SCHEDULING TASKS ON INTERMITTENTLY-POWERED REAL-TIME SYSTEMS

Bashima Islam

A dissertation submitted to the faculty at the University of North Carolina at Chapel Hill in partial fulfillment of the requirements for the degree of Doctor of Philosophy in the Department of Computer Science in the College of Arts and Sciences.

Chapel Hill
2021

Approved by:

Shahriar Nirjon

Kevin Jeffay

Donald E. Porter

Xiaofan Jiang

Parasara Sridhar Duggirala

ABSTRACT

Bashima Islam: Scheduling Tasks on Intermittently-Powered Real-Time Systems
(Under the direction of Shahriar Nirjon)

Batteryless systems go through sporadic power on and off phases due to intermittently available energy; thus, they are called intermittent systems. Unfortunately, this intermittence in power supply hinders the timely execution of tasks and limits such devicesâ potential in certain application domains, e.g., healthcare, live-stock tracking. Unlike prior work on time-aware intermittent systems that focuses on timekeeping [1, 2, 3] and discarding expired data [4], this dissertation concentrates on finishing task execution on time. I leverage the data processing and control layer of batteryless systems by developing frameworks that (1) integrate energy harvesting and real-time systems, (2) rethink machine learning algorithms for an energy-aware imprecise task scheduling framework, (3) develop scheduling algorithms that, along with deciding what to compute, answers when to compute and when to harvest, and (4) utilize distributed systems that collaboratively emulate a persistently powered system.

Scheduling Framework for Intermittently Powered Computing Systems. Batteryless systems rely on sporadically available harvestable energy. For example, kinetic-powered motion detector sensors on the impalas can only harvest energy when the impalas are moving, which cannot be ascertained in advance. This uncertainty poses a unique real-time scheduling problem where existing real-time algorithms fail due to the interruption in execution time. This dissertation proposes a unified scheduling framework that includes both harvesting and computing.

Scheduling Mutually Exclusive Computing and Harvesting Tasks in Deadline-Aware Intermittent Systems. The lack of sufficient ambient energy to directly power the intermittent systems introduces mutually exclusive computing and charging cycles of intermittently powered systems. This introduces a challenging real-time scheduling problem where the existing real-time algorithms fail due to the lack of interruption in execution time. To address this, this dissertation

proposes Celebi, which considers the dynamics of the available energy and schedules when to harvest and when to compute in batteryless systems. Using data-driven simulation and real-world experiments, this dissertation shows that Celebi significantly increases the number of tasks that complete execution before their deadline when power was only available intermittently.

Imprecise Deep Neural Network Inference in Deadline-Aware Intermittent Systems. This dissertation proposes Zygarde- an energy-aware and outcome-aware soft-real-time imprecise deep neural network (DNN) task scheduling framework for intermittent systems. Zygarde leverages the semantic diversity of input data and layer-dependent expressiveness of deep features and infers only the necessary DNN layers based on available time and energy. Zygarde proposes a novel technique to determine the imprecise boundary at the runtime by exploiting the clustering classifiers and specialized offline training of the DNNs to minimize the loss of accuracy due to partial execution. It also proposes a single metric, η to represent a systemâs predictability that measures how close a harvester's harvesting pattern is to a constant energy source. Besides, Zygarde consists of a scheduling algorithm that takes available time, available energy, impreciseness, and the classifier's performance into account.

Persistent System Emulation with Distributed Intermittent System. Intermittently-powered sensing and computing systems go through sporadic power-on and off periods due to the uncertain availability of energy sources. Despite the recent efforts to advance time-sensitive intermittent systems, such systems fail to capture important target events when the energy is absent for a prolonged time. This event miss limits the potential usage of intermittent systems in fault-intolerant and safety-critical applications. To address this problem, this dissertation proposes Falinks, a framework that allows a swarm of distributed intermittently powered nodes to collaboratively imitate the sensing and computing capabilities of a persistently powered system. This framework provides power-on and off schedules for the swamp of intermittent nodes which has no communication capability with each other.

To my partner in crime, Tamzeed

ACKNOWLEDGEMENTS

I would like to take this opportunity to express my thanks to the people who have supported me in producing this dissertation. I especially want to acknowledge:

My advisor, Professor Shahriar Nirjon, for supporting, guiding, and funding my research. Without his guidance during numerous paper submissions, this dissertation would not have been possible.

Professors Kevin Jeffay, Donald E. Porter, Xiaofan Jiang, Parasara Sridhar Duggirala, Henry Fuchs, and Josiah Hester, who taught me extraordinary things about academia, job hunting, and scientific pursuit. Dr. Mostafa Uddin and Dr. Mahbubur Rahman for their mentorships during my internship at Nokia Bell Labs and Samsung Research America. I am forever grateful for their support, and I will remember all the excellent advice throughout my academic career.

All the present and past staff members of the Department of Computer Science at UNC, including Bil Hays, Murray Anderegg, Jim Mahaney, Mellisa Wood, Mike Carter, David Cowhig, Robin Brennan, Rosario Vila, Brett Piper and Denise Kenny. This dissertation would not exist without all of you.

My parents (Lailun Nahar and Sheikh Baharul Islam), parents-in-law (Jannatul Ferdaus and Md Rafiqul Islam), sisters (Naznin Nahar Bipasha, Halima Sadia), brothers (Shahanur Alam Ahmed, Shahruk Alam Ahmed, and Muhammad Delower Hossain Khan), aunt (Afroza Ahmed), niece (Tunaz Tareq Islam), and nephews (Saadid Arham Khan and Saahir Afran Khan), for supporting me and being there for me all my life. My friends (Samavi Farnush Bint E Naseer, Lamia Hossain, Sumaya Tasneem, and Nayna Tabassum), who have spent hours supporting me during the rough times.

Finally, my husband, Md Tamzeed Islam, for trusting my capabilities, encouraging me, and being patient with me. Tamzeed, your constant support and bottomless confidence in me empowered me to make this crazy journey. Thank you for pushing me to do better and making me laugh even during the darkest times. I could have never asked for a better life partner in my life. I love you, and I am looking forward to growing old with you.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

Introduction

The advancement in low-power computing systems works as the catalyst for the exponential growth of the internet of things (IoT), and their number will surpass one trillion by 2035 [5]. These extremely low-power tiny computing devices address the growing need for affordable, intelligent sensing and control solutions for a wide range of application domains, from infrastructure monitoring to wildlife tracking and long-term health monitoring. We envision a vast number of sensors embedded in clothing, jewelry as wearable and implantable, which will impact human, planetary, and infrastructure health. However, for mobility and ease of deployment, the current IoT world is dominated by battery-powered edge devices that are bulky and require periodic maintenance (replacing or recharging). IoT devices' boom will result in 274 million daily battery replacements [6], even with an ambitious battery lifetime of 10 years. Replacing that vast amount of batteries is unscalable, costly, and is an enormous threat to our environment when dumped in the environment [7].

Thus considering the maintenance cost, convenience, lifetime, and environmental effects, it is more logical to move from batteries and focus on available ambient energies, e.g., solar energy, thermal energy, kinetic energy, and radio frequencies. However, most ambient harvestable energy is not sufficient to power the IoT devices directly, and thus we need to accumulate adequate energy before using it. As a result, these devices go through power-on and power-off periods and experience intermittence during execution. Therefore, these systems are called intermittent systems. These systems typically consist of microcontrollers (MCUs), energy-harvesting and management circuitry, capacitors to store energy, sensors, and communication radios to transmit the data and interact with the environment. However, literature shows that communication radios are one of the most power-hungry components of batteryless systems. Instead of transmitting large raw data, sending the output is 98X more energy efficient [8]. Besides, on-device computation preserves privacy, reduces delay, and is more suitable for trillions of devices sharing the limited bandwidth. However, the intermittence of the available energy hinders the forward progress of the code execution. To ensure this forward

progress, a new paradigm of computing has emerged for these systems, known as intermittent computing [9]. Most existing works on intermittent computing systems concentrate preliminary on the lower-level goals, e.g., instruction execution and memory consistency [10, 11, 8, 12, 13], and allows successful on-device computation in intermittent systems.

By designing tiny systems that efficiently operate on intermittently available harvested energy, we can realize our vision of sustainable computing dust, which will sense, compute, and learn forever. These devices are applicable in various application domains requiring long-term sensing and inference, such as wildlife monitoring [14, 15], environment monitoring [16, 17], smart agriculture [18], infrastructure monitoring [19], wearables [20, 21, 22, 23, 24, 25, 26], and implantables [27]. Like most IoT devices, batteryless systems applications involve event detection, where they use sensors to sense the environment and detect if the target event took place. While every IoT application has an expected response time, many of them require timely feedback. For instance, in acoustic sensing systems, such as hearables and voice assistants [28, 29], car detectors [30], and machine monitors [31], events need to be detected and reported on time to ensure prompt responses, safety, and timely maintenance. Though batteryless systems are desirable in such scenarios for their prolonged lifetime, the complexity of on-device computation and unstable power supply from the ambient sources complicates the timely execution of computing tasks on such systems. Complex multitasking workloads, e.g., audio and image processing, multi-tenancy, and ensemble learning, proposed by recent works on intermittent systems [12, 32, 33, 8], further complicate the timely execution by increasing the CPU utilization.

Prior works on time-aware intermittent computing systems can be broadly categorized into two types. The first category focuses on *time-keeping*, i.e., maintaining a reliable system clock [1, 2, 3] even when the power is out. The second category includes runtime systems that consider data's temporal aspect across power failures [4] by discarding data after a predefined interval. Though works in this category consider timely execution, they focus on discarding tasks after a certain period rather than concentrating on finishing tasks on time. However, failing to process the result within the deadline makes such devices unsuitable for the applications mentioned above. To make batteryless intermittently powered devices applicable in real-world application scenarios and into the mainstream sensing applications, timely on-device execution of tasks in these devices needs to be introduced.

1.1 Research Challenges

Though all these scenarios demand a time-sensitive intermittent system, the sporadic nature of the intermittence and the high complexity intermittent tasks present unique challenges.

1. In many cases, harvesting and computation can not co-occur, which further complicates intermittent tasks' real-time execution. This mutual exclusion creates a scheduling problem where a balance between harvesting and computing is needed to avoid deadline misses due to energy or time scarcity.

2. The sporadic nature of intermittent power and the absence of continuous energy hinders the timely execution of intermittent tasks. Moreover, complex intermittent tasks, e.g., deep neural networks, 3D reconstruction, impose a heavy workload on intermittent systems' constrained resources. The sporadic nature of harvested energy, resource constraints of the embedded platform, and the computational demand of deep neural networks (DNNs) pose a unique and challenging real-time scheduling problem for which no solutions have been proposed in the literature. Though partial execution of tasks is a viable solution to meet the deadline, the variable semantic complexity of data demands dynamic portions of a task. For such a system to be feasible, this determination method must ensure minimal overhead for feasibility. Besides, developing an energy-aware scheduling algorithm requires a prediction of available energy. Direct energy prediction is not only challenging, but it is also not suitable for short periods. Moreover, the constrained memory of intermittent systems hinders tracking energy history for future prediction.

3. Due to the energy intermittence, the system might go through power-off mode when the sensing event occurs and miss the event. Mitigating such misses due to lack of power is a unique and exciting challenge as the energy source is uncontrollable and communication is expensive.

1.2 Thesis Statement

The goal of this dissertation to establish the following thesis: *"Intermittently-powered systems can ensure deadline-aware life-long sensing and computation by using unified frameworks that integrate harvesting and real-time systems, exploiting specialized characteristics of computing tasks to employ imprecise scheduling, and utilizing a cluster of distributed nodes to resemble a persistently powered system."*

Through this thesis, I present– (1) a unified framework for scheduling tasks in intermittently powered real-time systems, (2) scheduling algorithms that, along with deciding what to compute, answers when to compute and when to harvest, (3) an energy-aware imprecise task scheduling framework for deadline-aware complex task execution (e.g., deep neural network inference), and (4) a framework for emulating a persistently powered system with multiple intermittent systems. This system research focuses on bringing real-time scheduling and adaptive computing in intermittent systems, impacting the other areas of Computer Science and Computer Engineering.

1.3 Contributions

This dissertation adds a new dimension to intermittent system literature by developing intermittent aware real-time systems that ensure timely response in batteryless Internet of Things. Following are the three significant contributions I made through this dissertation.

- First, I present scheduling algorithms that, along with deciding what to compute, answers when to compute and when to harvest. First, I present a unified framework for scheduling time-aware tasks in an intermittently powered system. (Chapter 3)

- I formulate the scheduling problem for intermittent systems where harvesting and computing are mutually exclusive. To solve this, I propose an offline and an online scheduling algorithm, namely *Celebi-Offline* and *Celebi-Online*, that schedules both harvesting and computing jobs to maximize the number of jobs that meet the deadline. I further deduce necessary conditions for a taskset to be schedulable on an intermittent system. (Chapter 4)

- I devise a deadline-aware runtime framework, *Zygarde*, for sporadic deep neural network (DNN) inference on intermittently-powered systems. To ensure timely response in these devices, a runtime adaptation of a DNN is necessary on top of compile-time compression [8, 34, 35, 36, 37, 38, 39]. Compile-time compression alone is not sufficient when the remaining deadline is inadequate for full inference of the DNN but long enough to compute the inference result from the partial execution of the DNN. To achieve this I propose (1) a policy/test to determine how much partial execution at the runtime, (2) a specialized offline training of the DNNs to minimize the loss of accuracy due to partial execution, (3) a single factor (η) to model the energy harvesting pattern by representing the predictability of the system and (4) a scheduling algorithm that takes both impreciseness and energy availability into account. (Chapter 5)

- Finally, I propose a distributed scheduling framework, *Falinks*, that enables a swarm of intermittent nodes to emulate a persistently powered node without any communication collaboratively. I propose two optimal algorithms (*Duty-Cycle*, *Prime-Coprime*) that imitate a persistent node when each swarm node gets energy from a constant energy source. I formulate variable energy source scenarios as a Partially Observable Markov Decision Process (POMDP) and propose different heuristics for updating the states with the varying source. (Chapter 6)

1.4 Dissertation Overview

The rest of the dissertation is organized as follows.

Chapter 2 explains the background materials relevant to all the contributions across this thesis.

Chapter 3 first defines various types of energy, energy source, energy harvester systems, and tasks. Then it demonstrates a unified real-time scheduling framework for intermittently powered computing systems.

Chapter 4 presents a pair of scheduling algorithms that not only answers which task to compute next, but also, when to compute and when to harvest. This chapter also shows schedulability analysis of the proposed algorithms.

Chapter 5 describes Zygarde, an energy- and accuracy-aware soft real-time task scheduling framework for batteryless systems that flexibly execute deep neural network inference tasks that are suitable for running on microcontrollers.

Chapter 6 presents Falinks, a framework that allows a swarm of distributed intermittently powered nodes to collaboratively imitate the sensing and computing capabilities of a persistently-powered system. This framework provides power-on and off schedules for the swamp of intermittent nodes which has no communication capability with each other.

Chapter 7, concludes the dissertation and discuss potential future research directions.

CHAPTER 2

Background and Related Works

2.1 Intermittent Computing

Most ambient energy sources cannot provide sufficient energy to power the processor directly due to the tight size constraint on the harvester. As a result, these batteryless computing systems experience a frequent power failures. Figure 2.1 shows an intermittently powered computing system. The processor stays off until the energy storage (capacitor) accumulates enough energy to turn on the processor (T_{charge} in Figure 2.1). This phenomenon happens because the power loss corresponds to the closure trigger that closes the capacitor's energy path to the processor. Once sufficient energy is harvested, the open trigger re-opens the energy path to the processor, and the processor turns on. The processor executes until the harvested voltage reaches the minimum required operating input voltage to the processor. Note that the time to accumulate enough energy ($Tcharge$) varies as the energy available for harvesting may vary.

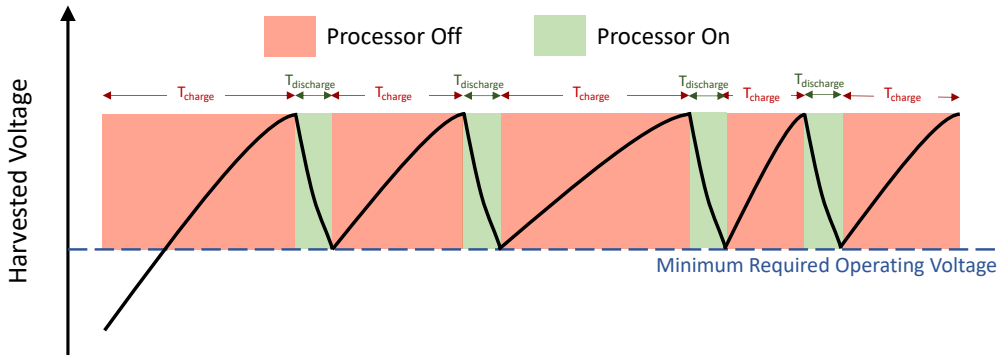


Figure 2.1: The charge-discharge cycle of energy harvesting devices forces the processor to compute intermittently.

2.1.1 Intermittent Execution Model

The frequent power failures experienced by the intermittently powered systems reset the software execution and result in repeated execution of the same code and inconsistency in non-volatile memory.

When the processor loses power at each interruption, the program loses all state variables, including program counter, which reside in volatile memory structures. The volatile memory structure includes registers, SRAM memory, caches, memory-mapped I/O registers of on-chip peripherals, and the off-chip peripheral registers. In a persistent power system at each boot-up, these program states are reinitialized, starting the program execution from the beginning. This will result in repeated execution of the same code and may cause an infinite loop where the code never gets enough energy to finish. Before the processor turns off, the program or the runtime system must save some state to non-volatile memory during the execution and restore it upon reboot to avoid this deadlock. The program state's size and saving frequency determine the intermittent execution model's time, energy, and memory overhead. The intermittence execution model literature aims to minimize this overhead while maintaining the forward execution and consistency in memory.

Existing works on the execution model can be grouped into two classes – checkpointing-based models and task-based models.

Checkpoint based Model. The checkpoint-based model's core idea is to save the program state to the non-volatile memory at the checkpoints. The checkpoints' location is either determined during the compile-time or is determined at the runtime using hardware interrupt.

Mementos [10] save periodic snapshots of the system state to non-volatile memory (NVM), enabling it to return to a previous checkpoint after a power failure in a software system. It uses compile-time execution passes to insert checkpoint to codes. It uses runtime energy estimation to consider that no energy is harvested between the trigger point and power failure. Computational RFID implements a preliminary design of Mementos [40]. Due to unnecessary periodic checkpoints, the time and energy overhead of this approach are high.

Mementos [10] saves periodic snapshots of system state to non-volatile memory (NVM), which enable it to return to a previous checkpoint after a power failure a software system. It uses compile time execution passes to insert checkpoint to codes. It uses run-time energy estimation where it considers that no energy is harvested between trigger point and power failure. [40] presents computational RFID, a preliminary design of Mementos. Due to unnecessary periodic checkpoints, the time and energy overhead of this approach is high.

To address this, Hibernus/Hibernus++ [11, 41] saves a snapshot of the system to non-volatile

memory only once, immediately before the power failure. Rather than relying on checkpointing, it depends on hardware interruption to detect the capacitor's current-voltage (V_{CC}) drops below a certain threshold. Though this approach lowers the time and energy overhead, it requires custom hardware support, which is not suitable for existing systems.

Chinchilla [42] proposed a compiler and runtime system that allows adaptive checkpointing with no additional hardware support. It disables checkpoints dynamically to adapt to energy conditions efficiently. DICE [43] uses differential checkpointing To reduce the energy cost and additional execution times of checkpoint.

While using checkpoints preserves execution progress, it may leave the non-volatile state incorrect by partially updating it. This memory inconsistency causes the behavior of an intermittent system to deviate from a persistently powered system.

Task-based Model. Task-based programming and execution models preserve the execution progress and the non-volatile memory consistency. Such models decompose a program into a collection of tasks, which are top-level atomic functions. To ensure the atomicity of these tasks, if a power failure occurs during a task's execution, the task is re-executed from the beginning. After successfully executing a task, the system tracks and atomically commit modifications to the non-volatile memory to maintain the consistency of the program state. The overhead of this approach depends on the number of atomic commits, which is known as transitions. More task transitions result in higher runtime overhead. Though larger tasks reduce the number of transitions, they re-execute more work after a power failure. Moreover, a larger task requires more energy to complete, and the energy storage often can not hold sufficient energy leading to a non-termination problem.

DINO [44] introduces a programming and execution model to simplify intermittence programming. It also ensures data consistency between volatile and non-volatile memory. Chain [45] proposed a model for intermittent programming devices. This runtime library introduces a Chain program, a set of programmer-defined discrete tasks that compute and exchange data through channels. This eliminates checkpoint costs and avoids inconsistent state, and guarantees forward progress at task granularity. Alpaca [13] is a task-based model to ensure low overhead intermittence without a checkpoint. This framework supports only single thread tasks, and task decomposition needs to be manually done by the programmer.

However, these task-based programming approaches require a programmer to decompose a program into multiple tasks. CleanCut [46] introduces a tool that automatically decomposes code to terminate the task. It supports non-terminating path bugs and uses an average case statistical energy model. Coala [47] is an adaptive and efficient task-based execution model which progresses on a multi-task scale when energy permits and preserves the computation progress on a sub-task scale if necessary.

Deep Neural Network Inference. To guarantee correct execution, a task-based model suffers from significant runtime overhead. For a computationally expensive task such as deep neural network (DNN) inference, such overhead must be avoided. To address this, SONIC [8] exploits the characteristics of DNNs to allow loop continuation while ensuring that each loop iteration is idempotent. This loop continuation minimizes task transitions and wasted work.

This dissertation uses SONIC [8] and Alpaca [13] as the intermittent execution model. The contribution in this dissertation is at the algorithmic level, which complements and enhances the existing literature.

