaborated with our observations in Section III. We illustrate each challenge in a separate paragraph and propose the MII's solutions.

Motivated by Obs. 1 and 2, MII introduces an *offline phase* to tackle the challenge of finding a layer-wise optimal CM solution under a given environment, while considering the possible device shutdowns and device memory constraints. We opt for an offline solution because the profiling of the DNN execution time and memory footprint on a perlayer basis is feasible only in an offline setting. In ①, the offline phase models the energy supply pattern of a given environment condition e into the cumulative shutdown time $S^e(n)$ and cumulative live time $L^e(n)$ functions, where n is the number of power cycles. In ②, the offline intermittent-aware CM search finds the optimal CMs for the layers of each task to minimize their execution time while adhering to the device memory constraint.

Stemming from Obs. 3, actual energy supply patterns can vary drastically at runtime, which requires a timely adaption of CMs to cope with changing environmental conditions. Therefore, MII presents an *online phase* to tackle this challenge. The MII scheduler in ③ takes into account the CMs found at the offline phase and updates the offline $S^e(n)$ and $L^e(n)$ to their online versions, $S^{e'}(n)$ and $L^{e'}(n)$ based on the online shutdown and live times, and $\hat{s}$ and $\hat{l}$ collected using an on-board RTC. The scheduler harmonizes task execution with the power cycles, makes scheduling decisions based on the online energy pattern, and employs the proactive shutdown to mitigate the wasted work problem in a mixed JIT and the ST system. Finally, we introduce the online adaption method in ④, which adapts the CMs according to the latest $S^{e'}(n)$ and $L^{e'}(n)$ using a linear heuristic search for each scheduled inference task.

V. Offline Phase

A. Modeling Shutdown and Live Time Patterns

Before conducting the offline CM search, we model the energy supply pattern of a given environmental condition e based on the profiling and formulate it into the shutdown and live time functions, $S^e(n)$ and $L^e(n)$.

Recall that the execution of a job of an inference task τ_i can take multiple power cycles as illustrated in Fig. 7. We denote the shutdown time of τ_i during the jth power cycle in an environment condition e as $\hat{s}^e_{i,j}$, and the live time as $\hat{l}^e_{i,j}$. During each $\hat{s}^e_{i,j}$, the IPD remains off, only harvesting energy. Conversely, during each $\hat{l}^e_{i,j}$, the IPD powers on and starts consuming the capacitor's energy to execute τ_i while still harvesting energy. Given the IPD's fixed capacitor size, we make the assumption A1 about the shutdown time as below.

A1: The duration of shutdown in the jth power cycle is solely affected by the environment condition e, not by any task on the IPD. Hence, for ease of presentation, we use $\hat{s}^e_j$ to denote the shutdown time for the jth power cycle.

This is a valid assumption because at the beginning of the jth power cycle, the voltage level of the IPD's capacitor is at the power-off voltage regardless of the type of tasks executed in the previous power cycle, and the IPD turns on only when it reaches the power-on voltage, the timing of which is affected by the energy harvesting rate. Hence, with A1 and the profiled $\hat{s}^e_j$ data, we can represent the shutdown time as a function of the number of power cycles.

Definition 1 (Cumulative Shutdown Time): $S^e(n)$ gives the cumulative maximum shutdown time over the n consecutive power cycles in an environment condition e. Given $\mathcal{N} \gg n$ shutdown time profiles, $S^e(n)$ can be obtained by

$$S^e(n) = \max_{1\le k\le \mathcal{N}-n+1} \sum_{j=k}^{n+k-1} \hat{s}^e_j. \tag{1}$$

Fig. 8 top plot gives an example of $S^e(n)$ in various real-world environment conditions (the x-axis indicates n). Static and dynamic light are collected under a controllable artificial light source, whereas sunny and cloudy are collected under the natural sunlight under two different weather conditions. For each condition e in this figure, $S^e(n)$ determines a conservative estimate of the total time required to charge the IPD for the execution across the n power cycles. Note that, $S^e(n)$ is nonlinear, e.g., $S^e(n+1) \le S^e(n) + S^e(1)$.

Unlike the shutdown time, the live time of an inference task, $\hat{l}^e_{i,j}$, depends not only on the energy harvesting rate of the environment e but also on the energy consumption rate of the system. As shown in Section III-B with Fig. 5, the energy harvesting rate is the dominant factor in the live time per power cycle, while the variation due to the type of tasks or CMs is relatively small (less than 7% of the live time per power cycle). We therefore make the following assumption A2.

A2: While turned on, the energy consumption rate of the IPD is the same for all the tasks.

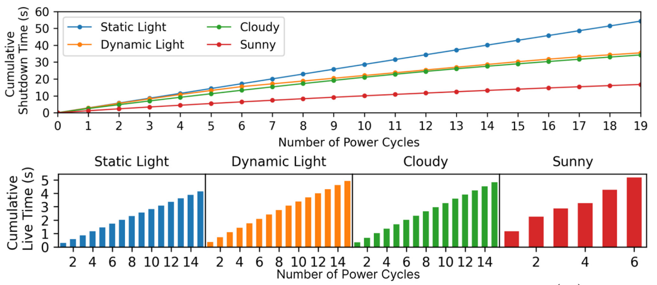


Fig. 8. Top: cumulative maximum shutdown time $S^e(n)$ under different environment condition e. Bottom: cumulative minimum live time $L^e(n)$ for CIFAR10-12 inference task τ_{C12}.

Strictly speaking, this assumption is not necessarily true because tasks may have different memory and I/O access patterns. However, since our work focuses on DNN inference tasks that do not involve direct I/O access and the energy harvesting rate has a much higher impact on the device's live time per power cycle, we find A2 works well in practice. With A2, we use $\hat{l}^e_j$ to denote the live time for the j-th power cycle and derive a live time function $L^e(n)$, similar to $S^e(n)$.

Definition 2 (Cumulative Live Time): $L^e(n)$ gives the cumulative minimum live time over the n consecutive power cycles in an environment condition e. Given $\mathcal{N} \gg n$ live time profiles, $L^e(n)$ can be obtained by

$$L^e(n) = \min_{1 \le k \le \mathcal{N}-n+1} \sum_{j=k}^{n+k-1} \hat{l}^e_j. \tag{2}$$

Unlike $S^e(n)$, $L^e(n)$ captures the *minimum* cumulative time. This allows us to have a conservative estimate of the time available for the task execution over the n power cycles. We can use $L^e(n)$ to find out how many power cycles are needed to execute a job of a task, assuming no other tasks are executing in the system. For instance, if a job has the execution time of t units, finding n that satisfies $L^e(n-1) < t \le L^e(n)$ tells us the number of power cycles involved. Hence, we can derive an inverse function, $\overline{L^e}(t)$, which gives the number of power cycles for the t units of execution.