2.1.2 Time-aware Intermittent Computing Systems

Remanence Time Keeper. Keeping time through power failure in a batteryless system is challenging. To develop a time-aware intermittent system, keeping track of time during a power failure is mandatory. A battery-powered real-time clock (RTC) is not suitable in this scenario due to the long startup time for a lower operation. TARDIS [1] uses time and remanence decay in Static Random-Access Memory (SRAM) to estimate power failure duration. CusTARD [2] uses a millimeter-scale capacitor-centered circuit to use the capacitor's energy dissipation during power failure to estimate the time as a function of capacitor voltage decay.

However, these techniques can either support a more prolonged power outage or fine grain clock resolution. Cascaded Hierarchical Remanence Timekeeper (CHRT) [3] supports more extended power outages and finer resolution by featuring an array of different RC circuits to enable multi-tier timekeeping architecture.

In Chapter 5 uses CHRT [3] for evaluation.

Data Staleness. Mayfly [4] is a declarative, task-based, and graph-inspired runtime for the timely execution of sensing tasks on a tiny, intermittently powered, energy harvesting sensing device. Mayfly

runtime maintains temporal aspects of data automatically across power failure by discarding the stale data. However, Mayfly does not aim to finish a task within a deadline and minimize stale data.

2.1.3 Runtime for Intermittent Systems

Runtime for intermittent systems aims to increase the number of completed jobs [48, 49, 32] without explicitly considering their deadlines. InK proposes a kernel for intermittent systems, which continuously executes the next task's control flow of the highest-priority task thread. Note that INK does not consider DNN tasks or utilize partial execution of tasks for increasing schedulability. Unlike INK, which schedules kernel threads and only one data sample at a time, proposed schedulers in this dissertation schedule multiple data samples present in the job queue. Moreover, this work does not consider when to harvest energy or schedule tasks from predefined priority task threads to minimize missed deadlines.

Several previous works have proposed real-time schedulers to schedule sensing and transmission tasks in batteryless sensor nodes [50, 51]. However, they only consider sense-and-send operations where only the consumed energy is considered instead of considering both the consumed energy and the execution time. This dissertation focuses on both types of demands of computing jobs.

Recently some works for batteryless sensor systems use reinforcement learning for increasing the performance of batteryless nodes. Automatic Configuration of Energy Harvesting Sensors (ACES) [52] uses Q-learning at each node for determining their duty-cycle and maximizes each intermittent nodes sensing performance. However, such algorithms are not suitable for a swarm of intermittent nodes that operated collaboratively, as shown in Chapter 6.

2.1.4 Energy Storage for Intermittent Systems

Intermittent systems primarily use supercapacitors as energy storage. The size of the capacitor plays a crucial role in the performance of the system. If the capacitor size is small, more tasks miss their deadlines as they re-execute an atomic fragment when the power goes off before its completion. On the other hand, when the capacitor value is high, tasks miss a deadline due to the extra time required to charge such a large capacitor. Moreover, different intermittent tasks have various energy requirements, which often can not be satisfied with a single-sized capacitor.

Capybara [53] is a co-designed hardware-software power system with dynamic re-configurable energy storage capacity to meet varied application energy demands using an array of programmatically controllable capacitors. Using programmer-specified energy mode allows reactive applications.

UFoP [54] presents a federated energy storage solution that allows power-hungry operations to proceed without the device's immediate ability to use other peripherals. To achieve this, it uses individual peripheral energy storage and low-power control circuitry to isolate and prioritize individual peripherals. Besides, an ultra-low-power cooperator is used to prioritize and charge individual energy storage.

2.1.5 Harvesting Energy Prediction

Previous works on energy harvesting (EH) modeling of a specific energy source has achieved promising results in predicting available energy [55, 56, 57]. Exponentially Weighted Moving Average [58, 59, 60, 61] is used to predict solar energy generation, abstract the complex time-varying nature of sources and impose duty cycle on batteryless sensor nodes. Weather-Conditioned Moving Average [62] presents a fast and reliable solar prediction algorithm for solar and considers both current and past weathering conditions and seasonal adaptation. Though some methods use clear-sky solar performance to measure the average daily solar energy, they are not suitable for short-term predictions [63]. Some other works have focused on analytically model the trade-off associated with length of history to maximize forward propagation [64]. However, none of the prior works are generalizable and fails to model energy harvesting systems irrespective of energy source. Section 5.2 provides a single metric, η factor that models an energy harvesting system's predictability irrespective of the source.

2.2 Deep Neural Network

Neural networks are hierarchical structures consisting of an input layer, one or more hidden layers, and an output layer. A network with at least two hidden layers qualifies as a "deep" neural network (DNN) [65, 66, 67, 68, 69].

The output of each layer of a DNN expresses a distinct representation of the input. This representation is fed to the next layer to obtain a new and richer representation. This hierarchical structure increases the complexity and abstraction of features as the depth increases. For instance, the first hidden layer of the DNN in Figure 2.2(a) that classifies human faces learns basic geometric features such as edges, the second hidden layer learns face parts such as nose and eyes that constitutes these edges, and the third and deeper layers learn more complex features, such as the face abstraction that constitutes of the face parts.

Because an increased number of hidden layers results in a richer representation of the input signal,

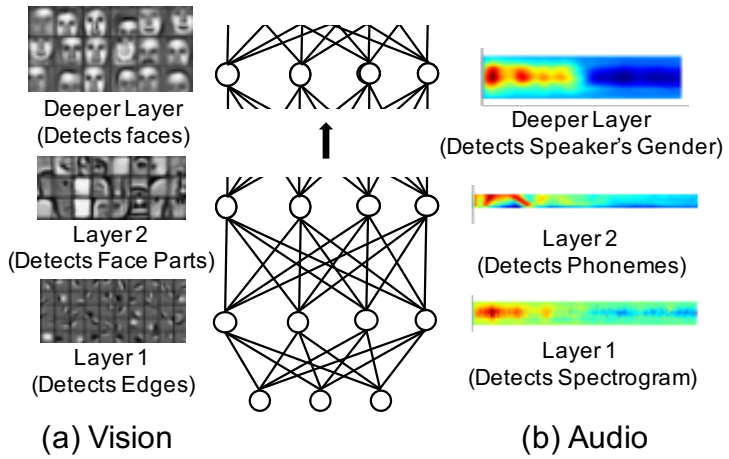


Figure 2.2: Example of DNN learns layers of features.

generally, a deeper network yields a higher classification accuracy [70]. For example, ResNet (152 layers, 2M parameters) achieves higher accuracy than VGGNet (16 layers, 140M parameters) [65]. However, after a certain tipping point, the accuracy does not increase and can drop dramatically if we continue to add new layers caused by the vanishing gradient effects [71, 72]. This phenomenon does not affect embedded learners since these memory-constrained systems typically hold a much smaller number of layers at their limit.

2.2.1 Model Compression

Recent works reduce the cost of DNN inference by pruning and splitting models [35, 73]. Hidden layers are dropped to reduce the execution cost of a DNN inference. Deep Compression [34] introduces a compression technique for a deep neural network comprising three stages – pruning, trained quantization, and Huffman coding [74]. This compression reduces the storage requirement by 35x to 49x without affecting the accuracy. The first step prunes the network by learning only the essential connections. Next, the weights are quantized to enforce weight sharing using kmeans. DeepIoT [36] is a generalized compression technique for deep learning network inference that learns the dropout probability for dropping hidden elements instead of random or pre-defined dropout probability. DyNS [75] compresses a network by pruning the connections on the fly. Other works have focused on resource optimization for DNN models to allow execution on a mobile platform [76].

The reduction of floating-point and weight precision [77, 78, 79] is another approach for compressing a DNN model. A fixed point has been implemented with 8-bit integer activation instead of a 32-bit

floating-point [80]. Ternary weights and three-bit activation optimizes a fixed-point network [81]. The neural network's linear structure contributes to finding an appropriate low-rank approximation of the parameter to compress the network [82]. The fully connected model size is reduced using a hash function to randomly group connection weights and use global max pooling [83, 84]. This way, all the parameters of the same hash bucket share a single parameter value. Factorization of computation [85, 86, 87, 88, 84, 89, 90] is another approach to compress a DNN model. Other works apply Singular value decomposition (SVD) on the weight matrices of DNN and then restructure the model based on the original matrices' inherent sparseness without negligible accuracy loss [91].

DeepX [92] is a software accelerator for deep learning inference execution based on a pair of inference-time resource control algorithms (**R**un-time **L**ayer **C**ompression and **D**eep **A**rchitecture **D**ecomposition). This accelerator dramatically lowers resource overhead by leveraging a mix of heterogeneous processors. RDeepSense [37] is a deep learning model inference that gives uncertainty estimation for resource-constrained mobile and embedded systems. It reduces the complexity by converting arbitrary fully connected NN to NN with uncertainty estimation. It uses dropout training instead of modeling ensemble (reduces computational complexity) and uses a tuneable proper scoring rule as a loss function (helps predicting uncertainty in DNN) with distribution estimation instead of point estimation as output. Predictive uncertainty is a random variable here. Apdeepsense [38] has the same goal as Rdeepsense but does not have the same need for retaining. It addresses the retraining issue by replacing the resource-hungry sampling approach with effective layerwise distribution approximation. It approximated the non-linear activation function as a piece-wise linear function. However, this will not work for RNN or CNN.

Knowledge distillation is one of the recent approaches to compress a deep neural network using a teacher and student model [93] and ensemble models [94]. Though these works are crucial for enabling fast DNN execution, they alone are not sufficient for batteryless systems. Binary networks [95, 96, 97, 98] are not suitable for batteryless systems due to the higher number of required parameters [8].

2.2.2 Adaptive Neural Network

Recent works propose early exit during DNN inference [99, 100, 101, 102]. However, they are not sufficient for highly constrained batteryless systems due to the significant termination overhead. In most cases, the terminations require a neural layer's execution requiring 45 times

more execution cycles than performing utility tests and classification with our proposed algorithm in Chapter 5. Moreover, these approaches lack model adaptation capability, which is required for life-long sensing. Some works on anytime neural networks depend on module selection [103, 104, 105], dynamic layer pruning during inference [106, 107, 100, 108, 105], and depth and width adjusting techniques [109, 110, 111, 112, 113] for anytime prediction. However, none of these works have considered the effect of energy intermittence, and they are yet to be modeled as an imprecise task.

2.3 Real-time Scheduling

Systems in which the computations' correctness depends on logical correctness and temporal correctness are known as real-time systems. Here, logical correctness impels that the system produces correct output, and temporal correctness denotes that the system produces outputs at the right time. In a real-time system, the value of a computation depends not only on the answer's correctness but also on how timely it is. In such a system, late completion of computation has diminishing or no value, while early completion of computation has no extra value.

2.3.1 Definitions

This section describes the definition of different terms used in real-time systems.

Task. The workload of a real-time system is known as the task. In other words, a task is a sequential piece of code that executes in a system.

Job. A job is an instance of a task that requires resources, e.g., processors, to execute.

Release Time of Arrival Time of a Job. Release time or arrival time of a job is when the job becomes ready to execute.

Deadline of a Job. The time instant by which a job must complete execution is known as the deadline of that job.

Periodic, Sporadic, and Aperiodic Task. A task can be of three types – periodic, aperiodic, and sporadic. A task is periodic when two consecutive jobs have a fixed and known time difference between their release/arrival time. This time is known as the period. A sporadic task has a known minimum inter-arrival time among successive instances of a (periodic) task instead of strictly being periodic. In the worst-case scenario, a sporadic task performs as a periodic task. An aperiodic task is event-driven, where the release time of the jobs is unknown.

Implicit, Constrained, and Arbitrary Deadline System. In an implicit-deadline system, the deadline is equaled to the period for each task. On the other hand, in a constrained-deadline system, the deadline is less than or equals to the period. There is no relation between the deadline and periods in an arbitrary-deadline system.

CPU Utilization. Utilization of a task is the ratio between the execution time and period of the task. The CPU utilization of a taskset is the summation of all the task utilization in a taskset.

This dissertation focuses on periodic and sporadic tasks with implicit, constrained, and arbitrary deadline systems.

2.3.2 Task Scheduling Algorithms

Earliest Deadline First (EDF) Scheduling Algorithm. Earliest Deadline First (EDF) is a dynamic priority scheduling algorithm where a task with a shorter deadline has a higher priority. It executes a job with the earliest deadline. EDF is the optimal online scheduling algorithm for a single processor in persistently powered systems. A schedule is optimal if the optimal scheduler must schedule a taskset scheduled by any other scheduler. A real-time system is schedulable under EDF if and only if $\sum U_i \leq 1$, where U_i is the CPU utilization of task i.

Rate Monotonic (RM) Scheduling Algorithm. Rate Monotonic (RM) scheduling algorithms prioritize the task with a smaller deadline or period. RM is a static priority scheduling algorithm, and the static priorities are assigned according to the cycle duration of the job, so a shorter cycle duration results in a higher job priority.

As-Late-As-Possible (ALAP) Scheduling Algorithm. As-Late-As-Possible (ALAP) scheduling algorithm schedules each job at the latest opportunity. ALAP scheduling algorithm delays a task as much as possible without violating the deadline constraint.

2.3.3 Imprecise Job Scheduling

Imprecise computation models divide each task into two portions – mandatory and optional. By meeting its deadline, such a model refers to executing the mandatory part of a task before its deadline. After the mandatory portion is scheduled, imprecise schedulers try to schedule as much as an optional portion as possible. If necessary, the optional portion can be terminated before it is completed for the task and other tasks to meet their deadline. The result of a prematurely terminated task has an error which is a non-increasing function of processing time. These sorts

of tasks are called monotone tasks. A task is a monotone if the quality of its intermediate result does not decrease as it executes longer. In this model, a feasible schedule refers to completing the mandatory portion of every task by its deadline.

Online scheduling of imprecise tasks has been studied where the goal is to minimize the total error of all tasks [114]. This scheduler assumes that the system does not accept tasks whose mandatory portions cannot be feasibly scheduled at the arrival time. Therefore, the task system presented to the scheduler satisfies the feasible mandatory constraints. The scheduler reserves a mandatory interval with reverse scheduling and then uses EDF to schedule optional portions in the rest of the time intervals. A feedback-driven online scheduler for sporadic real-time processes with imprecise computing assumes that processes are ready for execution upon their arrival [115]. For the feedback controller, which controls the admission control, this scheduler considers the number of processes that missed the deadline and the number of processes that met the deadline. Another online feedback-driven scheme to schedule a process with imprecise computation utilizes a PID controller that takes the missing ratio as input from the scheduler(EDF) and then calculates how long the optional part should be executed [116]. The goal is to bind the deadline-miss ratio to a set point so that a balanced trade-off between the precision of computation and CPU utilization exists. A semi-priority scheduling algorithm based on a rate monotonic schedule for imprecise tasks achieves high schedulability [117]. However, in all these works, mandatory-optional partitions are fixed, determined in the compile time, and known a priori. In Chapter 5, the proposed algorithm determines the dynamic imprecise boundary at the runtime.

Existing works on Quality of Service (QoS) based resource management [118, 119] do not handle the data-dependent dynamic relationship between quality and required execution. Previous works on mixed-critical systems [120, 121, 122, 123, 124, 125] propose scheduling schemes for predetermines critical levels, which is a characteristic of a task. However, these algorithms are not suitable, then the performance of the system varies for each job.

2.3.4 Deep Neural Network Scheduling

Several works focus on scheduling deep neural network tasks within a deadline. ApNet [126] is a timing-predictable runtime system to guarantee deadlines of DNN via efficient approximation built upon the theoretical analysis of a multi-layer DNN end-to-end framework. This paper also exploits that resource sharing and approximation can mutually supplement one another in a multitasking

environment. The key observation in this paper is that the runtime response of each layer of a DNN instance to approximation is different; in other words, the approximation potential of various layers is distinctive. ApNet divides the end-to-end deadline of a task into sub-deadline for each layer. S^3DNN [127] (Supervised Streaming and Scheduling for DNN) optimizes DNN workloads on GPU in a real-time multitasking environment. Its goal is to optimize real-time correctness and throughput simultaneously. S^3DNN extends least-slack first (LSF) into a kernel-level LSF algorithm to prioritize and schedule multiple DNN instances. It schedules workload in the granularity of GPU kernels and dynamically aggregates underutilized kernels. By doing this, it maximizes throughput and GPU resource utilization. However, these works focus on GPUs and do not consider resource-constrained systems, unlike the proposed algorithm in Chapter 5.

CHAPTER 3

Scheduling Framework for Intermittently Powered Computing Systems

This chapter first defines various types of energy, energy sources, and energy harvester systems. Next, we introduce and formulate different types of tasks. Finally, it provides a scheduling framework for intermittent systems and describe the major components.

3.1 Energy and Energy Sources

I classify types of energy, energy sources, and energy harvested systems below.

3.1.1 Types of Energy

I define two types of energy for ease of development – harvestable and harvested energy.

Harvestable Energy. The amount of energy available to be harvested from energy sources is defined as harvestable energy. To simplify scheduling and analysis, I quantize the harvestable energy into discrete levels. I divide the total energy for each time slot by a constant (unit energy) and express harvestable energy at each time slot as an integer. The time slots' length is constant and depends on the shortest execution time and the lowest energy consumption of a task.

Harvested Energy. The harvested energy is the energy harvested by the system and stored in its energy storage (e.g., supercapacitor). The MCU consumes this energy to execute jobs. Note that the harvested energy is not just a cumulative sum of the harvestable energy since (1) it changes as energy is consumed by the MCU, and (2) the system may decide not to harvest energy at a time slot even though there is available harvestable energy. I denote harvested energy at time t as E_t.

3.1.2 Types of Energy Sources

I categorize the energy source for harvesting energy based on stochasticity and ubiquity.

Categorizing Energy Source based on Stochasticity. Depending on the energy source's stochasticity, I categorize harvesting energy sources into two classes – constant energy source and variable energy source. An energy source is constant if the harvestable energy does not vary over time. An RF harvester at a constant distance from the RF harvester without any interference

receives constant harvestable energy. Thus in this scenario, the RF harvester is a constant source. On the other hand, when the harvestable energy changes with time, the energy source is variable. The harvestable energy of an energy-harvesting tile that harvests kinetic energy from the pedestrians is a variable energy source.

Categorizing Energy Source based on Ubiquity. Based on the ubiquity of the harvestable energy to all the nodes in a swarm, I have further categorized the energy sources into two categories – balanced and unbalanced. This categorization is practical for a swarm of intermittent nodes. An energy harvesting source is balanced if the harvestable energy at any node is the same at any point in time. In other words, if all nodes in the swarm have access to equal harvestable energy, the energy source is balanced. A swarm on solar-powered nodes in an open field is an example of a balanced energy source. If the nodes in a swarm get different harvestable energy simultaneously, the energy source is unbalanced. A group of RF harvester placed at a different distance from the RF transmitter gets unbalanced harvestable energy.

3.1.3 Types of Energy Harvester Systems

There are two popular designs for energy harvester systems – direct-usage-based design and energy storage-based design [9]. The direct-usage-based design is the most straightforward design for an energy harvester system where the harvester output is directly connected to the load. However, this design is not widely used as it wastes energy when the harvestable energy is not equal to the required energy to run the system. To illustrate, when harvestable energy $<$ required energy, the harvestable energy can neither be used nor be stored due to the absence of energy storage. Similarly, when harvestable energy $>$ required energy, this design only uses the energy required to execute the system and wastes the access energy that could have been stored if energy storage was present.

On the contrary, in the energy-storage-based design, the load is usually decoupled from the harvester by an energy buffer, e.g., a capacitor and hardware or software-based controllers control the charging and discharging of the storage element. Most intermittent computing systems [9, 12, 33, 8, 14, 53, 20, 128] use an energy-storage-based design. In this design, when harvestable energy is less than the required energy, the capacitor continues to harvest energy to the storage until it accumulates enough energy to run the system. When the harvestable energy is high, the capacitor stores the excess energy for future use. Therefore, I use an energy-storage-based energy harvester

system in this dissertation.

3.2 Tasks

An intermittent computing system can have three types of tasks – (1) computing task, (2) harvesting task, and (3) NOP task.

3.2.1 Computing Task

I consider the processing of a data stream from a sensor on the device as a computing task. These computing tasks are sporadic in nature and a task is denoted by $\tau_i = (T_i, D_i, c_i, e_i)$, where T_i denotes the period (i.e., the minimum separation between two consecutive jobs), D_i is the relative deadline, c_i is the worst-case execution time, and e_i is the energy consumption rate (i.e., power). The hyperperiod is the least common multiple of the periods and is denoted by $\overline{T}$. Deadlines can be both implicit where the deadline equals the period, i.e., $T_i = D_i$, and explicit where the deadline equals the period, i.e., $T_i > D_i$.

An instance of a task τ_i, aka a job, is defines as $j_{ik} = (a_{ik}, d_{ik}, c_i, e_i)$, where a_{ik} is the arrival time, d_{ik} is the absolute deadline, c_i is the computation time and e_i is the energy consumption rate. A job misses its deadline if it fails to execute for c_i units of time before the deadline d_{ik}.

Intermittency Management for Computing Tasks. The size of a typical job is generally too large to execute without intermittence. Hence, at the implementation level, to avoid corrupted results and ensure forward code execution progress, these units are further divided into atomically executable *fragments*—which guarantees correct intermittent execution using ALPACA [13] and SONIC [8] APIs.

Task Preemption. We allow limited preemption [129] of computing jobs where a computing job can be preempted by another preemptive job only at certain instances, aka a *unit*. A unit represents a logical grouping of related modules of the task (more details in Chapter 5). By prohibiting a unit's preemption by another computing job and using double buffering [130], I reduce context switching and read-write overheads and minimize the memory requirements to $\mathcal{O}(N)$ for N computing jobs. Note that harvesting and NOP jobs can preempt a computing job at any point, including within a unit. However, harvesting and NOP jobs are non-preemptive themselves.

3.2.2 Harvesting Task.

Energy harvesting aka charging cycles is defined as harvesting tasks. A harvesting job $h_i = (a_i, d_i, c_i, e_i)$. Here, a_i is the arrival time of the job, c_i is the execution time or the length of the charging cycle, d_i denotes the deadline where $d_i = a_i + c_i$, and e_i is the rate of available energy that is harvested by the harvesting job h_i. Harvesting tasks are non-preemptive and have the highest priority. Based on the relation between the available energy and the computational need, harvesting jobs and computing jobs can occur concurrently or are mutually exclusive.

3.2.3 NOP Task.

When the harvested energy is insufficient to execute a computing job and the harvestable energy is zero, no computing or harvesting job can occur. We consider such cases as NOP tasks.

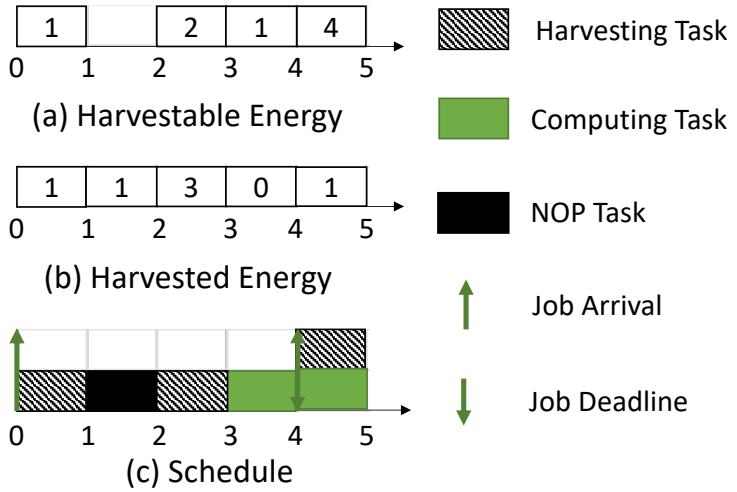


Figure 3.1: An example of the energy and task model over 5 time units. The computing task $\tau_1 = (4, 4, 1, 3)$.

3.2.4 Example

Figure 3.1 shows an example of the energy and task models for an intermittent system. Figure 3.1(a) shows the harvestable energy at different time slots, and Figure 3.1(b) shows the harvested energy (i.e., energy stored in the capacitor) at each time slot. Figure 3.1(c) shows two jobs, j_{11}=(0, 4, 1, 3) and j_{12}=(4, 8, 1, 3), of the computing task $\tau_1 = (4, 4, 1, 3)$, four harvesting jobs– $h_1 = (0, 1, 1, 1)$, $h_2 = (2, 3, 1, 2)$, $h_3 = (3, 4, 1, 1)$, and $h_4 = (4, 5, 4, 1)$, and a NOP job.

At $t = 0$, harvesting job h_1 harvests 1 unit of harvestable energy, and the harvested energy becomes $E_0 = 1$. At $t = 1$, the harvestable energy is 0, which is insufficient to run the MCU. Thus, a NOP task takes place, and harvested energy E_1 remains the same as E_0. During $t = 2$, the

harvesting job h_2 takes place, and the harvested energy increases to $E_2 = 3$. The MCU consumes **3 units of energy** at $t = 3$ by executing the computing job j_{11}, which reduces the harvested energy to $E_3 = 0$. As the harvestable energy is less than the computing job's energy consumption rate, it cannot harvest at $t = 3$. At $t = 4$, harvestable energy is more than the computing job's energy consumption rate, and thus, computing job j_{12} and harvesting job h_4 execute simultaneously. j_{12} consumes three units of energy harvested by h_4, and the harvested energy becomes $E_4 = 1$.

3.3 Scheduling Framework for Intermittent Systems

Figure 3.2 shows the eight major components of a time-aware intermittent computing system: (1) one or more sensors, (2) a job generator, (3) one or more energy harvester, (4) energy storage, (5) an energy manager, (6) a processing unit, (7) a timekeeper, and (8) a scheduling unit. This section provides a generic description of each of these components. The later chapters provide more detailed descriptions as needed.

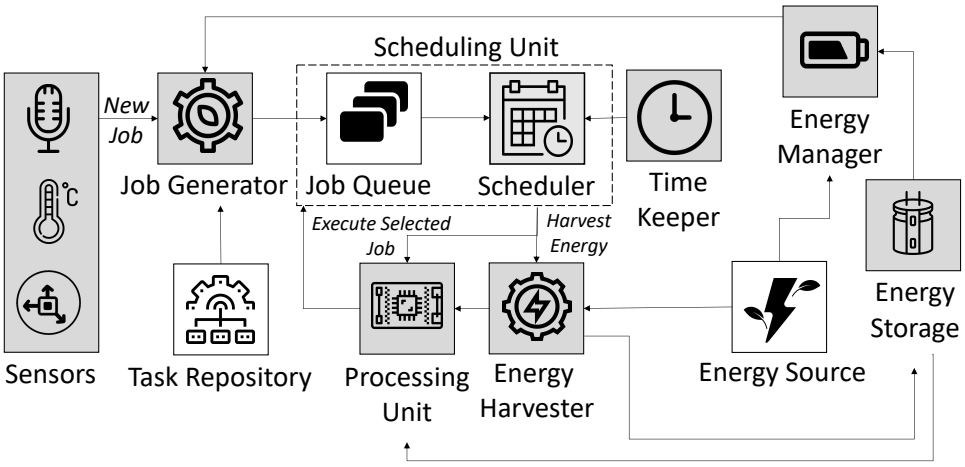


Figure 3.2: Scheduling Framework for Intermittent Systems.

Sensors. In real-life scenarios, sensors collect data from the environment and for analyzing or classifying it. The choice of a sensor highly depends on the application of the system. Some popular sensors used in battery-less devices are light sensors [131], temperature sensors [132], audio sensors [133], accelerometers [134], pressure sensors [135], image sensors [136], CO sensors [137]. Table 1 shows the power consumption of different sensors used in the literature.

Sensors collect either analog or digital signals at a given sampling frequency. This sampling

Table 3.1: Various sensors and their power consumption.

Sensor	**Power**
Ambient Light [138, 131]	2.34μW
Temperature Sensor [139, 132]	27.6μW
Audio Sensor [140, 133]	0.65mW
Accelarometer [141, 134]	0.9 mW
Pressure Sensor [142, 135]	27μW
Image Sensor [143, 136]	4mW
CO Sensor [144, 137]	45μW

frequency varies based on the types of sensors, e.g., accelerometers usually have a 100 Hz sampling rate while a more standard sampling rate for acoustic signals 11.25 KHz - 44.10 KHz. Sensors that collect analog signal requires an analog to digital converter (ADC) to convert the analog signal to digital signals which the processors can use. The granularity of the signal measurement depends on the sampling frequency of the ADC.

Job Generator. The system reads data from one or more sensors (e.g., microphone and accelerometer) and processes it using one or more preloaded computing tasks from the *Task Repository*. For example, a smart earbud may run two tasks – speaker recognition and hotword detection – both using the same microphone. We define a sensor stream's processing pipeline as a computing task and end-to-end processing of a sensor data sample as a computing job. Thus, if this system (having two tasks) generates k audio frames/second, then after 3 seconds, there will be a total of $6k$ jobs. The Job Generator creates and enqueues jobs into the *Job Queue*, which is part of the scheduling unit. This process includes writing the sensor data to the non-volatile memory (FRAM) of the microcontroller using the direct memory access (DMA) for computing jobs.

For creating and enqueuing the harvesting jobs, the job generator relies on the energy manager. Whenever there is harvestable energy, the job generator creates a harvesting job. Finally, when both harvesting and computing jobs are absent, the job generator creates a NOP job. A job leaves the queue when it gets scheduled for execution, or its deadline has passed.

Energy Harvester. One of the most popular energy harvesting techniques is to convert solar energy into electrical energy [145, 146]. Even though solar energy is uncontrollable (we can not control the sun's intensity or the occlusion by cloud), it has certain reliability based on time and season. Other popular technologies include converting kinetic energy (e.g. human step [147, 148, 149], wind [150, 151]), thermal energy [152, 153], radiofrequency energy [154, 10, 41, 136] to electric energy.

In Table 3.2 shows different energy harvester source and their generation capacity [155, 156].

Table 3.2: Various energy harvesters, their energy sources, and harvested energy.

Energy Source	**Harvesting Technology**	**Harvested Energy**
Light [157, 150, 158, 159]	Solar Cell	15mW/cm^2
Kinetic (Wind) [150]	Anemometer	1200mWh/day
Kinetic (Human Motion) [147, 149, 160]	Piezoelectric	2.1mW
Kinetic (Vibration in Indoor) [161]	Electromagnetic Induction	0.2mW/cm^2
Thermal [162, 153, 160]	Thermoelectric Generator	25μW/cm^2 – 10mW/cm^2
RF [154]	RF Energy Harvester	0.1μW/cm^2 – 1mW/cm^2

Energy Storage. Rechargeable batteries and supercapacitors are the most viable choices as the energy source of an energy harvesting system. Batteries utilize chemical reactions to store energy and have a limited cycle life. On the other hand, supercapacitors store energy by physical-charge storage and have an effectively infinite cycle life. Besides, supercapacitors are ideal power buffers between an energy harvester and a load demanding more power than the energy harvester can deliver due to its low equivalent series resistance to enable high power delivery, high capacitance to support peak power demand for the required duration, low leakage current, simple charging, and smaller footprint.

However, the size of the capacitor plays a crucial role in an intermittent system. If the capacitor size is too large, it requires more time to charge, and the system stays off for a long time. On the other hand, if the capacitor is too small, though the system turns on more frequently, it exhausts the available energy quickly and suffers from higher intermittence overhead. Section 5.7.6 shows the effect of different supercapacitor sizes with experiments.

Though Capybara [53] proposes an array of supercapacitors to address the abovementioned challenge to some extent, it fails to consider different types of energy harvesting systems mentioned in Section 2. To address this, Section 6.4 of Chapter 6 proposes a software-controlled cascading capacitor array.

Energy Manager. The Energy Manager monitors the *energy storage* state (e.g., supercapacitor or capacitor array [53]) and measures the harvestable energy from the *energy harvester*. The scheduler uses this information for scheduling decisions (described in Section 5.4). Besides, the job generator uses the predicted harvestable energy to generate harvesting and NOP jobs. The energy manager implements an open-source intermittent computing runtime [8, 13] to manage the execution of jobs

across power failures. Each job consists of multiple small atomic fragments that maintain a strict precedence order at the implementation level. These fragments execute atomically, and the runtime ensures that repeated attempts to execute a fragment are idempotent.

Processing Unit. The processing unit or microcontroller (MCU)draws power from either the energy storage or the energy harvester or both and executes the scheduled computing jobs. When no computing job is scheduled, the processing unit goes to the low-power mode or sleep mode to preserve energy. Chapter 5 describes the sub-components of the processing unit for inferring deep neural networks imprecisely.

Timekeeper. The most straightforward way to address this problem is to use an external low-power real-time clock (e.g., NXP PCF2123 [163], DS3231 [164], or Abracon AB08X5 [165] RTC chip), which is powered by an external coin cell. However, as the battery might wear out and add an external clock with the battery significantly increases the device footprint, several batteryless timekeepers have been introduced. Tardis exploits SRAM decay during a power failure, while CusTARD measures the amount of voltage decay on a dedicated capacitor using an analog-to-digital-converter after a power failure to estimate elapsed time. Recently CHRT has been proposed that uses remanence energy of a capacitor-resistor array to support longer elapsed time along with finer granularity.

Scheduling Unit. The scheduling unit consists of a job queue and a scheduler. The job generator adds jobs to the job queue. Besides, when a job is preempted, it again enters the job queue so that the remaining portion can the scheduled for execution. The scheduler is a dynamic priority real-time scheduler that considers the timing aspects, the expected performance of a job, and the system's energy harvesting status. The scheduler schedules jobs present in the job queue. The main contribution of this dissertation lies in this scheduler. The proposed scheduler decides how much of a computing job needs to be executed (details in Chapter 5). Along with answering which computing job to execute, this scheduler also determines when to compute and when to harvest when harvesting and computing are mutually exclusive (details in Chapter 4) or when a swarm of intermittent computing nodes operates collaboratively (details in Chapter 6).

Figure 3.3 shows an overview of how the scheduler works in an intermittent system. In this figure, P_G and P_C are the harvestable energy and energy consumption rate by the computing tasks (τ_1 and τ_2). This example shows the first job of each task ($j_{1,1}$ and $j_{2,1}$). $\{\tau_H\}$ is the harvesting

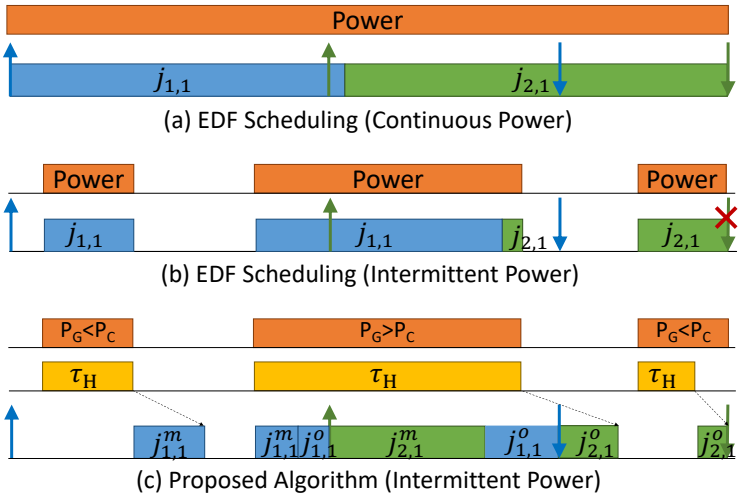


Figure 3.3: Overview of the scheduler.

task. The scheduler decides how much of a task to execute by partitioning each task into mandatory ($\tau_i^m = \{j_{1,1}^m, j_{2,1}^m\}$) and optional portions ($\tau_i^o = \{j_{1,1}^o, j_{2,1}^o\}$), where executing the mandatory portion in time is acceptable. Chapter 5 describes how we schedule the computing tasks. This figure demonstrates two cases: (a) when $P_G \geq P_C$, τ_H executes concurrently with $\{\tau_i^m, \tau_i^O\}$ and harvests the excess energy; and (b) when $P_G < P_C$, $\{\tau_H\}$ has to harvest energy before $\{\tau_i\}$ can be executed. However, there is a trade-off – if we always harvest, there will be sufficient energy but inadequate time to execute jobs of $\{\tau_i\}$. In contrast, if we always execute computing jobs and do not harvest, there may not be enough energy to finish the jobs. In Chapter 4, I answer this question by proposing a pair of scheduling algorithms. Figure 3.3 shows: (a) $j_{1,1}$ and $j_{2,1}$ are schedulable by EDF when power is always on; (b) $j_{2,1}$ misses the deadline when energy is intermittent; and (c) how the proposed algorithm schedules τ_H, $j_{1,1}^m$ and $j_{2,1}^m$, and some portion of of $j_{1,1}^o$ and$j_{2,1}^o$.

3.4 Summary

This chapter models the energy, energy source, energy harvesting system and three types of tasks for an intermittent system. Then it introduces the a scheduling framework for realtime tasks in an intermittently powered system and describe the major components of the framework with example.

CHAPTER 4

Scheduling Mutually Exclusive Computing and Harvesting Tasks in Deadline-Aware Intermittent Systems

The sporadic nature of harvestable energy and the mutually exclusive computing and charging cycles of intermittently powered systems pose a unique and challenging real-time scheduling problem where the existing real-time algorithms fails due to the lack of interruption in execution time. This mutual exclusion is introduced by storage-based energy harvesting system where a capacitor is used to store the harvested energy when the harvestable energy rate (supply) is less than the energy consumption rate (demand). Though other cases where harvestable energy rate is higher than consumed energy rate exists, this work focuses on the demand > supply as many intermittent systems follows it [8, 12, 13, 4]. The significant research question is – *how to integrate the variability of sporadic harvestable energy in the scheduler to harvest required minimum amount of energy while maximizing schedulability of the jobs?*

Through an array of observations and experiments, I develop scheduling algorithms that schedule both computational and energy harvesting tasks by harvesting the required minimum amount of energy while maximizing the schedulability of computational jobs to maximizes the number of jobs that meets deadline for both known and unknown harvestable energy pattern.

This is the first work that schedules not only the computing cycles but also the energy harvesting cycles of an intermittent system by considering the dynamic properties of the environment and the harvestable energy. Through this work, I make three significant contributions.

• First, I formulate the scheduling problem and deduce necessary conditions for a taskset to be schedulable on an intermittent system.

• Second, I propose an offline scheduling algorithm, namely *Celebi-Offline*, that schedules both harvesting and computing jobs to maximize the number of jobs that meet the deadline. To achieve this goal, after an initial round of scheduling, I iteratively remove energy harvesting cycles that harvest extra energy and accommodate computing jobs so that they can meet their deadlines.

• Finally, I present an online version of *Celebi-Offline* scheduling algorithm called the *Celebi-Online*, where the harvestable energy pattern is not known a priori. It is a threshold-based algorithm that avoids situations where no energy is available to harvest, and the harvested energy is not sufficient to run the system. It opportunistically execute computing tasks earlier than it is scheduled, when the harvestable energy is below the threshold.

I implement *Celebi* in a TI MSP430 microcontroller which is powered by harvesting solar energy. I implement four different complex sensing and computational applications along with system tasks, e.g., maintaining clocks, monitoring energy. These applications include – temperature anomaly detector, DNN based acoustic event classifier, RSA encryption, and bit counter.

I compare our scheduling algorithms with an optimal scheduler and three baseline schedulers (earliest deadline first, rate monotonic and as late as possible) in simulation, trace-based, and real-life experiments. *Celebi-Offline* scheduler, on average, shows 92% similar performance as the optimal scheduler in controlled experiments. *Celebi-Online* scheduler schedules 8% – 22% more jobs than baseline schedulers in controlled evaluation. Finally, in real-life evaluation, *Celebi-Online* performs 63% better than a non-real-time system and 8% better than the baseline scheduler.

To evaluate *Celebi*, I conduct simulation as well as trace-based and real-life experiments. Our results show that the proposed *Celebi-Offline* algorithm has 92% similar performance as an optimal scheduler, and *Celebi-Online* scheduler schedules 8% – 22% more jobs than the earliest deadline first (EDF), rate monotonic (RM), and as late as possible (ALAP) scheduling algorithms. I deployed solar-powered batteryless systems where four intermittent applications are executed in the TI-MSP430FR5994 microcontroller and demonstrate that the system with *Celebi-Online* misses 63% less deadline than a non-realtime system and 8% less deadline than the system with a baseline (as late as possible) scheduler.

4.1 Formulation of Scheduling Problem for Intermittent Systems

In this section, I describe the scheduling problem along with the assumptions and an example schedule.

4.1.1 Assumptions

• **A1:** *The energy consumption rate is higher than the energy generation rate.* Energy harvesters that power intermittent systems harvest energy at the rate of μW to mW (e.g. solar cell [166, 133], RF

signal [167, 168, 169, 170, 171, 172], and piezoelectric [173, 174]) harvesters harvest 2.5mW–1mW, 0.1μW–1mW, and 0.2mW–2.1mW, respectively. The active-mode power consumption of an MCU on the other hand is ≈6mW [175]. Hence, for most intermittent systems [176, 177, 42], the energy consumption rate is higher than the energy generation rate.

• **A2:** *Harvesting and computing jobs are mutually exclusive.* Due to the hardware design choices, such systems exist. For example, intermittent computing systems with a single capacitor [178, 13] have mulutally exclusive harvesting and computing tasks. In storage-based models (Section 3.1.3 – Energy Harvested System Design Choice), where the energy consumption rate is higher than the energy generation rate, the mutual exclusion between harvesting and computing tasks is a fundamental characteristic. Other systems where harvesting and computing may happen simultaneously are not the target of this paper.

• **A3:** *The capacitor has a fixed charging rate.* A capacitor's charging rate is not linear but it decreases as the voltage across it increases. However, to simplify the scheduling and analysis, I consider a fixed charging rate. The storage is assumed to be sufficiently large.

• **A4:** *For the offline scheduling algorithm, the harvestable energy pattern is assumed to be known a priori.* Estimating the energy harvesting pattern is an unsolved problem. Many [58, 59, 60, 61, 63, 179, 180] have achieved up to ≈90% accuracy in estimating the energy generation pattern. For analysis purpose, the offline scheduling algorithm considers that the harvestable energy pattern in known. Later in this chapter, I provide an online scheduling algorithm where this assumption is lifted.

4.1.2 Problem Formulation

I formulate an optimization problem that maximizes the number of computing jobs that meet the deadline given a set of computing (J) and harvesting (H) jobs.

The decision variables are defined as follows:

- $x_{jt} \in \{0, 1\}$ indicates whether job $j \in J \cup H$ execute ($x_{jt} = 1$) or not ($x_{jt} = 0$) at time t.
- $R_j \in \{0, 1\}$ indicates whether job $j \in J \cup H$ executed fully. $R_j = 1$ when $\sum_t x_{jt} = c_j$, and $R_j = 0$ otherwise.
- $z_j \in \{-1, +1\}$, where $z_j = -1$ when $j \in J$ and $z_j = +1$ when $j \in H$.