Fig. 8 bottom illustrates $L^e(n)$ during one job execution of the CIFAR10-12 inference task under the same four real-world environments. Obviously, it takes the least number of power cycles in the sunny condition. Both $S^e(n)$ and $L^e(n)$ are stored in the device's NVM so that the scheduler can access them for the online adaptation.

The execution time and memory usage of each layer of a task τ_i are affected by the CM choice and whether the layer experiences shutdown during the execution (Section III-B). Therefore, for each layer k of τ_i, we record two lists of execution times with JIT, ST-L, ST-F, and ST-T: $(a^{\text{JIT}}_{i,k}, a^{\text{ST-L}}_{i,k}, a^{\text{ST-F}}_{i,k}, a^{\text{ST-T}}_{i,k})$ represent times without shutdown ("alive") and $(d^{\text{JIT}}_{i,k}, d^{\text{ST-L}}_{i,k}, d^{\text{ST-F}}_{i,k}, d^{\text{ST-T}}_{i,k})$ denote times with shutdown ("dead"). We also record the maximum memory usage of the kth layer of τ_i under four CMs: 1) ($v^{\text{JIT}}_{i,k}$; 2) $v^{\text{ST-L}}_{i,k}$; 3) $v^{\text{ST-F}}_{i,k}$; and 4) $v^{\text{ST-T}}_{i,k}$). For ease of reference, we introduce the functions $a_i(x,k)$ and $d_i(x,k)$ to obtain the execution times of the τ_i's layer k under a given CM x when the layer is alive or dead respectively. We also introduce a function $v_i(x,k)$ for the memory usage.

Algorithm 1: Minimize Task Execution Time

1 $\varepsilon_{\tau_i}[0][1 \cdots V_{ipd}] \leftarrow 0$; $cm_{\tau_i}[0][1...V_{ipd}] \leftarrow \emptyset$;
2 **for** $V \in \{1 \cdots V_{ipd}\}$ **do**
3   **for** $k \in \{1 \cdots \eta_i\}$ **do**
4     $min_\varepsilon \leftarrow \infty$; $min_{cm} \leftarrow \emptyset$;
5   **for** $x \in$ {*JIT, ST-L, ST-F, ST-T*} **do**
6   **if** $v_i(x,k) > V$ **then**
7   **continue**; /* Ignore CM violating mem limit */;
8   **end**
9       $val \leftarrow \varepsilon_{\tau_i}[k-1][V] + a_i(x,k)$ /* No shutdown */;
10   **if** $\overline{L^e}(val) \ne \overline{L^e}(\varepsilon_{\tau_i}[k-1][V])$ **then**
11         $val \leftarrow \varepsilon_{\tau_i}[k-1][V] + d_i(x,k)$; /* Shutdown */;
12   **end**
13   **if** $min_\varepsilon > val$ **then**
14         $min_\varepsilon \leftarrow val$; $min_{cm} \leftarrow x$;
15   **end**
16   **end**
17     $\varepsilon_{\tau_i}[k][V] \leftarrow min_\varepsilon$;
18     $cm_{\tau_i}[k][V] \leftarrow cm_{\tau_i}[k-1][V] \cup min_{cm}$;
19   **end**
20 **end**

B. Offline Intermittence-Aware CM Search

Our goal is to determine layer-wise CMs to minimize the solo execution time of a task τ_i across the power cycles, i.e., when τ_i runs without the temporal interference from the other tasks, while ensuring that the collective peak memory usage of all the tasks stays within the IPD's memory constraint, V_{ipd}. We solve this problem through a two-level dynamic programming approach.

First, we minimize each task τ_i's execution time under a memory constraint V. Let us define $\varepsilon_{\tau_i}[k][V]$ as follows:

$$\begin{aligned} \varepsilon_{\tau_i}[k][V] &= \tau_i\text{'s collective execution time from} \\ &\quad\ \text{layers 1 to } k\text{, while not exceeding the} \\ &\quad\ \text{memory constraint } V. \\ cm_{\tau_i}[k][V] &= \text{CMs achieving } \varepsilon_{\tau_i}[k][V]. \end{aligned} \tag{3}$$

As each layer uses nonzero memory, $\varepsilon_{\tau_i}[k][0] = \infty$ and $cm_{\tau_i}[k][0] = \emptyset$ for all $k \le \eta_i$. The execution time of the layers 1 to k can be found by considering the occurrence of shutdowns. The peak memory usage of a task τ_i is determined by the layer that uses the most memory among all the layers. Hence, we can compute $\varepsilon_{\tau_i}[k][V]$ and $cm_{\tau_i}[k][V]$ using Algorithm 1.

Algorithm 1 iterates over the memory size V from 1 to V_{ipd}, and for each V, iterates over the layers from 1 to η_i. For each layer k, it considers four CMs (line 5). Recall that the peak memory usage of τ_i is determined by the layer with the maximum usage, not by the summation of all the layers. Hence, if the use of a CM x for the kth layer violates the memory constraint V, that CM x should be ignored (line 6). If the kth layer has no shutdown, the cumulative execution time up to the layer k is calculated by summing $\varepsilon_{\tau_i}[k-1][V]$ and $a_i(x,k)$ (line 9). If a shutdown occurs during the layer k, the total number of power cycles up to the layer $k-1$ will be different from the number up to the layer k (obtainable using the pseudo-inverse function $\overline{L^e}(t)$), and $d_i(x,k)$ needs