The optimization problem is expressed as follows:

$$\max \quad \sum_{j \in J} R_j \tag{4.1}$$

$$\text{s.t.} \quad \sum_{j \in J \cup H} x_{jt} \leq 1, \qquad \forall t \in \overline{T} \tag{4.2}$$

$$\sum_{t \in \overline{T}} x_{jt} \in \{0, c_j\}, \qquad \forall j \in J \cup H \tag{4.3}$$

$$\sum_{n=0}^{t} \sum_{j \in J \cup H} z_j e_j x_{jn} \geq 0, \qquad \forall t \in \overline{T} \tag{4.4}$$

$$(x_{jt} = 1) \implies a_j \leq t \leq d_j, \qquad \forall t \in \overline{T}, \forall j \in J \cup H \tag{4.5}$$

• *Objective Function.* The objective function is expressed by Equation (1), which maximizes the number of computing jobs that get completed.

• *Task Constraint.* Equation (2) ensures that only one task can execute at any time slot.

• *Execution Time Constraint.* Equation (3) ensures that a job either executes fully or not at all.

• *Energy Constraint.* Equation (4) ensures that the harvested energy is always non-negative.

• *Deadline Constraint.* Equation (5) ensures that no job is scheduled before its arrival or after its deadline.

To solve this optimization problem, I use a linear programming solver [181] which uses simplex algorithm. The worst-case computational complexity of the simplex algorithm is exponential, although it can solve most problems in cubic time [182]. Such a computational cost is not feasible for larger jobsets and larger hyperperiods.

4.1.3 Example

Figure 4.1 shows a task set having two tasks τ_1 = (10, 10, 1, 6) and τ_2 = (20, 20, 3, 3). The hyperperiod $\overline{T}$= 20 and there are three jobs: j_{11} = (0, 10, 1, 6), j_{12} = (10, 20, 1, 6) and j_{21} = (0, 20, 3, 3). Figure 4.1(a) shows the schedule and harvested energy when the jobs are scheduled using EDF. Here, j_{12} misses the deadline due to the scarcity of energy. In Figure 4.1(b), j_{12} misses the deadline when scheduled by a lazy scheduling algorithm that schedules a job as late as possible before the deadline. In Figure 4.1(c), j_{21} misses the deadline due insufficient energy. Finally, Figure 4.1(d) shows an optimal schedule which is obtained by solving Equations (1) – (5).

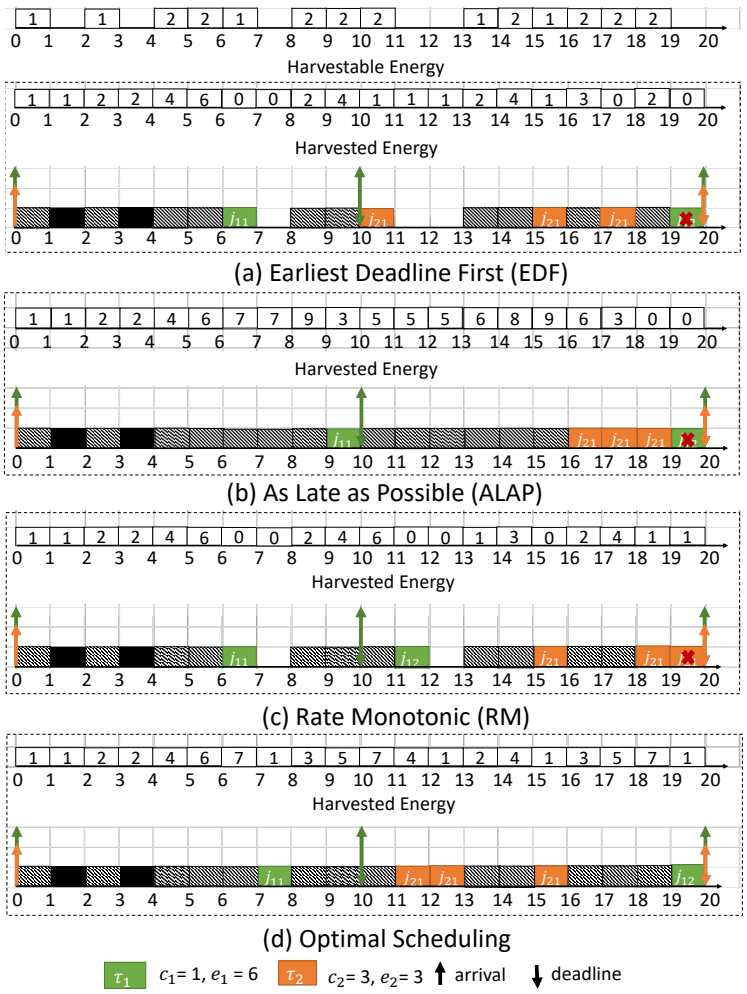


Figure 4.1: Comparison of different types of scheduling algorithms.

4.2 Observations

This section describes some key observations, which are later utilized to design the scheduling algorithms.

Theorem 1. *If the task, execution time, energy, and deadline constraints are satisfied, for all optimal schedules, a computing job is scheduled at time t when harvestable energy is zero.*

Proof of Theorem 1. I prove this by contradiction. I assume that a computing job j is schedulable at time t_m or t_n, where harvestable energy at t_m and t_n are k and 0, respectively. Let us assume that scheduling j at t_m is optimal. There are two cases.

In the first case, t_m occurs before t_n. The harvested energy at $(t_m - 1)$ is $\overline{E}$. Thus, the harvested energy at t_n, $E_n = \overline{E} - e_j$ and a computing job j' where $e'_j = \overline{E} - e_j + k$ cannot be scheduled. On

the other hand, by scheduling j at t_n, the harvested energy at t_n becomes $\overline{E} - e_j + k$, which is sufficient to execute j'. This contradicts our assumption.

In the second case t_n occurs before t_m and the harvested energy at $t_n - 1$ is $\overline{E}$. Like the previous case, the harvested energy at t_m is $\overline{E} - e_j$ which is not sufficient for executing j'. On the contrary, scheduling at t_n provides sufficient harvested energy $\overline{E} - e_j + k$ to execute job j' at $t_m + 1$, which contradicts our assumption. □

Theorem 2: *If a jobset is schedulable when harvesting and computing jobs are mutually exclusive, it is also schedulable when harvesting and computing tasks can occur concurrently.*

Proof of Theorem 2. I prove this theorem by contradiction. Let us assume that a computing jobset J is unschedulable when computing and harvesting jobs are not mutually exclusive and schedulable when they are mutually exclusive. Let, j be the first job that misses deadline, and $j - 1$ be the previous job that meets the deadline. A deadline miss occurs if the processor is not available for job j between its arrival a_j and deadline d_j for t_j time where $t_j < c_j$, or the available energy during that period is e'_j where $e'_j < e_j$. Let us consider Δt and Δe be the time and energy difference when harvesting and computing jobs are mutually exclusive. Thus, the available computation time and energy for mutually exclusive harvesting and computing jobs are $t_j + \Delta t$ and $e'_j + \Delta e$, respectively. When harvesting and computing jobs execute in parallel, the execution time is reduced and the harvested energy is increased. Thus, both Δt and Δe are non-positive numbers. Therefore, job j is not scheduled with mutually exclusive computing and harvesting jobs, which contradicts our assumption. □

Theorem 3: *For a computing jobset to be schedulable, it is necessary that the total energy consumed by computing jobs must be less than equal to total harvested energy by the harvesting jobs in that hyperperiod. Thus, a necessary condition for a computing job set J to be schedulable is- $\sum_{j \in J} e_j \leq \sum_{h \in H} e_h$, where H is the harvesting jobset.*

Proof of Theorem 3. I prove it by contrapositive [183]. Instead of proving the statement above, I prove that if $\sum_{j \in J} e_j > \sum_{h \in H} e_h$, then J is not schedulable.

Let us assume that $\sum_j e_j > \sum_h e_h$. Then, there exists a k such that $\sum_j e_j + k > \sum_h e_h$. Thus, there exists a job j' that fails to compute for $\frac{c_{j'}}{e_{j'}} \times k$ time unit in that hyperperiod. Therefore, j' misses the deadline and J is not schedulable. □

Theorem 4: *For a set τ of preemptive periodic computing tasks with implicit deadline to be schedulable, the necessary condition is –*

$$\sum_{i\in\tau} k_i c_i + \frac{\sum_{i\in\tau} k_i e_i}{\sum_{h\in H'} e_l} c_l \leq \overline{T}$$

Here, $\overline{T}$ is the hyperperiod, k_i is the coefficient that denotes the number of jobs of that task, and $H' \subseteq H$, where H is harvesting taskset.

Proof of Theorem 4. The total computation time of the computing jobs in the hyperperiod $\overline{T}$ is $\sum_{i\in\tau} k_i c_i$ and the execution time of required harvesting jobs is $(\sum_{i\in\tau} k_i e_i)/(\sum_{l\in L'} e_l) \times c_l$. For simplicity, let us denote these by m and n, respectively. Thus, the necessary condition becomes $m + n \leq \overline{T}$.

I prove this by contradiction. Let us assume that $m + n > \overline{T}$. Let us also assume that J is schedulable. $m + n > \overline{T}$ can be expressed as $m + n = \overline{T} + k_1 + k_2$, where $k_1, k_2 \in \mathbb{R}$ or, $(m - k_1) + (n - k_2) = \overline{T}$. Now, there are two cases.

Case 1: When $k_1 > 0$, there exists at least one job j, for which, the available execution time is less than its computation time. Let us assume that only one job j gets execution time $c_j - k_1$. Hence, j is not schedulable. This contradicts our assumption.

Case 2: If $k_2 > 0$, a sufficient number of harvesting tasks can not be executed. Let, h be the harvesting job with executing time k_2 that fails to execute, and e_h is the energy harvested by h. Thus, a job j fails to execute for $c_j - ((e_j/e_h) * k_2))$ time units and misses the deadline. This contradicts our assumption. □

Theorem 5: *For a set of n preemptive periodic computing tasks with implicit deadline scheduled by a static/ fixed priority scheduling algorithm where harvestable energy rate e_l is fixed, and harvesting and computing tasks are mutually exclusive, the worst case response time R_i of a task τ_i is $c_i + \left(\sum_{k=1}^{i-1} \lceil \frac{R_i}{T_k} \rceil (c_k \frac{e_k}{e_l})\right) + \frac{e_i}{e_l}$ and the utilization bound is $\sum_{i=1}^{n} \frac{c_i + (e_i/e_l)}{T_i} \leq n(2^{\frac{1}{n}} - 1)$.*

Proof of Theorem 5. I prove this by construction. Let us assume that tasks are ordered by their decreasing priority, $P(\tau_i) > P(\tau_j)$ when $i < j$. Here, $P(.)$ denotes the priority of the task. The worst-case response time, R_i of a task, τ_i depends on:

- The execution time of τ_i which is c_i.

- Execution time of higher priority tasks that can preempt τ_i and increase its response time. In fixed priority scheduling without energy constraints, this is $\sum_{k=1}^{i-1}\lceil\frac{R_i}{T_l}\rceil c_k$. However, for intermittent systems, the execution time is a combination of computation time and the time to harvest sufficient energy to execute the job. This can be written as $\sum_{k=1}^{i-1}\lceil\frac{R_i}{T_k}\rceil(c_k+\frac{e_k}{el})$.

- The required time to harvest sufficient energy (e_i). When harvestable energy rate e_l is fixed, required time is $\frac{e_i}{e_l}$.

Thus, $R_i = c_i + \big(\sum_{k=1}^{i-1}\lceil\frac{R_i}{T_k}\rceil(c_k+\frac{e_k}{el})\big) + \frac{e_i}{e_l}$.

The utilization bound of a rate monotonic scheduling algorithm for implicit deadline periodic task model is : $U_n = \sum_{i=1}^{n}\frac{c_i}{T_i} \leq n(2^{\frac{1}{n}}-1)$ [184]. As the computation time of each job includes the computation time of the harvesting jobs required to harvest sufficient energy, the utilization bound for intemittent systems, $U_n = \sum_{i=1}^{n}\frac{c_i+(e_i/e_l)}{T_i} \leq n(2^{\frac{1}{n}}-1)$. □

Lemma 1: *Given that the harvestable energy rate is variable, and harvesting and computing tasks are mutually exclusive, a necessary condition for a preemptive periodic task, τ_i with implicit deadline to be schedulable by a fixed priority scheduling algorithm is: $e_l \geq \frac{e_i+\sum_{k=1}^{i-1}\lceil\frac{R_i}{T_k}\rceil e_k}{D_i-c_i-\sum_{k=1}^{i-1}\lceil\frac{R_i}{T_k}\rceil c_k}$*

Proof of Lemma 1. For a task τ_i to be schedulable with fixed priority scheduling, $R_i \leq D_i$ must be true. Using the value of R_i from Theorem 4, I can derive this necessary condition, where e_l is the average harvestable energy rate. □

4.3 Celebi-Offline Scheduling Algorithms

This section describes the *Celebi-Offline* scheduling algorithm for intermittent computing systems that exploits the observations from Section 4.2. It is an offline scheduling algorithms where the pattern of harvestable energy is assumed given. I lift this requirement in the next section where an online version of it is described. *Celebi-Offline* iteratively removes unnecessary harvesting jobs to accommodate computing tasks. *Celebi-Offline* is applicable in scenarios where energy sources are controllable [185], e.g., in offices and warehouses where the lighting and the position and transmission power of RF readers are controllable by using timers or by presetting trajectories.

4.3.1 Scheduling Algorithm

Using the example in Figure 4.2, the four steps of *Celebi-Offline* is described as follows –

• **Initialization.** First, the time slots and harvested energy list are initialized. The example in

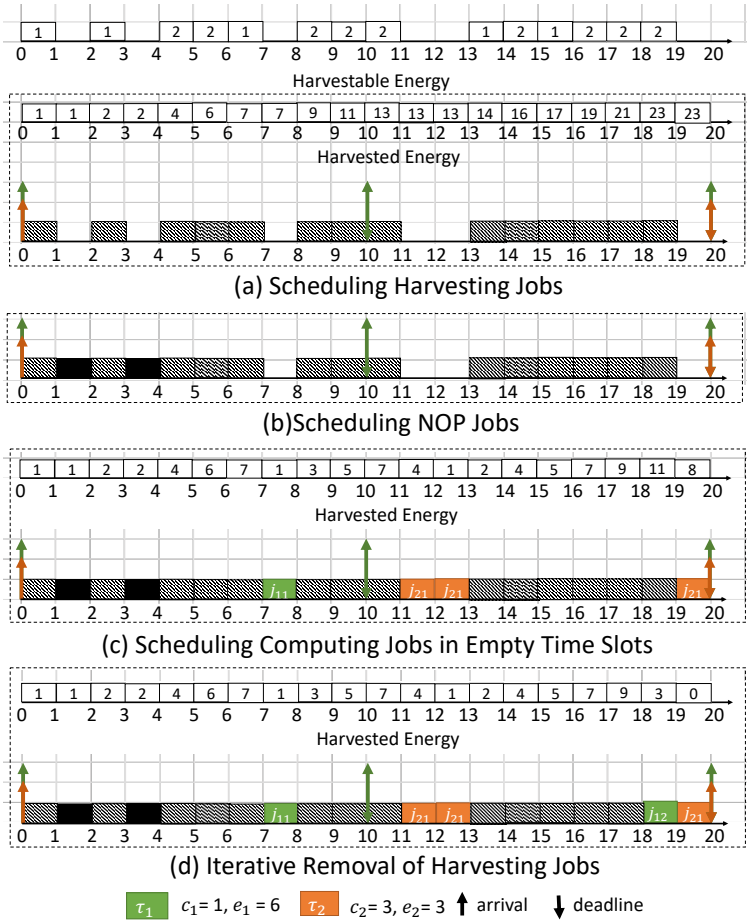


Figure 4.2: Step-by-step execution of *Celebi-Offline* algorithm.

Figure 4.2 has 14 harvesting jobs and three computing jobs: $j_{11} = (0, 10, 1, 6)$, $j_{12} = (10, 20, 1, 6)$ and $j_{21} = (0, 20, 3, 3)$. The hyperperiod $\overline{T}$ is 20.

• **Step 1: Scheduling Harvesting Jobs.** In this step, first, I schedule harvesting jobs at all time-slots where the harvestable energy is present. Then, I update the harvesting energy list by calculating the cumulative sums of the harvestable energy. Figure 4.2(a) shows the schedule and the updated harvested energy list after this step.

• **Step 2: Scheduling *NOP* Jobs.** I generate and schedule the *NOP* jobs from the jobset by checking the empty time slots where harvested energy is smaller that the minimum energy consumption rate of the computing jobs. Scheduling NOP jobs does not update the harvested energy list as no energy is being harvested or consumed at that time slot. Figure 4.2(b) shows the updated schedule with *NOP* jobs.

• **Step 3: Scheduling Computing Jobs in Empty Time Slots.** According to Theorem 1, the remaining time slots after scheduling the harvesting and *NOP* jobs are optimal for scheduling the computing jobs. Using EDF scheduling algorithm and Theorem 4, I determine the computing jobset that is schedulable in the remaining time slots. Note that fixed priority scheduling algorithms such as rate monotonic scheduling algorithm, deadline monotonic scheduling algorithm can also be used instead of EDF having Lemma 1 as a necessary condition. After getting the schedulable jobset I schedule the jobs using EDF and update the harvested energy list by deducting consumed energy from the harvested energy. Figure 4.2(c) shows the resultant schedule after this step.

• **Step 4: Iterative Removing of Harvesting Jobs.** This is the crucial step of the *Celebi-Offline* algorithm. In this step, for each unscheduled job, j (starting from largest deadline), I check the presence of harvesting jobs between the arrival and deadline of job j. If there is no harvesting job, the computing job cannot be scheduled, and this job is added to an unschedulable list. Otherwise, for each harvesting job (starting from latest arrival time), I check if the replacement results in energy scarcity in any of the scheduled jobs. If the job becomes schedulable by replacing the harvesting jobs without resulting in any energy scarcity for already scheduled jobs, I replace the chosen harvesting jobs with the computing job and add it to the scheduled job list. In Figure 4.2(d), the harvesting job at t = 18 gets replaced by j_{12} and all jobs are scheduled.

The computational complexity of *Celebi-Offline* is $O(nD)$, where n is the number of computing jobs and D is the maximum relative deadline of the computing jobs.

4.3.2 Schedulability Analysis

NOP jobs are not assumed for simplicity. I consider two cases: (1) harvesting tasks are periodic, and (2) harvesting tasks are aperiodic.

• **Case 1: Periodic Harvesting Tasks.** For a task to be schedulable in an intermittent system, it has to satisfy two constraints – the *timing constraint* and the *energy constraints*.

The first and the third steps of *Celebi-Offline* schedules the harvesting and computing tasks using EDF. When both tasksets are periodic, the combined periodic taskset is schedulable if they satisfy the *timing constraint*, i.e.,$\Delta(t) \leq t; \forall t > 0$ [186]. Here, $\Delta(t)$ is the processor demand function that calculates the maximum execution time requirement of all jobs which have both their arrival times and their deadlines in a contiguous interval of length t. In an intermittent system, all tasks include

both the harvesting taskset, H and the computing taskset, τ. $\Delta(t)$ for an intermittent system is as follows–

$$\Delta(t) = \sum_{i \in \tau \bigcup H} max\left\{0, 1 + \left\lfloor \frac{t - D_i}{T_i} \right\rfloor\right\} c_i \tag{4.6}$$

Similarly, the *energy constraint* is: $\delta(t) \geq 0$; $\forall t > 0$. Here, $\delta(t)$ is the energy demand function that denotes the difference between the maximum energy requirement of all computing jobs and the maximum energy generation of all harvesting jobs which have both their arrival times and their deadlines in a contiguous interval of length t. $\delta(t)$ for an intermittent system is given by–

$$\delta(t) = \sum_{i \in \tau \bigcup H} max\left\{0, 1 + \left\lfloor \frac{t - D_i}{T_i} \right\rfloor\right\} (c_i e_i z_i) \tag{4.7}$$

Here, τ and H are the set of computing tasks and harvesting tasks, and $z_i = -1$ when $i \in \tau$ and $z_i = 1$ when $i \in H$. If $\delta(t) < 0$, then the energy demand by the computing tasks are greater than the energy harvested by the harvesting tasks, and thus, the taskset is not schedulable.

Let us assume that I removed K harvesting tasks during step 4 of *Celebi-Offline*. Thus, the time constraint is: $\Delta(t) \leq t + t_K$, where t_K is the total execution time of the removed harvesting jobs. The energy constraints is: $\delta(t) \geq e_K$, where e_K is the harvestable energy during K harvesting jobs.

• **Case 2: Aperiodic Harvesting Tasks.** When harvesting tasks are aperiodic, each job have different harvesting tasks between its arrival and the deadline. Thus, I determine whether each job j_{ik} is schedulable on arrival. A job j_{ik} is schedulable only if

$$d_{ik} - a_{ik} - \sum_{m \in H_1} c_m - \sum_{n \in J_1} (c_n - f_n) \geq c_{ik} \tag{4.8}$$

and

$$E_{a_{ik}} + \sum_{m \in H_1} (e_m \times c_m) - \sum_{n \in J_1} e_n \times (c_n - f_n) \geq (e_{ik} \times c_{ik}) \tag{4.9}$$

Here, H_1 and J_1 are the harvesting and the computing jobsets which have higher priorities than job j_{ik}, and are scheduled between a_{ik} and d_{ik}. f_n is the scheduled execution time of job j_n before a_{ik}. Equation 4.8 is the timing constraints which states that the remaining time after the execution of the high priority jobs between the deadline and the arrival time is greater than or equal to the execution time of j_{ik}. Similarly, Equation 4.9 is the energy constraints which denotes that the

available energy after the execution of the high priority jobs between the deadline and the arrival time is greater than or equal to the energy demand of job j_{ik}.

4.4 Celebi-Online Scheduling Algorithm

Many IoT tasks demand for an online scheduling approach where the decision needs to be made on the go. In such algorithms, the harvestable energy is not known a priori. I propose an online, threshold-based scheduling algorithm for intermittent systems, named *Celebi-Online* scheduling algorithm. When the harvestable energy is below a threshold this algorithm executes computing jobs early. In this algorithm, the harvestable energy at the beginning of each time-slot is either predicted or measured using a sensor, and is assumed to remain unchanged during that time slot. Section 4.7 measures the harvestable energy with a sensor by measuring the voltage of the solar panel and the capacitor.