Algorithm 2: MII Online Scheduler

1 **Input:** $S^{e'}(n)$, $L^{e'}(n)$, $\hat{l}$ and t_{prev} in NVM;
2 $t_{start} \leftarrow$ RTC_now(); $\hat{s} \leftarrow t_{start} - t_{prev}$;
3 /* Power Cycle Harmonizing (PCH) */;
4 $Q_{tasks} \leftarrow$ Tasks arrived by t_{start} but not finished their jobs;
5 **for** $\forall \tau_i \in Q_{tasks}$ **do**
6   Assign_Priority(τ_i); /* LST */;
7 **end**
8 /* CMs Adaptation */;
9 **if** $S^{e'}(1) < \hat{s} \vee L^{e'}(1) > \hat{l}$ **then**
10   Update $S^{e'}(n)$ and $L^{e'}(n)$; /* Stored in NVM */;
11   **for** $\forall \tau_i \in Q_{tasks}$ **do**
12         $cm_{\tau_i} \leftarrow$ Online_Adaptation(τ_i);
13   **end**
14 **end**
15 /* Task Scheduling*/;
16 **while** $Q_{tasks} \neq \emptyset$ **do**
17     $\tau_i \leftarrow$ Pick_Highest_Priority(Q_{tasks});
18   **if** $cm_{\tau_i} \neq$ JIT **then**
19   Check_Proactive_Shutdown();
20   **end**
21   Run(τ_i);
22   **if** τ_i *completed its job* **then**
23         $Q_{\text{tasks}} \leftarrow Q_{\text{tasks}} \setminus \tau_i$;
24   **end**
25 **end**
26 /* $Q_{tasks} = \emptyset$ or JIT threshold or Proactive Shutdown triggered */;
27 Save the states of JIT tasks to NVM;
28 $t_{prev} \leftarrow$ RTC_now(); /* Store in NVM */;
29 $\hat{l} \leftarrow t_{\text{prev}} - t_{\text{start}}$; IPD Shutdown;

to be used instead (line 11). Once the algorithm finishes, $\varepsilon_{\tau_i}[\eta_i][V_{ipd}]$ gives the minimum execution time of τ_i under the memory constraint.

The next step is to minimize the collective sum of solo execution times of all the m tasks under the device's memory constraint. The memory usage of the system Γ is determined by adding up the peak memory usage of each task $\tau_i \in \Gamma$. Let us define $E_\Gamma[i][V]$ as the minimum sum of the solo execution times of the i tasks (from τ_1 to τ_i) in Γ with the memory constraint of V, and $CM_\Gamma[i][V]$ as the corresponding CM information. This can be solved by dynamic programming with the following recurrence relation:

$$E_\Gamma[i][V] = \min_{1 \leq j \leq V-1} E_\Gamma[i-1][j] + \varepsilon_{\tau_i}[\eta_i][V-j] \tag{4}$$

and with j found for $E_\Gamma[i][V]$

$$CM_\Gamma[i][V] = CM_\Gamma[i-1][j] \cup \{cm_{\tau_i}[\eta_i][V-j]\}. \tag{5}$$

The initial conditions are

$$E_\Gamma[0][1 \cdots V_{ipd}] = 0, E_\Gamma[1][1 \cdots V_{ipd}] = \varepsilon_{\tau_1}[\eta_1][1 \cdots V_{ipd}], \text{ and}$$

$$CM_\Gamma[0][1 \cdots V_{ipd}] = \emptyset, CM_\Gamma[1][1 \cdots V_{ipd}] = cm_{\tau_1}[\eta_1][1 \cdots V_{ipd}].$$

For all the m tasks in Γ, the solution is given by $E_\Gamma[m][V_{ipd}]$ and $CM_\Gamma[m][V_{ipd}]$. We use $CM_\Gamma[m][V_{ipd}]$ as the initial CM settings of the tasks when the system is deployed. Also, we store each task's peak memory usage corresponding to the solution as M_{τ_i} in the device's NVM since it will be used as guidance by the online adaptation.

C. MII Online Scheduler

The main goal of our scheduler is to make task scheduling decisions based on the environment condition at runtime e'.

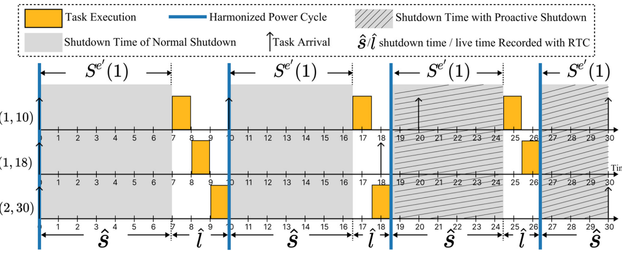


Fig. 9. MII online scheduler with three tasks. Each task τ_i is characterized by (execution time and period). PCH delays the execution of τ_2's second job. Proactive shutdown is shown by hatched gray boxes.

The pseudocode of the scheduler is given in Algorithm 2, which begins upon each reboot. Upon start, the scheduler uses the onboard RTC to compute the *online shutdown time* of the current power cycle, denoted as $\hat{s}$, by taking the difference between the timestamp recorded at the previous shutdown, t_{prev}, and the current timestamp at the start, t_{start} (line 2). It also estimates the *online live time* $\hat{l}$, calculated at the end of the final power cycle (line 29). These $\hat{s}$ and $\hat{l}$ are used to determine the condition to trigger the online CM adaptation, presented in the next subsection. The other components of the scheduler are explained below.

Power Cycle Harmonizing (PCH): An arbitrary arrival of periodic tasks is one of the reasons causing runtime deviations from the power cycle used by our offline search. To address this issue, our scheduler introduces PCH, which harmonizes the task execution with the power cycle. PCH forces the scheduler to take into account only the tasks that have either arrived by t_{start} or those that have not finished their job execution within their periods (line 4, Q_{tasks}). Therefore, the execution of any task that arrives during the live time of the current power cycle is deferred to the next power cycle, allowing the CM choices of the tasks not to be disrupted by newly arriving tasks. Fig. 9 gives an example of the scheduling behavior with PCH. For the nth power cycle, the start of execution for all the three tasks is harmonized to the end of $\hat{s}_n$, ensuring that the $n+1$ power cycle begins when the IPD turns off.

Task Scheduling: For the tasks found by PCH, Q_{tasks}, our scheduler uses a variant of the least slack time (LST) scheduling policy to dynamically change the task priorities, with each task's period as its deadline (lines 5 and 6). Our LST variant checks slack time at the boundary of each layer execution to mitigate the overhead of the standard LST. The reason behind the use of LST is the following. Since, DNN inference jobs are relatively long, their response times can be easily greater than their periods and the unscheduled jobs are skipped from the execution. If we use the other policies like EDF which is a job-level fixed-priority policy, a long-running job may dominate the live time over the multiple power cycles as its priority does not change until completion, leading to starvation to the other jobs and resulting in a disproportionate number of successful jobs per task compared to their periods. In other words, the use of LST can help achieve fairness in skipped jobs across tasks, as we will show in our evaluation. After updating CMs with the online adaption (lines 9–12), the scheduler proceeds to execute the tasks in Q_{tasks} in the priority order and remove it from Q_{tasks} upon successful job completion (lines 16–25).

Proactive Shutdown: With ST, an IPD may shut down during the execution of an atomic block and re-execute it in the next power cycle. The re-executed portion is called *wasted work* [1], [6], [12], [13]. Although existing work mitigates this by voluntarily shutting down the IPD after a fixed number of calculations [3], it no longer works in a multitask system with a mixture of JIT and ST because the JIT threshold can be triggered at any time of atomic block execution. To address this problem, we propose a proactive shutdown method. Proactive shutdown can be triggered by the scheduler before running any task τ_i that uses ST as its CM (line 19). The scheduler proceeds with τ_i's execution only if the stored energy is enough to execute at least a single block of τ_i. Otherwise, it makes the device shut down (line 26). This can also be triggered when no other task to execute ($Q_{\text{tasks}} = \emptyset$) or the JIT threshold is met. The overhead of the proactive shutdown is negligible since it only requires an ADC reading at the end of each ST's block. Furthermore, for any JIT-enabled IPDs, ADC reading is already a prerequisite [7], [10], [11], [26].

One might have a concern that the proactive shutdown turns the IPD off earlier and shortens the shutdown time $\hat{s}$ (Fig. 9 hatched gray boxes), which might affect the condition to trigger the online adaption. However, since the online CM adaption is triggered only when $\hat{s}$ reaches a larger value than expected (explained in the next subsection), unnecessary online adaptations are effectively prevented.

D. Online CM Adaption

To enable online CM adaptions, MII maintains $S^{e'}(n)$ in NVM, which is an online version of the cumulative shutdown time $S^{e}(n)$ and initialized as $S^{e'}(n) = S^{e}(n)$. Also, $L^{e'}(n)$, an online version of the live time $L^{e}(n)$, is also maintained in NVM and initialized as $L^{e'}(n) = L^{e}(n)$. Both $S^{e'}(n)$ and $L^{e'}(n)$ are given as input to the scheduler since they are essential to quantify the deviation between the online environment condition e' and the profiled environment condition e.

Recall Definitions (1) and (2). If the IPD follows $S^{e'}(n)$ and $L^{e'}(n)$, both $\hat{s} \le S^{e'}(1)$ and $\hat{l} \ge L^{e'}(1)$ hold for one power cycle ($n = 1$), indicating no shift of environment. Otherwise, either $\hat{s} > S^{e'}(n = 1)$ or $\hat{l} < L^{e'}(1)$ (line 9), meaning a potential shift of environment; hence, the scheduler first updates $S^{e'}(n)$ and $L^{e'}(n)$ with $\hat{s}$ and $\hat{l}$ (line 10), and then triggers the online adaptation (line 12). Since the direct use of the offline search algorithm (Section V-B) for the online adaption introduces a huge overhead, we take a heuristic approach presented below to find a near-optimal solution by using the offline solution's pertask memory usage, M_{τ_i}, as a constraint for τ_i.

The online adaptation is done individually for each task $\tau_i \in Q_{\text{tasks}}$ to update task's CM list $cm_{\tau_i}[0 \cdots \eta_i]$. It initializes $cm_{\tau_i}[1 \cdots \eta_i]$ as follows: for each layer k of τ_i, $cm_{\tau_i}[k] = \arg\min_x a_i(x, k)$, where $x \in \{\text{JIT, ST-L, ST-F, ST-T}\} \wedge v_i(x, k) \le M_{\tau_i}$. This discards any CM that causes the memory usage to exceed M_{τ_i}, and chooses the CM giving the minimum execution time without considering the shutdown during the execution. It also uses a vector variable $\varepsilon_{\tau_i}[1 \cdots \eta_i]$ to keep track of each layer's execution time corresponding to $cm_{\tau_i}[1 \cdots \eta_i]$. Then, it takes the following steps to take into account the effect of shutdown for each τ_i.

1) Start with the first power cycle $n = 1$.
2) Find a layer K that experiences a shutdown in the power cycle p. Such a layer K satisfies: $\sum_{k=1}^{K-1} \varepsilon_{\tau_i}[k] \le L^{e'}(n)$ and $\sum_{k=1}^{K} \varepsilon_{\tau_i}[k] > L^{e'}(n)$.
3) If K is found, this means the layer K is a shutdown layer. Hence, update $\varepsilon_{\tau_i}[K] = \min_x d_i(x, K)$ and $cm_{\tau_i} = \arg\min_x d_i(x, K)$.
4) Repeat steps 2 and 3 until K reaches η_i.

This method takes a linear search approach, much faster than the offline algorithm. However, its optimality may be compromised due to its reliance on the M_{τ_i} determined offline. Nonetheless, by incorporating $L^{e'}(n)$ which is continuously updated for the current environment e', this approach offers substantial benefits as we will demonstrate in the evaluation.

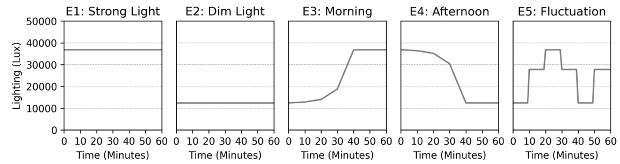


Fig. 10. Five controlled energy patterns represented as the lighting in a unit of illumination (lux). E1 and E2 are static lighting, whereas E3, E4, and E5 are dynamic lighting.

VI. MII Implementation

Energy Source: For the continuous power source, we used an X-NUCLEO-LPM01A. For the intermittent energy source, we harvested energy using an LTC3588 energy harvester and solar cells of 1.5 W peak power. We regulated the input voltage of IPD to 1.8 V and stored the harvested energy in a set of capacitors with the size of 1 mF. During the operations, the capacitor's voltage range is 2.87 to 4.03 V.

Evaluation Hardware: We chose the Ambiq Apollo4 Blue Plus evaluation board that has an ARM Cortex-M4 MCU, 2 MB MRAM as NVM, an on-board RTC unit for timekeeping,[3] and an on-board ADC for the capacitor voltage monitoring.

DNN Setup: We selected eight DNN models and categorized them into three test cases (TCs) based on their dominant layers. Details of these DNNs are found in Table I.

Our current implementation did not use the hardware acceleration for the DNN execution. However, since MII performs all the computations in the VM for both the JIT and ST, it can be safely adapted to the hardware acceleration features that usually require direct access to the data in the VM, e.g., TI's low energy accelerator (LEA). We leave such extensions as part of the future work.

VII. Controlled Environment Evaluation

A. Evaluation Setup

Controlled Energy Patterns: We conducted a three months study of the lighting condition changes inside a greenhouse environment and identified six distinct energy patterns (E0–E5) that can capture more than 97% of the lighting

[3]The RTC circuit ran with its own dedicated capacitor and it did not deplete during the experiment. For more reliable timekeeping, techniques like "persistent clocks" [27] can be considered.