4.4.1 Scheduling Algorithm

The *Celebi-Online* scheduling algorithm has three steps. Using the example in Figure 4.3, the steps of *Celebi-Online* are described as follows –

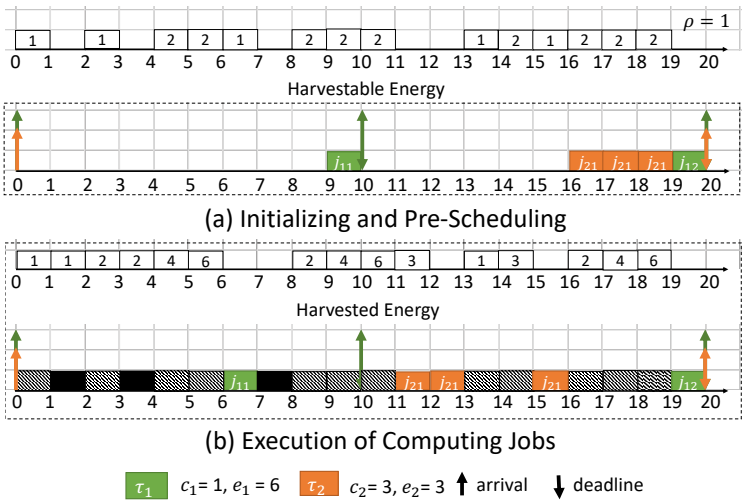


Figure 4.3: Step-by-step execution of *Celebi-Online* algorithm.

• **Step 1: Initializing and Pre-Scheduling.** First, I schedule the computing jobs using As Late As Possible (ALAP) scheduling algorithm [187] based on the deadline and the execution time. As late as possible scheduling algorithm starts the execution of a job at the latest time as long as it meets the deadline. This presents the intermittent system with the opportunity to harvest as much energy as possible before executing the tasks. All the unscheduled jobs after applying ALAP

algorithm are added to an unscheduled list for consideration at a later step. Figure 4.3(a) shows the schedule after this step.

• **Step 2: Execution of Computing Jobs.** If a computing job is scheduled at the current time by the previous step and the harvested energy is sufficient, I execute the scheduled computing job. If not, I add this job to the unscheduled list for consideration. I then check the unscheduled list to see if jobs can be executed at current time, where the remaining time in the schedule is enough to meet its deadline. If so, then I execute that job; otherwise, I execute a harvesting job (if harvestable energy is present) or a *NOP* job (if harvestable energy is zero).

If no jobs are scheduled at the current time and the harvestable energy is greater than a threshold, I harvest energy. If the predicted harvestable energy is smaller than or equal to the threshold, I check if any of the scheduled jobs has the opportunity to be executed early. If so, I execute it; otherwise, I harvest energy. In Figure 4.3(b), I execute j_{21} earlier than scheduled by ALAP.

• **Step 3: Adapting Threshold.** The threshold ρ is updated after each hyperperiod if either of the two conditions are true: (1) the remaining harvested energy after the hyperperiod is greater than the summation of (a) the maximum energy consumed by a computing job that misses the deadline, and (b) the minimum harvestable energy, and (2) the total execution time of NOP jobs are greater than the summation of the maximum execution time a computing job that misses the deadline and the time required by a harvesting job with lowest energy generation rate to harvest sufficient energy for executing that computing job. Condition (1) implies that I have wasted time to harvest more energy than required. Condition (2) refers that the system is not harvesting enough energy and creating energy scarcity. The updated threshold equals to the minimum harvestable energy during the previous hyperperiod because a lower threshold might result in condition (1), whereas a higher threshold might result in energy scarcity.

4.4.2 Computational Complexity

The computational complexity of ALAP is $O(1)$ as it can be prescheduled with known periodic tasks. The most time consuming computation for *Celebi-Online* is to determine the schedulable job when the harvestable energy is below the threshold, given all n jobs are available. Thus, the computational complexity of *Celebi-Online* is $O(n)$.

4.4.3 Schedulability Analysis

For a job j to be schedulable with *Celebi-Online*, following is a necessary condition:

$$c_j + c_{H(j)} + \sum_{k \in HP(j)} (c_k + c_{H(k)}) \leq d_j \tag{4.10}$$

Here, $HP(j)$ are the higher priority jobs than j in the jobset J. $c_{H(j)}$ is the total execution time of the harvesting tasks that harvest at least $(c_j \times e_j)$ units of energy. In *Celebi-Online*, the condition for a job k to be of higher priority than job j is: $a_j \leq (d_k - e_k) \leq d_j$.

4.5 Simulation-based Evaluation on Synthetic Dataset

This section compares the performance of *Celebi* scheduling algorithms against baseline algorithms using synthetic taskset and harvestable energy pattern.

4.5.1 Baseline Algorithms and Performance Metric.

I evaluate *Celebi* by comparing them with an optimal scheduler and three online baseline scheduling algorithms – *earliest deadline first (EDF)* [184], *rate monotonic (RM)* [184] and *as late as possible (ALAP)* [187]. Table 4.1 shows the worst case computational complexity of these scheduling algorithms, where n is the number of computing jobs.

Optimal	**Celebi-Offline**	**Celebi-Online**	**EDF**	**RM**	**ALAP**
O(e^n)	O(nD)	O(n)	O(n)	O(1)	O(1)

Table 4.1: Worst case computational complexity.

I use the ratio of number of jobs scheduled by the target scheduling algorithm and the number of jobs scheduled by the optimal scheduling algorithm as the performance metric.

4.5.2 Synthetic Dataset

The synthetic dataset contains 1,000 randomly generated computing tasksets. I provide the maximum allowed period, the minimum number of tasks, and the CPU utilization, i.e., the summation of the ratio of the execution time and the period of all computing tasks, as the inputs to the random task generator, and it generates tasks with random execution time, energy consumption rate, and periods. The periods are chosen randomly from a predefined range of 1s to 60s, following existing literature on intermittent computing systems [188, 189, 190, 191, 33, 8, 26, 21, 23]. I choose the execution time randomly between 1s and the period. The period is considered as the upper bound as

the execution time can not be greater than the period in an implicit deadline system. The random selection of execution time depends on the number of tasks (2 to 10) and the CPU utilization (in multiples of 10). To select the energy consumption rate, I randomly choose one of the three levels of energy consumption rates. The first two levels correspond to the two levels of power consumption of an MSP430FR5994 microcontroller in its active mode. Additional sensors consume more energy to operate; hence, I add the third level to support the activation of these sensors. For each evaluation, I generate 10 iterations of 1,000 tasksets and report the average performance over these iterations. To generate synthetic energy traces, I randomly generate four levels of harvestable energy. I use these synthetic tasksets and harvestable energy traces for evaluation in Section 4.5.3, 4.5.4, and 4.5.5.

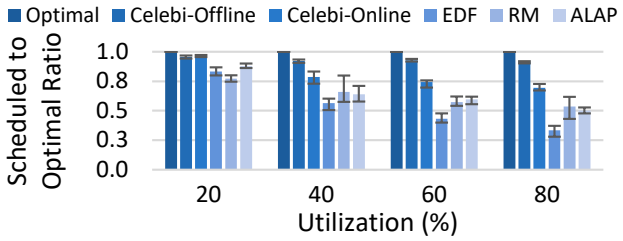


Figure 4.4: Performance of scheduling algorithms for different CPU utilization. The optimal scheduler schedules 40%–98% of jobs in the jobset.

4.5.3 Effect of CPU Utilization

Figure 4.4 shows the performance of the schedulers for various CPU utilization. The CPU utilization is the summation of the ratio of the execution time and the period of all computing tasks, $\sum_{i=1}^{N}(c_i/T_i)$, for $\tau_1, \tau_2, ..., \tau_N$. To demonstrate the effect of CPU utilization on different tasksets, I vary the number of tasks to find combinations of periods and execution times that have the same CPU utilization. Though the performance of *Celebi-Offline* is unaffected by the variation of CPU utilization, online algorithms suffer when utilization is high. The inability to rectify greedy/ suboptimal decisions in online algorithms contributes to this by executing jobs which later fails to meet the deadline due to lack of energy. *Celebi-Online* schedules 70% of the jobs scheduled by the optimal scheduler for 80% CPU utilization, whereas EDF, RM, and ALAP schedule 33%, 54% and 50% jobs, respectively. With further experiment, I observe that at very low CPU utilization ($< 10\%$) all schedulers behave close to the optimal scheduler. EDF schedules jobs with higher time and energy demands more frequently which results in energy scarcity for the remaining jobs and decreases performance. In summary, *Celebi* scheduling algorithms perform better that the baseline

algorithms even with high CPU utilization.

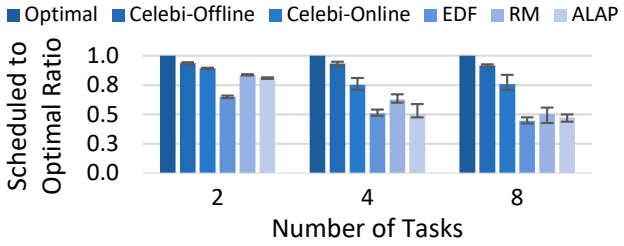


Figure 4.5: Performance of scheduling algorithms for different number of tasks.

4.5.4 Effect of Taskset Size

Figure 4.5 shows the performance of the schedulers over different number of tasks where CPU utilization is 50%. I randomly choose different periods and calculate the required execution time within the range to generate task-sets having a fixed number of tasks and a fixed CPU utilization. The performance of the schedulers drop with increasing number of tasks. Though *Celebi-Offline* experiences 2% performance drop, the online schedulers incur 14% – 34% performance drop. Higher number of jobs results in more choices during selection of jobs. Among the online schedulers, *Celebi-Online* shows higher resistance to increasing number of tasks with a performance drop of 14% because it provides more charging time than the RM and the EDF and has lesser NOP tasks than the ALAP scheduling algorithm.

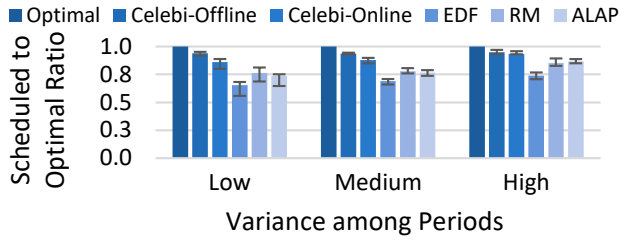


Figure 4.6: Performance of scheduling algorithms for different variance among the periods of the taskset.

4.5.5 Effect of Different Periods

Figure 4.6 shows the behaviour of the schedulers for different task periods. I consider tasksets with three different variance levels among task periods. At low variance, a taskset has tasks with same periods. At high variance, periods on all the tasks are significantly different. At medium variance, a taskset has tasks with same periods as well as tasks with significantly different periods. At high variance, *Celebi-Online*, EDF, RM and ALAP incur 7.5%, 8%, 9% and 13% performance drop,

respectively. In this scenario, RM and EDF achieve relatively better performance by choosing jobs with short periods over the longer periods. This results in higher number of scheduled jobs but jobs with the longer period never get scheduled. When the variance is low, *Celebi-Online* struggles in step 1 to choose between jobs with the same deadline, which decreases the performance. The same reason also contributes to the lower performance of ALAP, RM and EDF scheduling algorithms. Therefore, in systems where different tasks have different periods, e.g., small period for timer and large period for transmission, *Celebi-Online* performs relatively better than the baseline online schedulers.

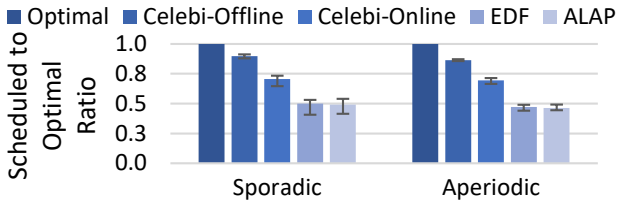


Figure 4.7: Performance of scheduling algorithms for non-periodic tasksets.

Even though our algorithms are intended for periodic task models, they can schedule non-periodic (sporadic and aperiodic) tasks, if the arrival times of the jobs are known before-hand, in addition to the execution time, relative deadline, and energy consumption rate of these non-periodic tasks. Figure 4.7 compares the same scheduling algorithms as in Figure 4.6, except for RM which is impractical for aperiodic tasks. Figure 4.7 shows that *Celebi-Offline* and *Celebi-Online* successfully schedules ≈39% and ≈23% more jobs, respectively, compared to EDF and ALAP.

4.6 Simulation-Based Evaluation on Trace-based Harvestable Energy

This section evaluates the performance of *Celebi* with two types of energy sources (i.e., solar and RF) in two types of scenarios: dynamic and static. I use the synthetic taskset described in Section 4.5.2 along with real-world energy harvesting traces for this evaluation.

4.6.1 Energy-Trace Collection

Solar Energy Trace. I collect solar energy trace in two scenarios – static and dynamic. In the static scenario, the harvestable energy is nearly constant. I collect solar energy traces during cloud-free sunny days to represent the static scenarios (Figure 4.10 (left)). In the dynamic scenario, the harvestable energy varies over time. This solar energy trace is collected from the side-walk of a busy street to represent the dynamic scenarios (Figure 4.10 (right)) where pedestrians and passing vehicles momentarily overshadow the sunlight. To collect the energy trace, I use a Raspberry Pi that

measures the voltage across the solar panel connected to a load resistor. As the Raspberry Pi is not equipped with an ADC, I use an Arduino Uno to collect the voltage and send it to the Raspberry Pi using UART. Figure 4.8 shows the energy trace collection setup.

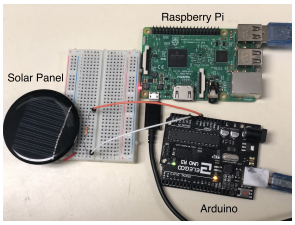
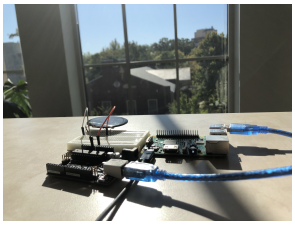
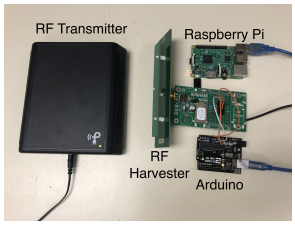


Figure 4.8: Solar energy trace collection setup.





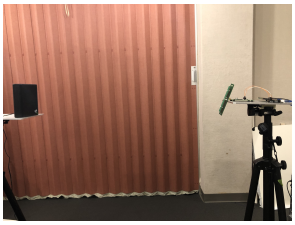
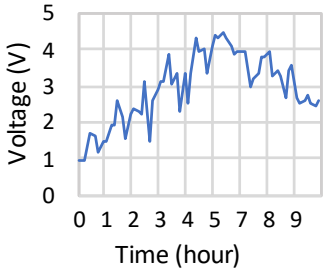


Figure 4.9: RF energy trace collection setup.



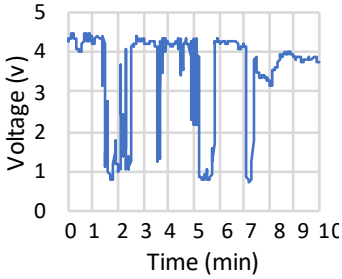
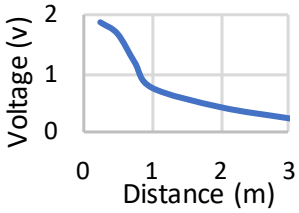
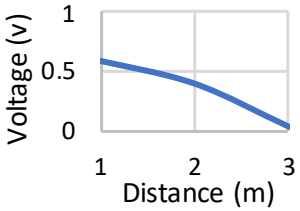


Figure 4.10: Solar energy trace collected beside a window (left) and on the sidewalk (right).





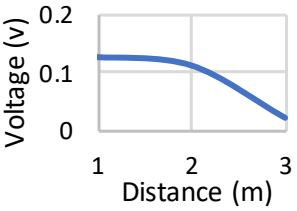


Figure 4.11: Analog voltage level corresponding to the harvested power at different transmitter to RF harvester distance in line of sight (left), non line of sight with wooden obstacle (middle), and non line of sight with human obstacle (right).

RF Energy Trace. To collect the RF energy trace, I use a 915 MHz harvester-transmitter pair [192, 193] (Figure 4.9). I measure the analog voltage level corresponding to the harvested power that is provided by pin D_{out} of the harvester at different transmitter-to-harvester distances using an Arduino Uno and Raspberry Pi. Figure 4.11 shows the analog voltage level for different distances and scenarios, i.e., line-of-sight and non-line-of-sight. For the static scenario, the harvester and the transmitter are in the line-of-sight at 1m distance.

To simulate a real-life dynamic scenario, I collect location trajectory of a mobile robot from [194].

For each position of the robot, I estimate the RF energy it would have harvested if it carried a RF harvester. The estimation process maps the distance to RF energy, which is measured in our lab by varying the transmitter-to-receiver distance of the RF harvester.

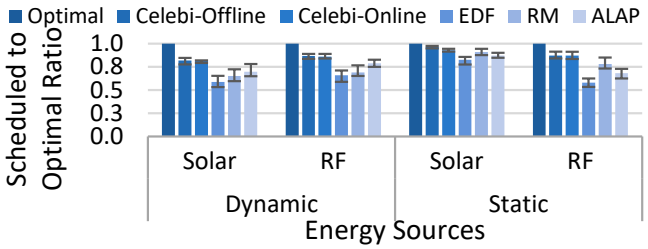


Figure 4.12: Performance of scheduling algorithms over various energy sources for taskset with random execution time and energy consumption.

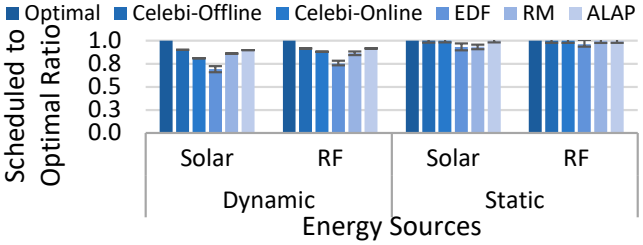


Figure 4.13: Performance of scheduling algorithms over various energy sources for taskset with same execution time and energy consumption.

4.6.2 Effect of Different Energy Sources

In the dynamic scenario of Figure 4.12, both *Celebi-Offline* and *Celebi-Online* perform better than the rest due to their capability of handling the variation in the harvestable energy. Despite having less harvestable energy than the solar, the RF harvester in the dynamic scenario is better for the scheduler as the transmitter-to-receiver distance changes linearly. In the static scenario of Figure 4.12, *Celebi-Online* performs slightly better than RM by executing jobs that have larger periods but smaller execution time or smaller energy consumption rate. Such jobs get interrupted by jobs with smaller periods in RM and thus, they miss their deadline. In ALAP, more jobs misses deadline as unlike *Celebi-Online*, it does not reconsider the unschedulable jobs.

Figure 4.13 evaluates the performance of the scheduling algorithms on the trace-based harvestable energy and synthetic datasets. In this experiment, all tasks have the same execution time and energy consumption rate to understand the effect of the energy sources without the influence of the taskset. I choose the average execution time and the average energy consumption of the tasks in the synthetic

dataset for these taskset generation. In the dynamic scenarios of Figure 4.13, *Celebi-Online* and ALAP performs similarly due to the lack of opportunity to execute a scheduled job early. Both RM and EDF suffers due to executing tasks too early and choosing non-optimal harvesting tasks. In the static scenario of Figure 4.13, *Celebi-Offline*, *Celebi-Online* and ALAP perform similar to the optimal scheduler as both the demand of the tasks and the harvestable energy are static, and therefore, executing any task is optimal.

4.7 Real System Evaluation

This section demonstrates the performance of *Celebi* in uncontrolled real-life scenarios. Unlike Section 4.5 and Section 4.6, this evaluation is performed using real tasksets executing on an MSP430 microcontroller that is deployed in the wild.

4.7.1 Hardware Implementation

To implement a real system, I use TI MSP430FR5994 [175] MCU (in Figure 4.14 and 4.15) having 256KB FRAM, 8KB SRAM, direct memory access (DMA), and an operating voltage range of 1.8V to 3.6V at 8MHz CPU clock speed. I use a solar panel with polycrystaline solar cells [195] which outputs at most 5V at 40mA. As the operating voltage of the MCU is below 3.6V, I use a step-up regulator [196] that ensures that the output voltage is always at 3.3V. As the energy storage, I use a 680mF super capacitor. To monitor the harvested and harvestable energy, I utilize the analog to digital converter (ADC) in MSP430 and a 1MΩ capacitor. I use this high capacitance to reduce energy flow in the measurement circuit which draws energy from the capacitor.

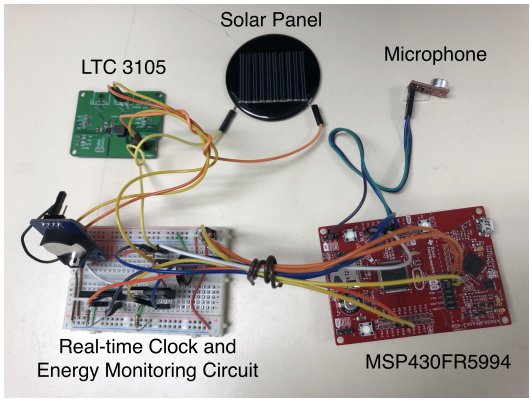


Figure 4.14: System setup.

For sensing, I use an electret microphone [197] and the on-board temperature sensor of the MSP430FR5994 launchpad. I read the audio sensor at 8KHz using the ADC, perform FFT, and

write the data to the FRAM using the low energy accelerator (LEA) and direct memory access (DMA) without involving the CPU. Like [12, 4], I use a real-time clock (DS3132 [164]) connected via I2C for timekeeping. I use this clock only during the power up to sync and maintain the internal clocks of the MCU. This clock is replaceable with an SRAM or capacitor-based timekeeping system during power outages [1, 2]. As the capacitor can charge by draining energy from the battery of the real-time clock, I implement rectifiers using an N-channel MOSFET and a P-channel MOSFET to isolate the clock signal (SCA) and data signal (SDA) when the real-time clock is not being used. Note that the worst-case energy consumption rate of a task can be estimated [198, 199] or measured. I use TI Energytrace++ [200] to estimate energy requirements.

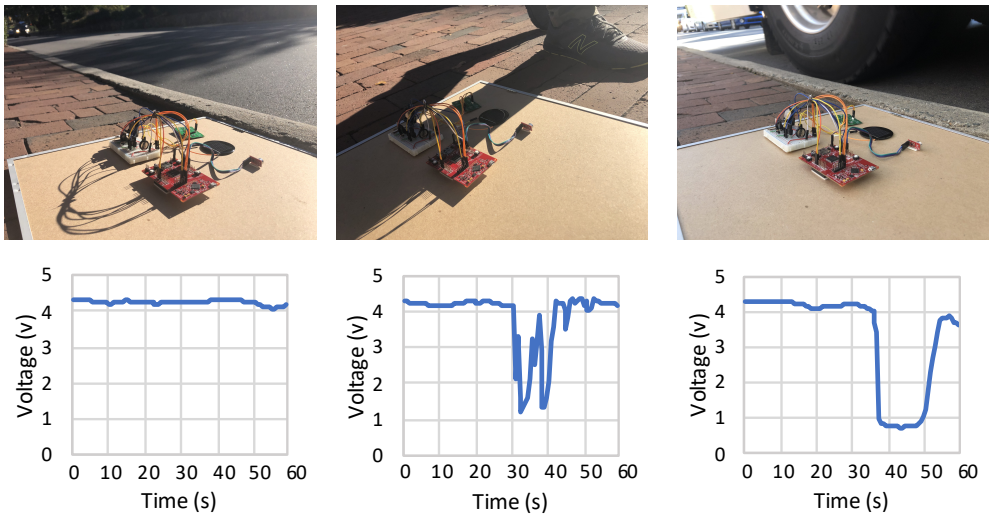


Figure 4.15: Real System Evaluation Setup.

Task Type	Task Name	Execution Time	Period
Sense and Compute	Temperature Anomaly Detector	0.36s	60s
	Acoustic Event Classifier (DNN)	9.72s	60s
Compute	RSA Encryptor	4.68s	40s
	Bit Counter	2.16s	30s

Table 4.2: Description of the taskset.

4.7.2 Software Implementation

I implement a taskset consisting of four tasks which are described in Table 4.2. The temperature anomaly detector reads data from the on-board temperature sensor and calculates the local outlier factor (LOF). If the LOF >> 1, the data sample is an outlier. This task is constructed with five fragments implemented using an open-source task-based intermittent computing framework,

ALPACA [13].

The second sensing task is an acoustic event detector using a scaled-down deep neural network (DNN) that runs on an MCU. This event detector reads audio signal from the microphone and executes a 5-layer DNN having 3 convolution layers and 2 fully connected layers. I use max pool layers and rectified linear unit (RELU) activation function. Due to the high computational demand of a DNN, this is the highest energy and time consuming task in our taskset. I implement this using an open-source framework for executing DNN in intermittent systems, named SONIC [8]. SONIC is a special framework for DNN built on top of ALPACA.

The RSA encrypts a fixed, in-memory input string of an arbitrary size using a fixed encryption key. I use a 6,000 character string and 64-bit key in our experiment. The bit counter uses seven different algorithms to count the set bits in a random string and compares their results to ensure correctness [201]. I repeat each operation 10,000 times. I use the open-source intermittent execution framework ALPACA [13] to implement these tasks.

I implement *Celebi-Online* scheduling algorithm and a baseline online ALAP scheduling algorithm to schedule these four tasks. The execution time overhead of *Celebi-Online* and ALAP are 12ms and 1ms over a hyper-period of 120s, respectively. ALAP incurs less overhead as it does not update at runtime and requires only a queue lookup operation. Considering the task execution time, the overhead of *Celebi-Online* is 30x-810x smaller. Similar to previous work [12] where multiple tasks execute in an intermittent uniprocessor, these tasks are preemptive at fragment boundaries. This means that a task is only preempted at the end of a fragment. To implement this, I do not execute a fragment if the remaining time-slot is insufficient. Fragments being more than 20 times smaller than the time-slots, there is effectively no utilization loss.

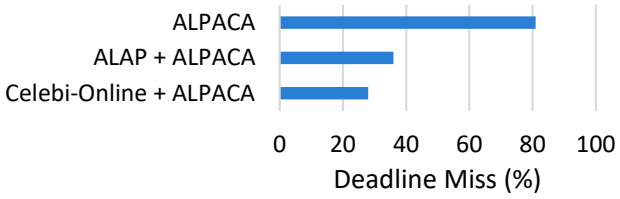


Figure 4.16: Comparison of the performance of *Celebi-Online* scheduler with a system with ALAP scheduler and a system without any scheduler.

4.7.3 Experimental Results

Figure 4.16 observes that ALPACA misses the deadline of 81% of the jobs as it does not check the deadline and keeps executing jobs from a queue whenever energy is sufficient. Although this is a non-real-time system, I report the results to demonstrate the necessity of a real-time scheduler for batteryless systems. The integration of as late as possible (ALAP) scheduler to ALPACA decreases the deadline miss to 36% by scheduling jobs and allowing the maximum time to harvest energy. Replacing ALAP scheduler with *Celebi-Online* scheduler decreases the deadline miss even more, resulting in a deadline miss ratio 28%. The adaptability of *Celebi-Online* to the randomness of harvestable energy contributes to this additional performance boost.

4.8 Discussion

This section discusses the limitations of *Celebi* along with possible solutions to address them.

The Case of Harvesters Directly Powering the Load. The proposed scheduling algorithms are not designed for intermittent systems that connect the harvester directly to the load without using any energy storage in between. These systems behaves like a persistently-powered system as long as the harvestable energy is abundant. Hence, I recommend using existing real-time scheduling algorithms for scheduling tasks on them. To incorporate intermittence into the scheduling framework, these systems can model the power-down phases as high-priority tasks prior to applying the scheduling algorithms.

The Case of Non-Periodic Tasks. Section 4.5.5 demonstrates that the proposed algorithms are applicable to non-periodic tasksets with known job arrival times. When the arrival times are not known apiori, these algorithms are not generally applicable to non-periodic tasks. However, *Celebi-Online*, can be extended to support a sporadic taskset. The scheduler, in this case, will schedule anticipated sporadic jobs based on the minimum period between consecutive sporadic jobs, and will delay computing jobs by adding more harvesting jobs or NOP jobs until the sporadic job actually arrives. It will also have to discard a scheduled sporadic job if the sporadic job eventually does not arrive before the release time of the next anticipated sporadic job.

The Case of Abundant Harvestable Energy. An energy harvesting system that has harvestable energy (supply) >= required energy (demand) behaves like a persistently-powered system because there is no intermittence in power supply. These systems are out of scope of this paper.

The Case of Highly Varying Harvestable Energy. There may exist energy harvesting systems where the relationship between the harvestable energy (supply) and required energy (demand) is unknown and may change at runtime, i.e., sometimes the supply >= demand, and sometimes supply < demand. To schedule real-time tasks on such systems, the runtime system should isolate these two cases and apply the proposed scheduling algorithms (*Celebi-Offline* or *Celebi-Online*) only when supply < demand, and use an existing real-time scheduling algorithm (e.g., ALAP) when supply >= demand. This is because, although *Celebi* would still be able to execute the real-time tasks correctly, I acknowledge that there is a loss of opportunity to harvest energy when energy is abundant (supply > demand) but our algorithm schedules a computing job due to the mutual exclusion of harvesting and computing jobs. The loss, however, is limited by the size of the capacitor. For instance, the potential loss of harvestable energy due to the mutual exclusion of harvesting and computing jobs is 12.6mW–17mW for the systems presented in Section 4.7 which is the difference between the consumption rate and the maximum rate of harvestable energy.

4.9 Summary

This chapter studies the real-time scheduling problem for intermittent systems that takes into account the time and energy demands of the tasks as well as the harvestable energy in the environment. I propose *Celebi*, an offline and an online scheduling algorithm, that schedule both harvesting and computing jobs to increase the number of jobs that meet the deadline. *Celebi-Offline* performs 92% similar to an optimal scheduler and *Celebi-Online* schedules 8%-22% more jobs than traditional scheduling algorithms. In real system evaluation, *Celebi-Online* scheduling algorithm schedules 63% more tasks than a non-real-time system and 8% more jobs than a baseline scheduling algorithm.

CHAPTER 5

Imprecise Deep Neural Network Inference in Deadline Aware Intermittent Systems

The sporadic nature of harvested energy, resource constraints of the embedded platform, and the computational demand of deep neural networks (DNNs)[1] pose a unique and challenging real-time scheduling problem for which no solutions have been proposed the literature. Existing works on time-aware batteryless computing systems primarily focus on maintaining a reliable system clock across power-failure [3, 1, 2] and removing state data [4, 48, 49, 12]. However, none of these works aims to finish a task within a deadline. Recent works on batteryless systems [178, 8, 33] have proposed frameworks and runtime to execute non-real-time DNN tasks on intermittently-powered systems. The real-time community has proposed techniques [126, 127, 203, 204, 205] for deadline-aware execution of DNNs, primarily on GPU and server-grade machines. However, due to these techniques' significant computation overhead, they are not suitable for batterless computing systems. Moreover, none of these works takes the sporadic nature of energy into account. Despite these commendable efforts, there is a gap in the existing literature that none has considered all three dimensions, i.e., intermittence of harvested energy, the variable utility of DNN inference, and real-time schedulability; and merely combining existing solutions do not entirely solve the problem. Though a taskset can be scheduled when the power is persistent, it may suffer from time scarcity when power is intermittent. To schedule the same taskset in an intermittently powered system within the deadlines, I must use approximation or imprecise computing, where partial execution with minimal error is acceptable.

I illustrate this using an example in Figure 5.1. The example consider two jobs, J_1 and J_2, released at time 0 and 20, respectively. Their relative deadline is 34, and the execution time is 28.

[1]The DNN, by definition, refers to neural networks having more than one hidden layers [66, 67, 68, 69]. Thus, a wide variety of networks qualify as a DNN in the existing literature. DNNs considered in this paper have up to 10^5 neurons and weights combined. They fit into 256KB memory of an MCU; have convolutional, ReLU, pooling, and fully-connected structures as regular DNNs; and perform on-device inference [8, 33, 202]

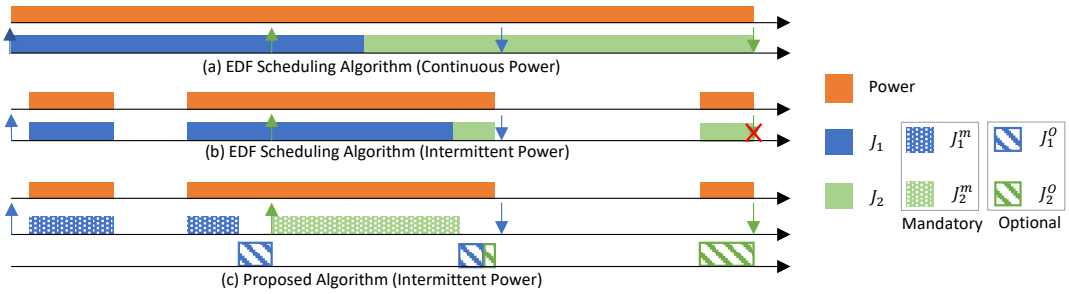


Figure 5.1: (a) With constant power both tasks meet their deadline. (b) With intermittent power, job J_2 misses its deadline. (c) When tasks are imprecise, the mandatory parts of both jobs complete on time, and some optional part of J_1 gets done as well.

The intermittency of energy generally has no consistent pattern in the duration of or in the gaps between ON/OFF phases. Figure 5.1(a) shows that when the power is uninterrupted, both jobs meet their deadlines under the earliest deadline first (EDF) scheduling. When power is intermittent (Figure 5.1(b)), task J_2 misses its deadline. Figure 5.1(c) illustrates our proposed approach which partitions a job into mandatory and optional portions and ensures that the mandatory portion (which is required to achieve a desirable inference accuracy) of each job finishes on time. And if there is extra time, our proposed approach schedules some optional jobs that may increase the accuracy further.

The primary research question for scheduling tasks within the deadlines in an intermittent system is – *How to imprecisely execute deep neural network inference without sacrificing significant accuracy to impose timeliness in intermittent systems?* To answer this, I present Zygarde — which is the first system that enables deadline-aware imprecise execution of DNNs on an intermittently-powered system. The design of Zygarde is motivated by two observations. First, DNN is a layered architecture, and deeper layers of a DNN extract a more detailed and fine-grained representation of the input data. Second, most real-world application scenarios contain both simple and complex data. As a result, they are imprecise, i.e., error-tolerant, and require partial execution to achieve the desired outcome. For example, in deep neural network inference, all input data often does not require inferring all the layers to classify correctly. Zygarde exploits these observations and proposes an *imprecise computing*-based [114, 206] online scheduling algorithm, which considers both the intermittence of energy and the accuracy-execution trade-off of a DNN.

The main contribution of this chapter is a deadline-aware runtime framework for DNNs executing on intermittently-powered systems, for which, runtime adaptation of a DNN is necessary, on top of

compile-time compression [8, 34, 35, 36, 37, 38, 39]. Compile-time compression alone is not sufficient when the remaining deadline is inadequate for a full execution of the DNN, but long enough to compute the inference result from the partial execution of the DNN. This runtime adaptation process requires (1) modeling the energy harvesting pattern using a single factor(η), which helps determine the system how much computation is possible in the near future, and (2) determining the dynamic imprecise boundary at the runtime with minimal overhead, which decides where the mandatory portion ends and the optional portion begins. Zygarde also includes a specialized offline training of the DNNs to minimize the loss of accuracy due to partial execution.

Zygarde complements prior work on batteryless systems such as time-keeping [1, 2] and execution of non-real-time tasks [178, 8, 10, 207, 11, 41, 13, 45, 42]. In contrast to all previous works, Zygarde's contribution is at the framework, modeling, and algorithmic level, while its implementation relies upon existing open-source frameworks and APIs [8, 13] that handle the lower-level aspects of an intermittent system.

I implement Zygarde on a TI MSP430FR5994 microcontroller and evaluate its performance using four standard datasets (MNIST [208], ESC-10 [209], CIFAR-100 [210], and Visual Wake Works [211]) as well as in six real-world acoustic event detection and visual sensing experiments. Zygarde achieves 5%–26% reduction in execution time using early termination. By using the layer-aware loss for early termination, it also increases the inference accuracy by upto 21% than the state-of-the-art solutions that use cross-entropy loss [212] and contrastive loss [213]. Moreover, it schedules 9%–34% more jobs with upto 30% higher inference accuracy than the earliest deadline first (EDF) scheduling algorithm. Furthermore, it gains up to 28% higher inference accuracy than the imprecise variant of EDF.

5.1 Zygarde System Design

This section describes the processing and the scheduling units of the scheduling framework (Section 3.3) for imprecise deep neural network inference along with an example to illustrate its execution. Figure 5.2 shows the components of processing and scheduling units as well as their interaction with each other.

5.1.1 Processing Unit

The processing unit of Zygarde contains two components – an agile DNN model and semi-supervised k-means classifiers.

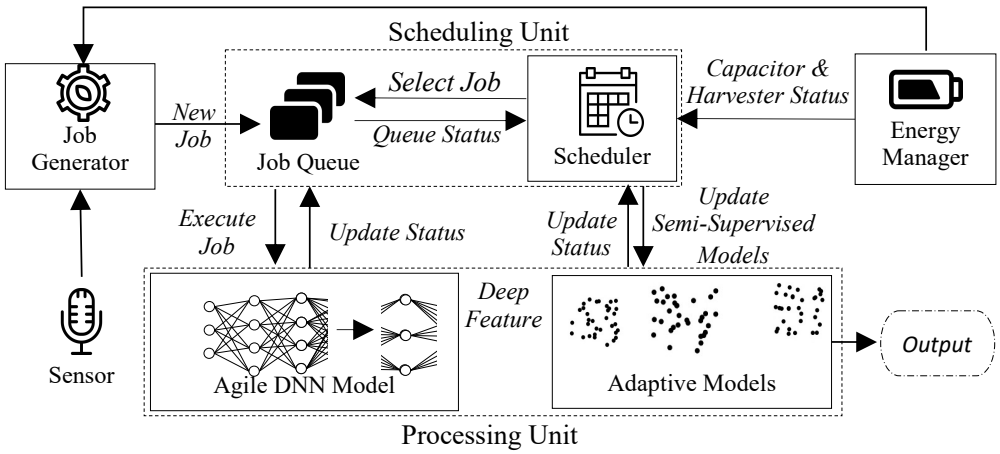


Figure 5.2: Zygarde System Architecture.

Agile DNN Model. The *Agile DNN Model* is a pre-trained deep neural network that converts data samples into feature representations [214]. The network is trained offline, on a high-end server, using labeled training data. I use rank-decomposition [85] and separation [34] to compress and fit the network into the limited memory of a microcontroller. Based on the quality of the input data sample, Zygarde may terminate the execution of the DNN early at runtime, and hence, I call it an *agile* DNN Model. I note that an agile DNN is a special type of anytime DNN [99, 109, 110, 111, 112, 113] — where the depth of a neural network is dynamically adjusted during inference. However, unlike anytime networks, where the output of a hidden layer is fed to a secondary shallow neural network for classification, agile DNN employs a cluster-based classifier to replace expensive matrix multiplication operations with 4X less costly additions/subtractions [215, 216]. In our implementation of agile DNN, this design saves 27,750 execution cycles per inference, when compared to anytime neural networks.

Semi-Supervised k-Means Classifiers. The feature representation obtained from the execution of the Agile DNN is classified by a semi-supervised k-means clustering algorithm [217]. The clustering algorithm uses the L1 distance between two feature vectors [218]. For layers (e.g., the convolution layers) that produce two or more dimensional features, they are flattened or vectorized prior to computing the L1 norm. Since the execution of an Agile DNN may terminate at any layer depending on the input data, Zygarde maintains a separate k-means classifier for each layer of the Agile DNN. These k-means classifiers are trained offline on a server machine. However, to enable online learning, these classifiers are updated at runtime using a model adaptation process described in

Section 5.3.3. The motivation behind the k-mean based approach is to reduce the execution time and energy consumption by avoiding multiplications which is over 4× more expensive than additions and subtractions.

5.1.2 Scheduling Unit

A job queue and a scheduler together works as the scheduling unit. Zygarde implements an online, dynamic priority, real-time scheduler that considers not only the timing aspects but also the expected inference accuracy of a job and the energy harvesting status of the system. It dynamically partitions an executing job into mandatory and optional portions based on the early termination of an agile DNN and prioritizes the execution of its mandatory portion to ensure both timeliness and accuracy under the constraints of intermittently available energy. Note that, the beginning portion of all jobs are mandatory and whether the next unit is mandatory or optional is determined during the execution of the current unit. An illustration of the scheduler is described next.

Table 5.1: Description of the workload.

Job	Total Layers	Mandatory	Optional	Release Time	Deadline
$J_{1,1}$	4	1	3	t_1	t_7
$J_{1,2}$	4	2	2	t_3	t_9

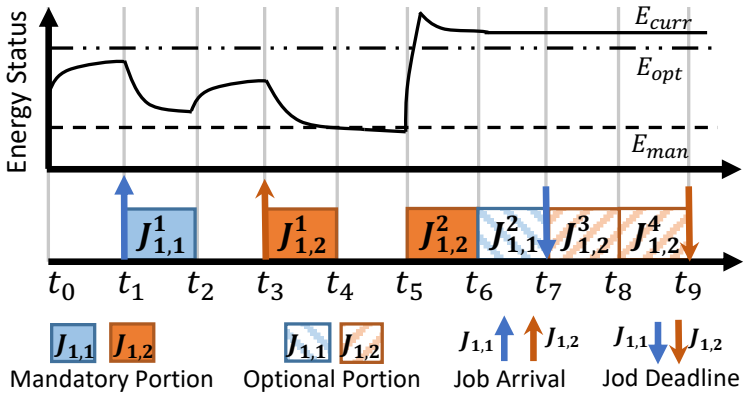


Figure 5.3: Execution schedule of the workload.

5.1.3 Example Execution

Table 5.1 defines two jobs $J_{1,1}$ and $J_{1,2}$ from the same task, τ_1. Figure 5.3 demonstrates the execution of the jobs along with the status of harvested energy over a timeline. E_{curr} refers to the current energy, and E_{opt} and E_{man} refer to two thresholds that determines whether the optional and the mandatory parts of a job should be executed, respectively. $J^k_{i,j}$ refers to the k^{th} partition of

Table 5.2: Explanation of the schedule in Figure 5.3.