TABLE I
DNNs Configuration

Test Case	DNN Name	Dataset	Layers Configuration	Params	Inputs	Top1 Acc.	Largest Layer
TC1	C7	CIFAR10	5xCONV2D+GP+FC	5,152	32x32x3	63%	63.9KB
TC1	M7	MNIST	5xCONV2D+GP+FC	5,136	28x28x2	98%	49.0KB
TC1	C12	CIFAR10	10xCONV2D+GP+FC	15,722	32x32x3	78%	63.6KB
TC1	H5	HAR	3xCONV1D+GP+FC	920	128x9	61%	3.5KB
TC2	FC4	KWS	4xFC	263,436	49x10x1	87%	503.8KB
TC2	AutoEncoder	ToyADMOS	10xFC	269,992	1x640	85%	328.7KB
TC3	MBV1	VWW	14xCONV2D+13xDCONV2D+AP+FC	221,794	96x96x3	80%	312.5KB
TC3	DSCNN	KWS	5xCONV2D+4xDCONV2D+AP+FC	24,908	49x10x1	90%	80.6KB

* CONV2D/1D:Convolution 2D/1D, GP: Global Pooling, FC: Fully Connected, DCONV2D: Depthwise convolution 2D, AP: Average Pooling

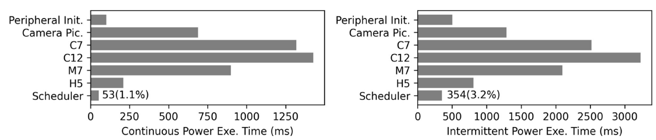


Fig. 11. System runtime breakdown when running four DNNs of TC1 under the continuous (the left) and intermittent (the right) power.

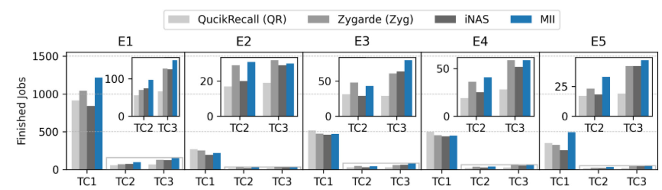


Fig. 12. Finished jobs by running each TC separately in E1–E5. MII only uses offline phase to search CMs under E1.

conditions during daytime. Apart from E0, the continuous power, E1 to E5 are all the intermittent power, and their energy patterns in one hour are shown in Fig. 10.

Baseline Configuration: We compare MII against the three established and state-of-the-art methods: QuickRecall [10] which is a JIT checkpointing-only system, iNAS [3] which uses ST-T with a tile size determined offline, and Zygarde [2] which uses ST-F and an early-exit model comprising mandatory and optional layers. For a fair comparison, we adjusted several settings of each method: in QuickRecall, we determined the JIT threshold to ensure the successful checkpointing of all the JIT tasks in the system. In Zygarde, we set the first layer of each model as a mandatory layer since at least one layer OFM is needed for the early exit. Also, we marked an inference result from a mandatory job as successful if it matched the result of a complete job. It is worth noting that the evaluation of Zygarde subsumes SONIC [1], which also focuses on the intermittent DNN inference, because Zygarde extends SONIC's APIs and has shown to outperform SONIC. In iNAS, we derived appropriate tile parameters by balancing the peak memory usage and execution time. Although a larger tile size can shorten execution time by improving the data reuse, it increases peak memory size, leading to out-of-memory if multiple tasks are executed concurrently.

Test Cases and Job Generation: The evaluation encompasses eight DNNs in three TCs given in Table I. To evaluate the performance of individual and combined TCs that represent different task sizes, we consider four scenarios: (A) *All TC1:* all the four DNNs in TC1; (B) *TC1+TC2:* two DNNs (C7 and M7) selected from TC1 and one model (FC4) from TC2; (C) *TC2+TC3:* one model (autoencoder) from TC2 and one model (DSCNN) from TC3; and (D) *TC1+TC3:* two DNNs (C7 and C12) from TC1 and one model (MBV1) from TC3. Each model is executed by a distinct task on FreeRTOS. We schedule the tasks with specific periods, as will be shown by the number of generated jobs in the later sections. We scale task periods w.r.t. the energy pattern because less lighting causes the sensor to sample fewer readings.

System Overhead: To evaluate the overhead of MII's runtime scheduler and other noninference tasks, we profiled the execution time of each software component when running all the TC1 models on one downsampled image. The IPD first took a picture from the camera, stored a downsampled version in NVM, and then processed the image by running the inference of each DNN sequentially. Fig. 11 shows the runtime breakdown of the described workload under both continuous power provided and an intermittent power source following E1 energy pattern. The overhead of the MII scheduler is composed of only 1.1% under the continuous power and 3.2% under the intermittent power of the entire system runtime.

B. Offline Effectiveness

CM Search Algorithm: To evaluate the effectiveness of our offline search across various energy patterns, we use the offline searched optimal CMs found from the energy pattern E1 and apply these CMs to E2–E5. During a one-hour period, we measure for each inference task how many jobs execute successfully (denoted as *finished jobs*). For a level comparison, Zygarde's finished jobs are captured by executing the complete DNN inference without relying on the early-exit feature. This feature will be assessed in the online evaluation in Section VII-C. The results in Fig. 12 showcase that, for E1, MII outperforms QuickRecall, Zygarde, and iNAS by completing 41%, 18%, and 40% more jobs, respectively. This is somewhat expected since the CMs are searched using E1. Although QuickRecall finished 7% more jobs on average than MII for smaller DNNs from TC1 in E2–E4, it suffers from the out-of-memory when running large DNNs and finishes the least TC2-3 inference jobs across all the baselines. Interestingly, even when the IPD runs inference tasks using the E1-optimized CMs for E2–E5, MII still achieves 10% and 28% more jobs than Zygarde and iNAS, respectively. These results show that the CMs searched by MII in a single energy pattern demonstrate efficacy across all the examined energy

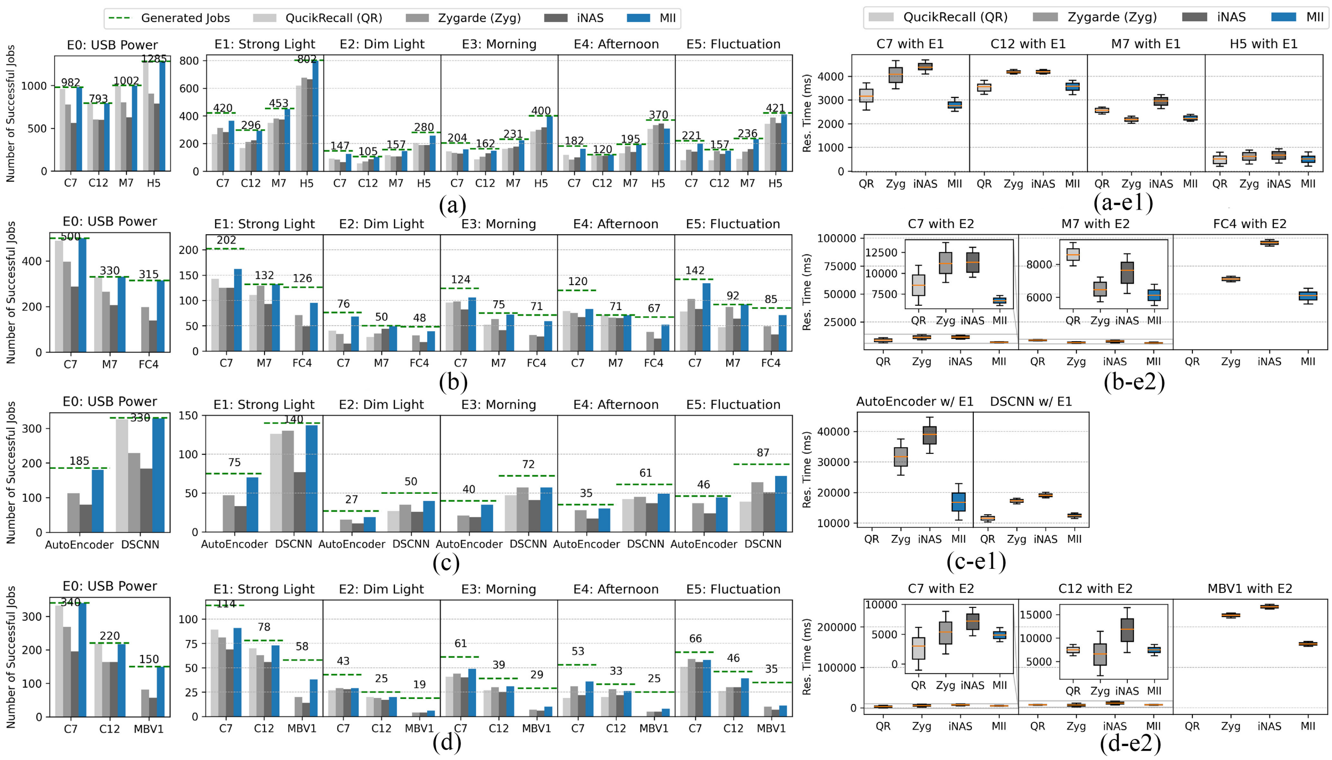


Fig. 13. Left: successful jobs of four TCs combinations with six energy patterns. Task periods are scaled with the energy pattern and are indicated by the numbers above green dashed lines. Right: response time of TCs under one stable energy pattern. (a) TC1 all DNNs (C7, C12, M7, and H5) successful jobs in the continuous (E0) and intermittent power (E1–E5). (b) TC1 (C7 and M7) and TC2 (FC4) successful jobs in the continuous (E0) and intermittent power (E1–E5). (c) TC2 (AutoEncoder) and TC3 (DSCNN) successful jobs in the continuous (E0) and intermittent power (E1–E5). (d) TC1 (C7 and C12) and TC3 (MBV1) successful jobs in the continuous (E0) and intermittent power (E1–E5).

patterns compared to the other methods. This underscores the robust generality of our search algorithm.

Peak Memory: Fig. 14 shows the peak memory usage when applying the CMs obtained from the MII's offline methods under the E1 energy pattern. MII can effectively reduce the peak memory compared to QuickRecall and Zygarde. QuickRecall consistently accesses the layer's IFM, OFM, and weights, resulting in the highest peak memory. Although Zygarde often needs to load the entire layer's OFM for an early exit, its memory usage is still smaller than QuickRecall. iNAS has the smallest because it only loads partial IFM, weights, and OFM in each power cycle; however, it suffers from increased execution time. MII's reduction in peak memory, compared to QuickRecall and Zygarde, is due to that MII can choose a non-JIT CM or finer-grained ST for a large layer by imposing the memory constraint.

C. Online Scheduling and Adaption Effectiveness

Successful Jobs: The left part of Fig. 13 illustrates the number of successful jobs in four combinations of TC1–TC4, with the total number of generated jobs indicated by green dashed lines. If an optimal solution exists for each online energy pattern (E1–E5), its number of successful jobs is upper-bounded by the green dashed lines. Hence, the gap in each subplot between the green line and each bar represents the deviation between each baseline and the optimal solution. Although it does not reach the optimal performance MII still

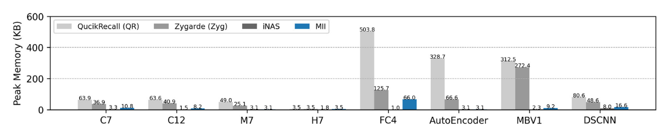


Fig. 14. Peak memory usage. MII uses E1 to search CMs.

achieves on average a 21% increase of successful jobs in stable energy conditions (E1 and E2) and on average 39% successful jobs increase under dynamic energy conditions (E3–E5) compared to the other methods. When comparing these results with the offline evaluation, MII's online phase significantly improves the job success rates over the other three methods. Specifically, for (A) TC1 all in E1, there is a 56% increase in successful jobs, thanks to the online scheduler and adaptation of MII.

In addition to the MII's significant improvement in total successful jobs, the following two observations can be made.

1) The LST variant of MII mitigates starvation of short-period jobs and helps achieve fairness across tasks. For example, in Fig. 13(b) E5, when using MII, the short-period task C7 can complete up to 94% of its generated jobs, and the long-period tasks M7 and FC4 complete 99% and 84% of their generated jobs. On the other hand, when using Zygarde, which uses an EDF variant, one of the long-period tasks M7 completes 95% while the other two tasks complete only 73% and 58% of their jobs, showing a much higher discrepancy in successful jobs per task than MII.

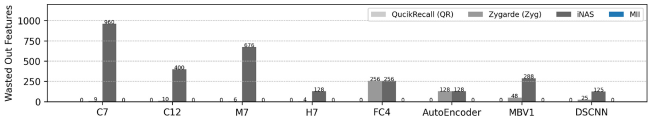


Fig. 15. Observed wasted work under execution with E1.

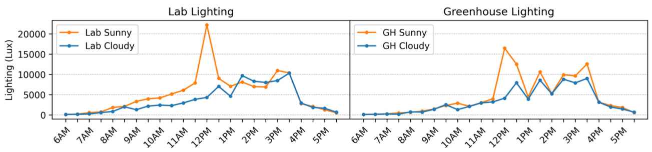


Fig. 16. Real-world solar energy pattern.