Time	Actions with Reasoning
t_0	There is no job in the system.
t_1	$J^1_{1,1}$ (the only job) gets scheduled.
t_2	Since $E_{curr} < E_{opt}$, optional $J^2_{1,1}$ is not scheduled.
t_3	System prioritized $J^1_{1,2}$ over $J^2_{1,1}$ (See: Section 5.4).
t_4	Since $E_{curr} < E_{man}$ no job is scheduled.
t_5	System prioritized mandatory $J^2_{1,2}$ over optional $J^2_{1,1}$.
t_6	Only optional jobs remain and $E_{curr} > E_{opt}$. The system prioritizes $J^2_{1,1}$ over $J^3_{1,2}$ due to its tighter deadline.
t_7	$J^3_{1,2}$ (the only job) gets scheduled.
t_8	$J^4_{1,2}$ (the only job) gets scheduled.

job $J_{i,j}$. Table 5.2 shows the action taken by the scheduler along with the reasoning at each time step. Here, the mandatory part of $J_{1,2}$ is longer than the mandatory part of $J_{1,1}$ because, for the second data sample, the classifier is not confident enough with the result after the execution of the first layer. Note that to simplify the illustration, this example assumes that each layer requires one time unit to execute and the partition (mandatory vs. optional) is known ahead of time. The proposed scheduling algorithm (described in Section 5.4) handles further complexities such as the different execution times of different layers, multiple time units per layer, dynamic partitioning, and power-failure during the execution of a layer.

Setting E_{man} and E_{opt}. The E_{man} is set to the minimum energy required to turn on the MCU and execute an atomic fragment. During the compile time, Zygarde programming tools (described in Section 5.5) estimates the maximum energy required by any atomic fragment by running Energy-Trace++ [200] and sets this threshold. The E_{opt}, on the other hand, is by default set to the energy required to fill up the capacitor. This is because, once the capacitor is full, the excess energy gets wasted if nothing is executing on the MCU. By executing optional tasks, I minimize this wastage, and increase the performance of the system. However, a developer can override these values using the APIs provided (Section 5.5.2). If E_{opt} is small, e.g., comparable or equal to E_{man}, then all the optional portion of the tasks will execute, causing starvation of the mandatory tasks. On the other hand, if E_{opt} is high, the optional portion of the jobs will never execute.

5.2 Modeling Intermittent Energy

This section models the energy harvesting pattern and derive a single metric, namely the η-factor, which characterizes an energy harvester used in a particular application. The scheduler uses this metric for energy-aware scheduling.

5.2.1 Energy Event

Batterless systems experience intermittence due to two reasons– (1) unavailability of harvestable energy and (2) required time to accumulate sufficient energy before executing. The intermittence pattern is stochastic, hard to predict, and heavily dependent on the available harvestable energy. Hence, instead of modeling and predicting the harvested energy to characterize a harvester, I define a binary random variable, called the *energy event*, that denotes the state of the energy storage to have at least ΔK Joules of energy over a ΔT period, where ΔK and ΔT depend on the application as well as the underlying system. I empirically determine and set ΔK to E_{man}'s values based on the application necessity and chosen energy source. Figure 5.4 illustrates energy events for an RF source with $\Delta K = 10mJ$ and $\Delta T = 5s$.

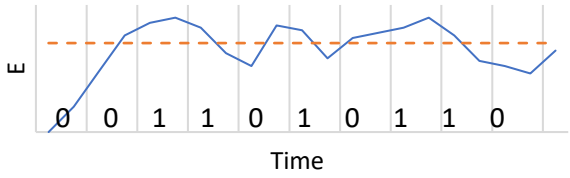


Figure 5.4: Illustration of energy events.

Furthermore, instead of directly dealing with the harvested energy, it is often easier to observe the physical phenomenon behind energy generation. For instance, for a piezoelectric harvester installed inside a smart shoe, the number of footsteps that generate at least ΔK Joules over ΔT time can be equivalently used to define an energy event. Likewise, a certain light-intensity for a solar harvester and a certain number of packet transmissions for an RF harvester over ΔT time can be used to define energy events for these systems.

To characterize an energy harvester using the definition of energy events, I conduct a two-month long study on solar and RF harvesters (using empirically collected data) and human footsteps (using the dataset [219]) to analyze the energy event's pattern. Our study reveals that energy events occur in bursts, i.e., every harvester has a tendency to maintain its current binary state, and there is a probabilistic relation between consecutive energy events over a period.

For instance, when a person starts walking, the probability that they will continue to walk is high over the subsequent few time units, and the probability decreases with time. Conversely, when a person is *not* walking, the probability that they will continue not to walk will be high over the following few time units and will diminish with time. This observation enables us to impose

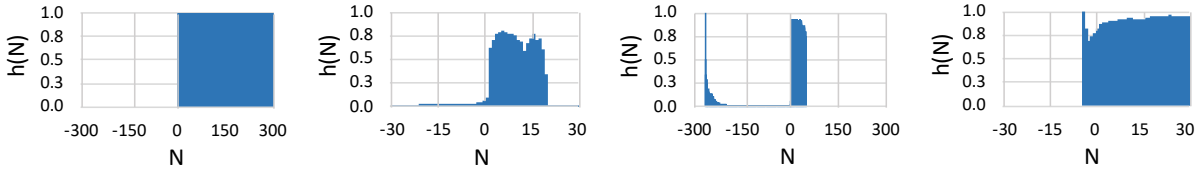
conditional probability on future energy events, given the recent history of energy events—which is the key to characterize an energy harvester. I denote an energy event at time t using the random variable $H_t \in \{0, 1\}$.

5.2.2 Conditional Energy Event

I define conditional energy event, $h(N)$ as the probability that an energy event will occur, given the immediately preceding N consecutive energy events have occurred (for $N > 0$) or have not occurred (for $N < 0$):

$$h(N) = \begin{cases} p(H_t = 1 \mid H_{t-1} \wedge \ldots \wedge H_{t-N} = 1), & \text{for } N > 0 \\ p(H_t = 1 \mid H_{t-1} \vee \ldots \vee H_{t-N} = 0), & \text{for } N < 0 \end{cases} \tag{5.1}$$

To illustrate, h(10) = 90% implies that an energy event will occur with 90% probability if ten immediately preceding consecutive energy events have occurred.



(a) Ideal Source (η=1) (b) Kinetic (η=0.65) (c) Solar (η=0.84) (d) RF (η=0.76)

Figure 5.5: Conditional energy event for (a) persistent power source, (b) piezo-electric harvester, (c) stationary solar harvester, and (d) stationary RF harvester. Here, $\Delta T = 5$ minutes.

Figures 5.5(a)-(d) show $h(N)$'s distribution for a persistently-powered and three energy harvested systems. To characterize an energy harvester, I measure the Kantorovich-Wasserstein (KW) distance [220] between its distribution, $\mathcal{H}^{(i)}$ from an ideal (persistent power) source, $\mathcal{P}$ to obtain:

$$KW(\mathcal{H}^{(i)}, \mathcal{P}) = \int_{-\infty}^{+\infty} |CDF(\mathcal{H}^{(i)}) - CDF(\mathcal{P})| \tag{5.2}$$

In Figure 5.5, I observe that $h(N)$ drops when $|N|$ increases. For example, in Figure 5.5(b), $h(N)$ drops after $N = 20$ since the person I studied never walked for more than 100 minutes. Similarly, in Figure 5.5(c), after about five hours of consecutive energy events (i.e., light intensity $>$ 2730 lux), the probability of energy event drops as the stationary solar harvester was placed beside a window that does not get enough light after five hours. I also notice that after about 19 hours of the absence of any energy event (i.e., light intensity $<$ 2730 lux), the following energy event's probability is high

as the sun shows up again at the window.

5.2.3 The η Factor

Despite being informative, the KW distance has a limitation that not all $h(N)$'s are estimated using the same number of instances. Hence, I normalize the KW score against a purely random harvesting pattern, R, to obtain a revised metric, called the η-factor:

$$\eta = 1 - \frac{KW\big(\mathcal{H}^{(i)}, \mathcal{P}\big)}{KW\big(\mathcal{R}, \mathcal{P}\big)} \tag{5.3}$$

The value of η lies in $[0, 1]$, and it measures how close a harvester's harvesting pattern is to a constant energy source. For a persistently-powered system, $\eta = 1$, and for an energy harvester that shows no apparent pattern has $\eta = 0$. For any other energy harvesting system, the η-factor will lie in-between, and it is generally high for small $|N|$. A higher η-factor indicates less randomness in its energy harvesting pattern and encourages a scheduler to make more aggressive decisions on scheduling tasks in the next few time slots. The η-factor needs to be empirically estimated for a given application-specific system.

5.3 Modeling DNN Tasks

In this section describes the task model of Zygarde, training procedure of agile DNN, and construction of semi-supervised k-means classifier.

5.3.1 Task Model

Zygarde extends the task model described in Section 3.2 for imprecise DNN tasks. The extended definition of computing tasks, jobs, and units are provided below.

Tasks, Jobs, Units, and Fragments. Similar to Section 3.2, Zygarde considers the processing of a sensor data stream for each classification task as an *imprecise sporadic task* [206], $\tau_i = (T_i, D_i, c_i, e_i)$.

A *job*, comprises of an ordered sequence of *units* (introduced in Section 3.2). The first M units are mandatory and must be completed before the deadline, whereas the rest of the units of a job are optional and can be executed if time and resources are available. Such a partitioning scheme is known as the imprecise computing model in real-time systems literature [206, 115, 221]. In this paper, however, the partition (i.e., the value of M) is dynamic and depends on the input data.

In Zygarde, a job that executes an L-layer agile DNN has L units, where each unit corresponds to processing one DNN layer, along with the execution of the corresponding semi-supervised k-means classifier. The cluster centroids of this semi-supervised k-means classifier are updated with new

unlabeled data. Based on the input data, Zygarde may decide to exit from a unit or continue executing the next unit. The decision is based on a *utility* function, which is described next.

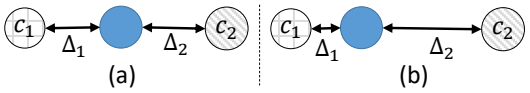


Figure 5.6: Two nearest clusters c_1 and c_2 of an input (in the middle) is shown: (a) early exit does not happen since $|\Delta_2 - \Delta_1|$ is small; (b) early exit happens since $|\Delta_2 - \Delta_1|$ is large.

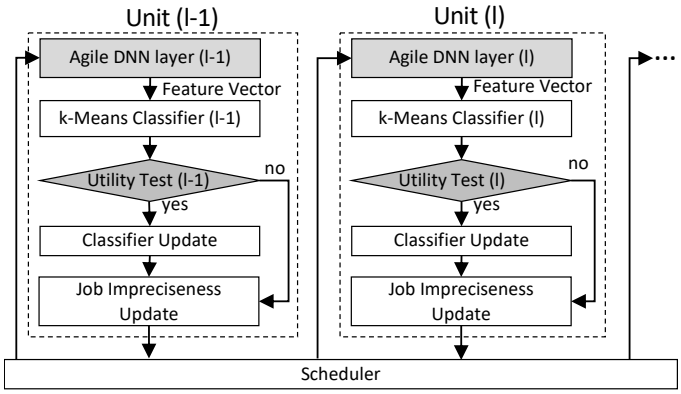


Figure 5.7: Sequential execution of units of an agile DNN.

Dynamic Partitioning and Utility. Unlike traditional imprecise computing models [206, 115, 221] where the partition of a job into mandatory and optional parts is known, the number of mandatory units in Zygarde is determined at runtime. I propose a *utility* function that estimates the confidence in classification at a given unit for that job. This represents the *utility of the data* where higher utility at earlier units is desirable.

Since I use a k-means classifier, I assume that a classification result is more likely to be correct if the input data sample being processed is unambiguously close to exactly one of the k means. To achieve this, I compute the L1 distance of an input data sample (represented in terms a DNN layer and then vectorized) from two of its closest of the k means, Δ_1 and Δ_2, and if their difference $|\Delta_2 - \Delta_1|$ is above a *unit-specific* threshold, I decide to classify it as belonging to its closest cluster; otherwise, the computation of the DNN continues to the next unit. The process is illustrated by Figure 5.6.

The utility function described above runs in linear time with the number of clusters, i.e., $\mathcal{O}(k)$. It is lightweight, energy-efficient, and suitable for resource-constrained systems as it uses the byproduct of clustering-based classification which computes the cluster distances, Δ_i's anyways. It, however, depends on an offline-estimated threshold. Section 5.3.3 describes how the utility threshold is computed using an empirical dataset. Section 5.9 further discusses alternative utility functions that are suitable for other types of classifiers.

Preemption and Task Switching Zygarde allows limited preemption [129] where a job can be preempted only after a unit completes its execution. The scheduler kicks in at the completion of a unit and at the deadline of a job. After the execution of a unit, a job returns to the job queue with updated utility and imprecise status (mandatory or optional). Then the scheduler chooses the next highest priority job from the job queue using the priority function described in Section 5.4. By prohibiting preemption of a unit, Zygarde reduces context switching and read-write overheads, and minimizes the memory requirements to $\mathcal{O}(N)$ for N jobs by using double-buffering [130]. Figure 5.7 shows the execution of two units. Each unit is shown as a large dotted rectangle and it contains four logical modules that are shown as solid rectangular boxes.

5.3.2 Agile DNN Construction

Unlike previous works where inference happens only at the last layer [222] or where a second classifier is used at hidden layers to decide early exit [99], in Zygarde, the output of *any* hidden layer can be directly used as a feature that gets classified by a k-means classifier. Features obtained in this manner neither guarantee that the data samples from the same class are closer nor guarantee that the data samples from different classes are farther in the feature space. Hence, to ensure that the feature representation obtained after an early exit from the DNN execution maximizes the separability of different classes and minimizes the distance between examples of the same class, Zygarde employs a *layer-aware* loss function.

Layer-Aware Loss Function. A convex combination of contrastive losses [213] at each layer is used as the loss function of an Agile DNN – which is called the *layer-aware* loss function:

$$LA = \sum_{i=1}^{L} a_i \times LC\Big(W^i, X_1^i, X_2^i, \cdots, X_N^i\Big) \tag{5.4}$$

where, a_i is the convex coefficient for the i^{th} layer and $\sum_{i=1}^{L} a_i = 1$; L and N represent the total number of layers and classes, respectively; W^i represent the weights of the i^{th} layer; $X_1^i, X_2^i, \cdots, X_N^i$ are the output vectors corresponding to the members of each class at the i^{th} layer; and LC is the contrastive loss function. For two classes, LC is defined as:

$$LC\Big(W^i, X_1^i, X_2^i\Big) = \frac{1}{2}\Big(1 - Y\Big)\Big(G_{W^i}(X_1^i) - G_{W^i}(X_2^i)\Big) + \frac{1}{2} Y \max\Big(0, \Delta - G_{W^i}(X_1^i) - G_{W^l}(X_2^i)\Big) \tag{5.5}$$

where, $G_{W^i}(X_n^i)$ is the output feature set of a member of the n-th class ($1 \leq n \leq N$) at layer i. The coefficient $Y = 0$, if X_1 and X_2 belong to the same class, and $Y = 1$, otherwise. Δ represents

the margin between the members of different classes in the feature space.

Training Agile DNN. To train an agile DNN, I use a siamese network architecture [213] as shown in Figure 5.8. In a siamese network, there are two identical neural networks, called the sister networks, that share the same weights. From the labeled dataset, I select pairs of data points and use them as the inputs to the twin networks. Among the selected pairs of data points, 50% belong to the same class, while the rest belong to different classes. Unlike [213], which only uses the contrastive loss at the last layer (LC_3), I use the layer-aware loss function (Equation 5.4) at every layer to train these networks. I perform an exhaustive search for hyper-parameter tuning and to determine the weights of each layer. After the training, I use only one of the sister networks for inference. During inference, I obtain a representation of the input data from each layer, and use them as features for the semi-supervised k-means classifiers. Note that, for convolution layers, I flatten the output of a layer to get a vector instead of a tensor in order to be able to compute the L1-norm during the clustering-based classification step of Zygarde.

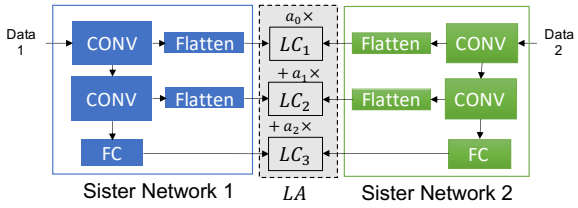


Figure 5.8: Training agile DNN with Siamese network.

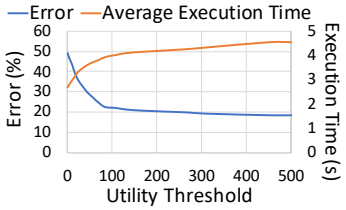


Figure 5.9: Effect of utility threshold on performance.

Note that although the combination of an agile DNN and semi-supervised k-means classifiers in Zygarde is inspired by *Anytime Neural Networks*, an agile DNN is different as it is a representation learner rather than a classifier. Besides, an agile DNN is trained using a siamese network and it only has one loss function, which is different from anytime neural networks that use multiple auxiliary loss functions. Furthermore, the exit policy and the utility function of an agile DNN is different from that of anytime neural networks, and are optimized for resource-constrained systems. Moreover, Zygarde forms an imprecise computing problem where an early exit from the network depends not only on the data but also on the time and energy budget.

5.3.3 Semi-Supervised k-Means Classifiers Construction

Zygarde maintains a semi-supervised k-means classifier at each unit. This section describes how to construct and update these classifiers.

Computing Cluster Centroids. Using the trained agile DNN, I obtain a feature representation from each layer for each data point in the training dataset. Using these representations, I train a semi-supervised k-means classifier corresponding to each layer of the agile DNN (see Figure 5.7). Using the labeled training data, I select the top N features using SelectKBest [223] and χ^2 tests, so that the features are computable on the resource-constrained target device. I utilize the labeled training data to determine the value of k (for k-means) and assign a class label to each cluster. Finally, I compute the centroid of each cluster in the selected feature space.

Determining Utility Threshold. Utility thresholds are crucial to determining whether a data point should exit from the current hidden layer or continue processing through the network. A smaller threshold is likely to force too early exits and thereby, a lower classification accuracy; whereas a larger threshold is like to delay exits and thereby, increase the inference latency. This trade off is demonstrated by Figure 5.9 for the first layer of the DNN on the CIFAR-100 dataset. For different layers, I observe similar trade offs. To determine a suitable utility threshold for each layer, I generate such a trade off curve and pick a utility threshold that ensures a desired minimum inference accuracy as configured by the programmer.

Updating Centroids at Run-Time. I incrementally update the *means*, i.e., the cluster centroids, of the k-means classifiers at runtime to evolve the classifiers over time and to learn from new examples—which is common in semi-supervised learning approaches [224, 225, 226]. Referring to Figure 5.7, this is done inside the *Classifier Adapter* when the classification result from the *k-means Classifier* passes the *Utility Test* at a unit. A new cluster centroid is computed by taking the weighted average of the current cluster centroid and the current example. Taking the weighted average guards against abrupt changes to the centroids due to the presence of an outlier or incorrect classifications. If the distribution of the input data points changes (e.g., the system is deployed in a new environment), the cluster centroid gradually shifts towards the new mean of the data points as it encounters the new data points.

Updating Centroids beyond Mandatory Layers. Due to early exit from the network, a data

sample fails to update the k-means models of the deeper layers. To achieve this, Zygarde adapts the cluster centroids of the deeper layers using the corresponding cluster heads of the layer from which the example exits early. Mathematically, the update operation is $c^{i+1} = \frac{1}{r}\sigma\big(W^{i+1} \times r \times c^{i}\big)$.

Here, c^{i} and c^{i+1} denote the corresponding cluster centroid of the k-means classifiers of layers i and $i+1$; W^{i+1} denotes weights (including the bias term) for layer $i+1$; r denotes the size of a cluster; and $\sigma(x) = \frac{x+|x|}{2}$ is the non-linear activation function [227].

Since this technique estimates the cluster centroids of a deeper layer instead of actually running the data samples through those layers, it saves $\mathcal{O}(r)$ multiplication operations and performs the operation in $\mathcal{O}(1)$; at the maximum approximation error of $(\sum_{k=1}^{r} |W^{i+1} \times X_k^i| - |W^{i+1} \times \sum_{k=1}^{r} X_k^i|)/(2r)$.

5.4 Real-Time Scheduler

This section describes the real-time scheduler in Zygarde. First, it introduces an online scheduling algorithm for dynamically-partitioned, sporadic, imprecise tasks on a persistently-powered system. Then, it describes extension of the algorithm for intermittently-powered systems.

5.4.1 Scheduler for Persistent Systems

Despite being an optimal online scheduling algorithm for sporadic tasks, the earliest deadline first (EDF) algorithm [228] is not directly applicable to Zygarde as EDF does not consider the accuracy of a DNN. Furthermore, traditional scheduling algorithms for imprecise tasks [206, 229] are not directly applicable to Zygarde as well since the mandatory and optional portions of an agile DNN is determined dynamically at runtime.

To address these challenges, I propose a priority function to prioritize the *units* of Zygarde. At the end of the execution of an unit, the scheduler selects the highest priority unit as the next unit for execution. The priority function considers not only the remaining deadline of a job, but also the utility (as defined in Section 5.3.1) and the dynamically determined impreciseness status (i.e., mandatory vs. optional) of a unit:

$$\zeta_{i,j}^{l} = \Big(1 - \alpha(d_{i,j} - t_c)\Big) + \Big(1 - \beta\Psi_{i,j}^{l}\Big) + \gamma_{i,j}^{l} \tag{5.6}$$

where the first term represents the remaining deadline, which is the difference between a job's absolute deadline $d_{i,j}$ and the current time t_c. The second term ensures that units with lower utility score $\Psi_{i,j}^{l}$ gets higher priority as these tasks need further execution for accurate classification. The third term is a binary variable $\gamma_{i,j}^{l} \in \{0, 1\}$ that denotes if the unit under consideration is mandatory

$\gamma^l_{i,j} = 1$ or optional $\gamma^l_{i,j} = 0$, which is determined at runtime based on the unit-specific utility threshold. α and β are scaling parameters that normalize the deadline and utility, which are the inverse of the maximum deadline and utility, respectively.