2) If we ignore the correctness of inference results of jobs, Zygarde has executed more jobs than MII in most cases due to its early-exit feature, which executes only a mandatory portion of the job. However, if a task uses a large DNN model, this feature makes Zygarde suffer from the low accuracy, resulting in a lower number of successful jobs. Therefore, in Fig. 13(d)-E5, although Zygarde achieves 2% more successful jobs than MII for a small DNN task C7, it achieves 30% and 10% less successful jobs than MII for the large DNNs tasks C12 and MBV1, respectively.

Response Time: The right side of Fig. 13 showcases the response time of DNN inference tasks under one stable energy pattern (E1 or E2). Although the response time is not the major optimization objective of MII (successful job is), MII tends to yield more preferable results overall, with a shorter average response time and a smaller variation. QuickRecall suffers from the out-of-memory issue when executing a large model, such as AutoEncoder and MBV1. Hence, although it gives a shorter response time for small models (A-E1 in the figure), it completes 35%, 40%, and 22% less successful jobs on average compared to MII for TC1+TC2, TC2+TC3, and TC1+TC3, respectively. Zygarde often gives a shorter response time due to its early-exit feature, but it comes at the cost of low accuracy.

Wasted Work: We characterize the amount of wasted work by profiling the number of output features discarded during the shutdown. Fig. 15 depicts the wasted work of each DNN when running the four combinations of TCs under the E1 energy pattern. Both Zygarde and iNAS show some wasted work due to the issue discussed in Section V-C. For QuickRecall, since it uses JIT, it obviously has zero wasted work. However, to ensure that JIT successfully checkpoints all the tasks' largest layers, a higher JIT threshold is required for QuickRecall compared to MII, i.e., 3.7 V versus 3.3 V. MII has zero wasted work while keeping a smaller JIT threshold because of our proactive shutdown technique (Section V-C) as well as the ability to avoid using JIT for the large layers.

VIII. Real World Evaluation

Experiment Setting: The IPD was deployed in two settings: 1) lab window side (Lab) and 2) greenhouse (GH). We evaluated the system under the sunny and cloudy conditions, both occurring within a single day. The lighting changes for each setting are depicted in Fig. 16. In Lab, the IPD was positioned under the direct sunlight with minimal interference. However, in GH, IPD faced consistent interference from the leaf shades and plant shadows. To demonstrate effectiveness in common smart sensing applications, all the TC1 DNNs listed in Table I were selected for the experiment and ran continuously from 6 AM to 6 PM on a day (Fig. 16). A comparison setup, running the same set of DNNs with Zygarde [2], was placed adjacent to the experimental setup.

TABLE II
Successful Jobs of MII Compared to Zygarde

DNN	Lab Sunny	Lab Cloudy	GH Sunny	GH Cloudy
C7	+17%	+51%	+20%	+61%
C12	+38%	+47%	+38%	+9%
M7	+18%	+34%	+33%	+6%
H5	+19%	+35%	+33%	−7%

Successful Jobs: Table II presents the percentage difference in the number of successful jobs executed by MII compared to Zygarde. In the GH Clody setting, MII outperformed Zygarde for all the three DNNs, executing 61%, 9%, and 6% more jobs than Zygarde. However, for H5 in the same setting, MII completed 7% fewer jobs. It was mainly due to that, although Zygarde executed only mandatory layers under the cloudy condition, it still managed to produce around 95% of the inference results that matched the results of H5's complete inference. However, this favorable outcome for Zygarde is largely confined to the small models with inherently low accuracy, which are not substantially affected by the accuracy degradation of early exits. Overall, MII outperformed Zygarde by an average of 33% more successful jobs in the Lab and 24% in the GH.

IX. Related Work

A. Intermittent Computing

In the context of the IPD software systems, the prior work has focused on issues, such as memory inconsistency [5], [6], [8], [10], [23], task idempotency [7], [8], [11], [13], [20], and sensing/computation task coordination [9], [12], [14], [25]. There are numerous studies on the IPD hardware improvement [26], [28] and energy harvesting circuits [21], [29].

Emergent research about running machine learning workloads on the IPDs is still in its very early stage. Existing work focuses on learning [19] as well as the inference tasks [1], [2], [3], [4], [13], [20], [25] on the IPDs. However, none of these has studied the tradeoff between the JIT [7], [10], [11] and ST [1], [3], [8], [9], [12], [13], [14] CMs as well as the different granularity of the atomic blocks in ST for the DNN inference tasks. Although there have been some attempts to co-use the JIT and ST in the same system [12], [25], they use ST exclusively for the peripheral access and JIT for the computational tasks, meaning that when applied to the inference tasks, all will be governed by JIT.

Recent work [26] has proposed to put the device into the sleep mode and trigger JIT checkpointing only when no more energy is available. While this has the potential to improve the standard JIT method used in MII, depending on the leakage current in the sleep mode, it can increase the recharge time of the capacitor. For example, in our evaluation platform, using its default deep sleep mode results in at least 58% more time to fully recharge under the E2 energy pattern.

B. Real-Time Scheduling on IPD

Supporting task scheduling on the IPDs has been considered a big challenge because the environment fluctuations lead to varying task response times [9], [14]. Existing work has approached this issue in two ways: 1) energy prediction [15], [16], [17], which analyses the supply energy pattern and makes scheduling decisions accordingly and 2) workload reduction [2], [14], [23], [24], [25], which reactively changes the amount of workload with respect to the environment variations. Specifically, Celebi [15] leverages the energy prediction approach and presents both the offline and online schedulers to meet the task deadlines on the IPDs. Zygarde [2] trains a DNN to be early exit-able at every layer and splits the layers into mandatory and optional parts. It then schedules either only mandatory layers or both the types of layers based on the ambient conditions. However, none of the existing work has studied the effect of the execution patterns given different CMs. MII addresses these limitations.

X. Conclusion

This articles presents MII: a multifaceted framework for the intermittent inference and scheduling. The design of MII originated from the three key observations in Section III and divided into the offline and online phases. MII is compared with the three representative state-of-the-art methods [2], [3], [10] through a controlled environment evaluation as well as a real-world field study. The results show that MII is able to achieve the performance efficiency in the intermittent DNN inference, adaptability to environment changes, and applicability to the real-world scenarios. Future work includes the consideration of different learning tasks, such as deep reinforcement learning on the IPDs.

References

[1] G. Gobieski, B. Lucia, and N. Beckmann, “Intelligence beyond the edge: Inference on intermittent embedded systems,” in *Proc. ASPLOS*, 2019, pp. 199–213.

[2] B. Islam and S. Nirjon, “Zygarde: Time-sensitive on-device deep inference and adaptation on intermittently-powered systems,”‘*Proc. ACM Interact. Mob. Wearable Ubiquitous Technol.*, vol. 4, no. 3, pp. 1–29, Sep. 2020.

[3] H. R. Mendis, C. K. Kang, and P. Hsiu, “Intermittent-aware neural architecture search,” *ACM Trans. Embed. Comput. Syst.*, vol. 20, no. 5s, pp. 1–27, Sep. 2021.

[4] C.-K. Kang, H. R. Mendis, C.-H. Lin, M.-S. Chen, and P.-C. Hsiu, “Everything leaves footprints: Hardware accelerated intermittent deep inference,” *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 39, no. 11, pp. 3479–3491, Nov. 2020.

[5] M. Buettner et al., “Dewdrop: An energy-aware runtime for computational RFID,” in *Proc. NSDI*, 2011, pp. 1–14.

[6] B. Lucia and B. Ransford, “A simpler, safer programming and execution model for intermittent systems,” in *Proc. PLDI*, 2015, pp. 1–11.

[7] D. Balsamo, A. S. Weddell, G. V. Merrett, B. M. Al-Hashimi, D. Brunelli, and L. Benini, “Hibernus: Sustaining computation during intermittent supply for energy-harvesting systems,” *IEEE Embed. Syst. Lett.*, vol. 7, no. 1, pp. 15–18, Mar. 2015.

[8] B. Ransford, J. Sorber, and K. Fu, “Mementos: System support for long-running computation on RFID-scale devices,” in *Proc. ASPLOS*, 2011, pp. 1–12.

[9] J. Hester, K. Storer, and J. Sorber, “Timely execution on intermittently powered batteryless sensors,” in *Proc. 15th ACM Conf. Embed. Netw. Sens. Syst.*, 2017, pp. 1–13.

[10] H. Jayakumar, A. Raha, W. S. Lee, and V. Raghunathan, “QuickRecall: A HW/SW approach for computing across power cycles in transiently powered computers,” *J. Emerg. Technol. Comput. Syst.*, vol. 12, no. 1, pp. 1–19, Aug. 2015.

[11] D. Balsamo et al., “Hibernus++: A self-calibrating and adaptive system for transiently-powered embedded devices,” *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 35, no. 12, pp. 1968–1980, Mar. 2016.

[12] K. Maeng and B. Lucia, “Supporting peripherals in intermittent systems with just-in-time checkpoints,” in *Proc. PLDI*, 2019, pp. 1–16.

[13] K. Maeng, A. Colin, and B. Lucia, “Alpaca: Intermittent execution without checkpoints,” *Proc. ACM Program. Lang.*, vol. 1, pp. 1–30, Oct. 2017.

[14] K. Maeng and B. Lucia, “Adaptive dynamic checkpointing for safe efficient intermittent computing,” in *Proc. OSDI*, 2018, pp. 1–17.

[15] B. Islam and S. Nirjon, “Scheduling computational and energy harvesting tasks in deadline-aware intermittent systems,” in *Proc. RTAS*, 2020, pp. 95–109.

[16] M. Karimi et al., “Real-time task scheduling on intermittently powered batteryless devices,” *IEEE Internet Things J.*, vol. 8, no. 17, pp. 13328–13342, Sep. 2021.

[17] M. Karimi, Y. Wang, and H. Kim, “Energy-adaptive real-time sensing for Batteryless devices,” in *Proc. RTCSA*, 2022, pp. 205–211.

[18] A. Bukhari, S. Hosseinimotlagh, and H. Kim, “OpenSense: An open-world sensing framework for incremental learning and dynamic sensor scheduling on embedded edge devices,” *IEEE Internet Things J.*, vol. 11, no. 15, pp. 25880–25894, Aug. 2024.

[19] S. Lee, B. Islam, Y. Luo, and S. Nirjon, “Intermittent learning: On-device machine learning on intermittently powered system,” *Proc. ACM Interact. Mob. Wearable Ubiquitous Technol.*, vol. 3, no. 4, pp. 1–30, Sep. 2020.

[20] A. Colin and B. Lucia, “Chain: Tasks and channels for reliable intermittent programs,” *SIGPLAN Notice*, vol. 51, no. 10, pp. 514–530, Oct. 2016.

[21] A. Colin, E. Ruppel, and B. Lucia, “A reconfigurable energy storage architecture for energy-harvesting devices,” in *Proc. ASPLOS*, 2018, pp. 767–781.

[22] J. Choi, L. Kittinger, Q. Liu, and C. Jung, “Compiler-directed high-performance intermittent computation with power failure immunity,” in *Proc. RTAS*, 2022, pp. 40–54.

[23] K. S. Yıldırım, A. Y. Majid, D. Patoukas, K. Schaper, P. Pawelczak, and J. Hester, “InK: Reactive kernel for tiny batteryless sensors,” in *Proc. 16th ACM Conf. Embed. Netw. Sens. Syst.*, 2018, pp. 41–53.

[24] K. Maeng and B. Lucia, “Adaptive low-overhead scheduling for periodic and reactive intermittent execution,” in *Proc. PLDI*, 2020, pp. 1005–1021.

[25] M. Surbatovich, N. Spargo, L. Jia, and B. Lucia, “A type system for safe intermittent computing,” *Proc. ACM Program. Lang.*, vol. 7, Jun. 2023, pp. 736–760.

[26] K. Akhunov, E. Yildiz, and K. S. Yildirim, “Enabling efficient intermittent computing on brand new microcontrollers via tracking programmable voltage thresholds,” in *Proc. 11th Int. Workshop Energy Harvest. Energy-Neutral Sens. Syst.*, 2023, pp. 16–22.

[27] J. Hester et al., “Persistent clocks for batteryless sensing devices,” *ACM Trans. Embed. Comput. Syst.*, vol. 15, no. 4, pp. 1–28, 2016.

[28] A. Colin, G. Harvey, B. Lucia, and A. P. Sample, “An energy-interference-free hardware-software debugger for intermittent energy-harvesting systems,” in *Proc. ASPLOS*, 2016, pp. 1–13.

[29] P. Nintanavongsa, U. Muncuk, D. R. Lewis, and K. R. Chowdhury, “Design optimization and implementation for RF energy harvesting circuits,” *IEEE J. Emerg. Select. Topics Circuits Syst.*, vol. 2, no. 1, pp. 24–33, Mar. 2012.