Note that, Zygarde supports multiple tasks including multiple DNN tasks as long as the required memory does not exceed the available memory of the system. For non-DNN tasks or other absolute (non-imprecise) tasks, γ_i is always 1 and Ψ_i is a constant for all units. Ψ_i is user-defined based on the priority of the task.

5.4.2 Scheduling for Intermittent System

For an intermittently-powered system, I utilize the η-factor introduced in Section 5.2 to extend ζ as follows:

$$\zeta^l_{I_{i,j}} = \begin{cases} \big(1 - \alpha(d_{i,j} - t_c)\big) + \big(1 - \beta\Psi^l_{i,j}\big) + \gamma^l_{i,j}, & \eta E_{curr} \geq E_{opt} \\ \gamma^l_{i,j}\Big(\big(1 - \alpha(d_{i,j} - t_c)\big) + (1 - \beta\Psi^l_{i,j})\Big), & \eta E_{curr} < E_{opt} \end{cases} \tag{5.7}$$

Here, E_{curr} is the current energy of the system and E_{opt} is a threshold that determines if the system has enough energy to execute both mandatory and optional units. The expression ηE_{curr} is high enough to cross the threshold as long as at least one of the two variables η and E_{curr} is high-valued and the other is not extremely low. I identify two cases:

First, when ηE_{curr} is above the threshold, both mandatory and optional units are considered for scheduling. Intuitively, it captures the cases when (a) an energy harvester is predictable and generating at least sufficient energy to keep the capacitor charged, and (b) when an energy harvester is predictable with medium confidence and generating more than sufficient energy. The explanation of the three terms are omitted in this case since they are similar to the persistent power system as described in the previous section.

Second, when ηE_{curr} is below the threshold, only the mandatory units are considered for scheduling. It captures the cases when an energy harvester is – (a) unpredictable, (b) predictable but generates insufficient energy, and (c) predictable with medium confidence and generates sufficient energy.

ζ_I minimizes two types of energy waste in batteryless systems: 1) wasted energy due to executing unnecessary portions of a job, and 2) wasted energy due to not executing any job while the harvester gets enough energy from the source to keep the capacitor charged [48]. The first type of waste is avoided by scheduling conservatively when $\eta E_{curr} < E_{opt}$, and the second type of waste is avoided

by executing optional units when $\eta E_{curr} \geq E_{opt}$.

5.4.3 Schedulability Condition

A set of N sporadic tasks is schedulable by an imprecise scheduler if the total utilization, $\sum_{i=1}^{N} \frac{c_i}{T_i} \leq 1$ [206], where the execution time, c_i includes only the mandatory portion of the task. Scheduling sporadic jobs in an intermittently-powered system adds further complexity as power outages essentially blocks the CPU and thus increases the CPU utilization by increasing the execution time c_i, although no task is actually executing on the system as power is out. In order to incorporate energy intermittence into the schedulability analysis framework, I model power outages event as a very high-priority job of a sporadic *Energy Task*. Energy Tasks are either a NOP task or a harvesting task.

In this extended task set having N+1 tasks, the schedulability condition becomes $\sum_{i=1}^{N} \frac{c_i}{T_i} + \frac{c_e}{T_e} \leq 1$, where, c_e and T_e are the duration and interval of energy intermittence. The execution time of an energy task, c_e is related to the η-factor of the system. The probability that an energy harvester will remain in its current power-outage state for the next d energy events can be derived from η using the properties of a geometric distribution $\eta^d(1-\eta)$, whose expected value is $E[c_e] = \eta/(1-\eta)$. Given this, the necessary condition for an intermittent computing system to be able to schedule N sporadic tasks is $T_E \geq \frac{\eta/(1-\eta)}{1-\sum_{i=1}^{N}(c_i/T_i)}$.

5.5 Zygarde Programming Model

Zygarde's programming model consists of: (i) a *Network Trainer* tool that is used by the developer to train and compress agile DNNs and to generate the k-means classifiers and corresponding hyper-parameters; and (ii) *APIs* for the target embedded device which are used by the developer to write custom C application for an intermittently-powered MSP430 MCU. A high-end development machine is recommended for these one-time, offline steps. At the end of these steps, I obtain an executable binary file for the MSP430 MCU.

5.5.1 Zygarde Network Trainer

The network trainer takes four inputs from the developer, i.e., (1) a labeled training dataset, (2) DNN architecture/model, (3) timing parameters, and (4) the η-factor. The network trainer generates C header files as the output – which are used by the APIs for the target embedded platform described in the next section. Figure 5.10 shows the intermediate steps inside the Zygarde network trainer.

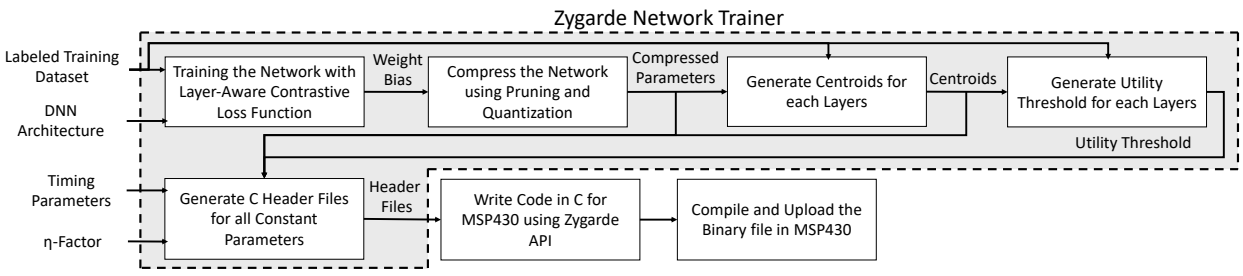


Figure 5.10: Zygarde Programming Framework.

At first, the network trainer trains the agile DNN model using the labeled dataset using the layer-aware loss function described in Section 5.3.2. It relies on an exhaustive search for hyper-parameter tuning, and outputs the weights and bias parameters of the network. Considering the limited memory of the target device, the DNN is compressed and pruned to reduce its memory requirement [85, 86, 230, 231, 232, 34, 35]. The network trainer also checks if the compressed network fits into the memory of the target device and signals an error if it does not. Using this compressed agile DNN model and the input dataset, the network trainer generates the cluster centroids and the utility threshold for each layer of the network – following the steps described in Section 5.3.3.

Finally, C header files are generated that contain the compressed DNN parameters, cluster centroids, utility thresholds, features used in clustering, and task-specific timing parameters (e.g., deadline and period) and the energy parameter (i.e., η-Factor).

5.5.2 Zygarde APIs

The Zygarde APIs extend open source SONIC [8] APIs for intermittent DNN computing by incorporating Zygarde-specific capabilities, such as early termination, cluster-based inference, and scheduling. Zygarde APIs are divided into two categories: (1) external APIs, and (2) internal APIs.

The external APIs contain library functions that a developer uses to implement early-exit capable agile DNNs for feature representation and the k-means classifiers. These library functions rely on the header files generated by the network trainer to access the classifier parameters and are sufficient for most developers who only want to define the high-level logic of their application. For instance, to implement the two tasks shown in Figure 5.11 on an MSP430 platform, a developer essentially has to write a C program that uses Zygarde external APIs to implement a state diagram similar to the one shown in Figure 5.12.

The internal APIs provide some of the lower level functions that are primarily used by the

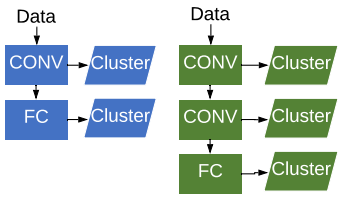


Figure 5.11: Two sample DNNs.

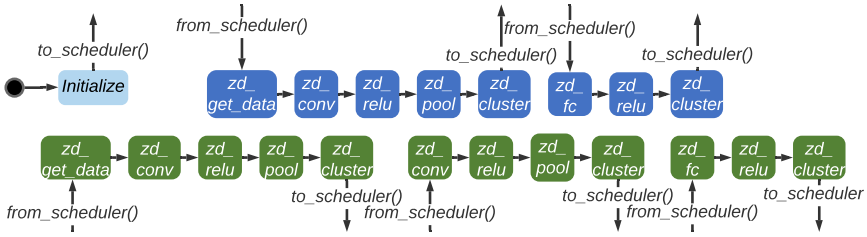


Figure 5.12: State diagram of the sample DNNs.

external APIs. These APIs implement several key features of Zygarde including the scheduler, job queue management, time management, and handling the timers. If a developer wishes to change the default implementation of any of these functions, they need to override these methods to provide their own implementation.

5.6 Implementation

Computing Device. I use TI-MSP430FR5994 [175] MCU (shown in Figure 5.13) that has 256KB of FRAM, 8KB of SRAM, 6-channel DMA, a low energy accelerator (LEA), and an operating voltage range of 1.8V to 3.6V. During the training phase, I use an Intel Core i7 PC with RTX2080 GPU to train and compress the agile DNN, initialize the centroids of semi-supervised k-means classifiers, and compute the utility thresholds.

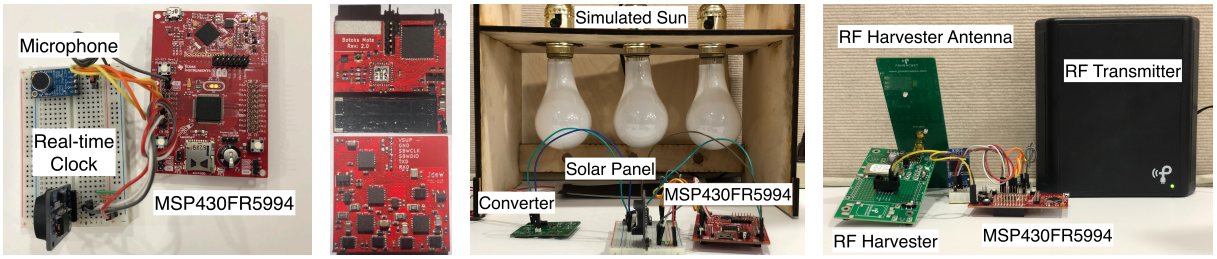


Figure 5.13: Zygarde experimental setup.

Energy Harvester. Figure 5.13 shows our solar and RF energy harvester setup. The solar harvester includes an Ethylene Tetrafluoroethylene (ETFE) based solar panel [233] and a step-up regulator [196]. I use the Powercast harvester-transmitter pair [192, 193] to harvest RF energy. Like previous works on intermittent systems [8, 13], both harvesters use a 50mF capacitor.

Sensor Peripheral. I use an electret microphone [197] and the built-in ADC in MSP430 for

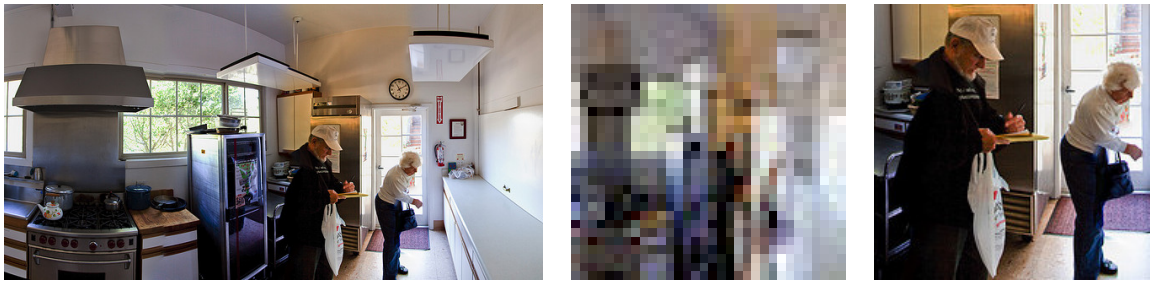
acoustic sensing. For visual sensing, I use an OV2640 CMOS camera module [234] connected via I2C and SPI. Using LEA and DMA, I perform FFT on audio data and write the audio data to the FRAM without involving the CPU.

Time Keeping. Like [12, 4], I use a real-time clock, DS3231 [164] for timekeeping in most of the experiments. I use this clock only during the power up to sync and maintain the internal clocks of the MCU. This clock is easily replaceable with an SRAM or capacitor-based timekeeping system during power outages [1, 2]. In order to quantify the effect of such a batteryless timekeeper on Zygarde, I implement and use an open-source remanence clock, called CHRT [3], in one of the experiments. To use the CHRT correctly with Zygarde, the energy required to charge the CHRT has been considered when defining the energy events to estimate the η-factor.

Libraries. Zygarde uses an open-source intermittent execution model SONIC [8] and related APIs (e.g., ALPACA [13]). I use Tensorflow [235] for training the DNN models.

5.7 Microbenchmarks

In this section evaluates each component of Zygarde using datasets and compare Zygarde with baseline algorithms. The effect of capacitor size and remanence clock on Zygarde is also observed.



(a) Original Image (b) Downsampled Only (c) Targeted Crop

Figure 5.14: Visual Wake Word dataset: (a) Original image (640×320), (b) Only downsampled (32×32), (c) After targeted cropping and downsampling (32×32).

5.7.1 Datasets and Environments

Datasets and DNNs. To evaluate the performance of different components of Zygarde, I use four datasets: MNIST [208], ESC-10 [189], CIFAR-100 [210], and Visual Wake Word [211]. MNIST is a popular image dataset having 80,000 28×28 pixel images (60,000 for training and 10,000 for testing) and ten classes, and it has been used for evaluating state-of-the-art intermittent computing systems [8]. ESC-10 also has ten classes and 44.1 kHz five seconds-long audio clips. I use 1s audio

Table 5.3: DNNs considered in this section.

Dataset	**MNIST**	**ESC-10**	**CIFAR-100**	**VWW**
Layers	CONV CONV FC FC	CONV CONV CONV FC	CONV CONV FC FC	CONV CONV CONV CONV FC
Dimensions	20×1×5×5 100×20×5×5 200×1600 500×200	16×1×5×5 32×16×5×5 64×32×5×5 95×256	32×3×5×5 64×32×5×5 384×1600 192×384	16×3×5×5 32×16×5×5 64×32×5×5 64×64×5×5 192×256
Parameters Size	8×10^3	55×10^3	27×10^3	14×10^3

downsampled to 8KHz. I split the dataset into 80% training and 20% testing datasets. CIFAR-100 contains 32×32 pixel color images from 100 classes. It has 500 training images and 100 testing images per class. In order to fit this dataset in the MSP430, I use randomized subsets of 5 classes from the dataset for 100 iterations and report the average.

Visual Wake Word (VWW) is a large dataset containing 82,783 training and 40,504 validation images from the state-of-the-art vision dataset COCO [236]. To fit these images into the MCU's memory, I first crop an image to move the target object (human) in the center and then downsample the cropped image to 32×32 pixels. Note that if I only downsample the image to 32×32 pixels without cropping it first, the resultant image scales down the target object (human) so much that they are not recognizable anymore. Figure 5.14 shows an example image from the VWW dataset, followed by two downsampled versions of it– with and without cropping.

I implement four compressed networks summarized in Table 5.3. Our feature-maps after each layer consist of a maximum of 150 features selected using k-best select. These feature-maps are used for the semi-supervised k-means classifiers. The scheduler has a queue-size of 3.

Controlled Energy Sources. To evaluate the system with different η-factors (ΔT=1s and ΔK=9.36mJ), I perform controlled experiments. To determine the value of ΔK, I run the system for multiple iterations and take the highest observed energy consumption. I vary the distance between the transmitter and the receiver between 1-5 feet for RF. I simulate solar power with three dimmable bulbs with varying intensity (5.6 Klx - 35 Klx) as shown in Figure 5.13. The seven scenarios considered for the evaluation is described in Table 5.4. Note that, outdoor scenarios and windowed rooms are used to get the sunlight for the real-life experiments in Section 5.8.

Table 5.4: Algorithm Evaluation Scenarios

System	Energy Source	η	Average Power (mW)
1	Battery	1	
2	Solar	0.71	600
3	Solar	0.51	420
4	Solar	0.38	310
5	RF	0.71	58
6	RF	0.51	71
7	RF	0.38	80

5.7.2 System Overhead

Figure 5.15 shows the overhead of different components of Zygarde described in Section 5.1. To measure the execution time and energy consumption, I use the TI eZ-FET debug probe with EnergyTrace++ [200], which provides milliseconds and μJ resolution data. I isolate each component of Zygarde and report the average overhead from five repeated measurements. To measure smaller overheads I repeat the experiment for multiple iterations (e.g., 2000 iterations for energy manager) and report the mean overhead of a single execution. I use a persistent power source during overhead measurements.

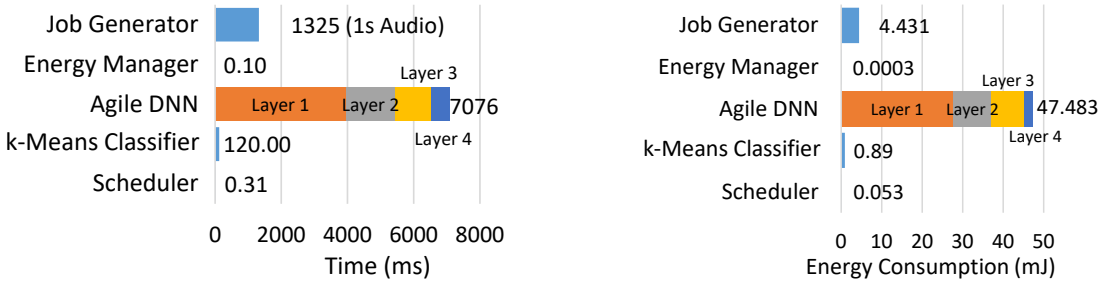


Figure 5.15: Overhead of Zygarde.

The job generator reads 1s audio data from the microphone, performs FFT, and writes it to the FRAM in 1.325s. The first convolution layer (ESC-10 network of Table 1) has 2.6×-3.6× higher execution time than other convolution layers due to larger input dimension. Using max-pool with stride decreases the input size, inference time, and energy consumption at each layer. The last fully-connected layer performs 50% less multiplications than the previous layer and thus has a lower cost. Each job executes the semi-supervised k-means classifiers at most four times. It is 14× faster and 13× more energy-efficient than executing the whole DNN. Execution of the k-means classifier includes performing the utility test, classifying with k-means classifier and updating the model

centroids for adaptation. For N examples in the system, the scheduler kicks in $4N$ times and the overhead of this specific example with three jobs are 3.72 ms and 636μJ, which is less than 1% of the overall cost of processing an example. The energy manager has negligible cost and runs once every time the scheduler executes.

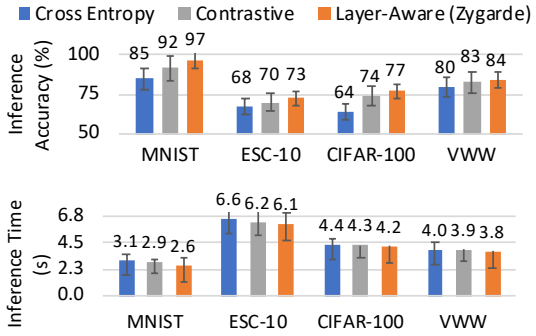


Figure 5.16: Comparison of Loss Functions with Early Exit.

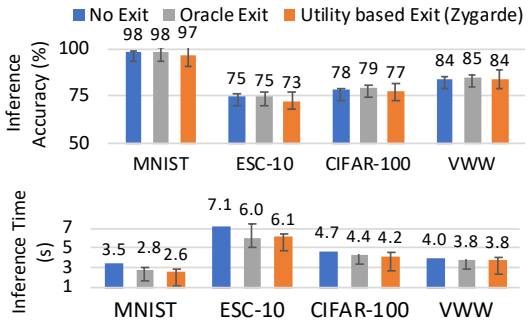


Figure 5.17: Comparison of the Termination Policies.

5.7.3 Effect of Layer-Aware Loss Function

In Figure 5.16, I compare our proposed layer-aware loss function with cross-entropy loss [212] and contrastive loss [213] functions when early termination is in action. Since loss functions are equally applicable to both persistently-powered and energy-harvested systems, I conduct this experiment in a persistently-powered setting. I train three agile DNNs with three different loss functions that have the same network structure, hyper-parameters, and training dataset. All three networks use the proposed utility test where the utility-threshold is determined during training.

Though the loss functions achieve similar accuracy (≈98% for MNIST and ≈75% for ESC-10) without early termination, their performance varies when early termination is applied. Note that, the inference accuracy of ESC-10 suffers due to downsampling of the 5s and 44KHz data samples to 1s and 8KHz data samples. In Figure 5.16, the layer-aware loss function demonstrates 4.13%-13.40% higher accuracy than cross-entropy loss by forcing the layers to learn distinguishable features [237]. It also decreases the average inference time by upto 13.97%, by executing the final layer of 14%-26% less jobs compared to cross-entropy loss. Layer-aware loss function further achieves 2%-5% higher accuracy and 2%-9% less average inference time than the contrastive loss function. Thus, layer-aware loss function achieves higher accuracy and lower inference time than other loss functions when early termination is active.

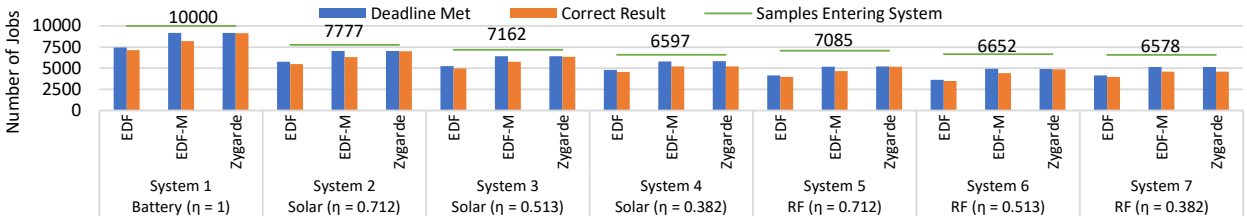


Figure 5.18: Real-time Scheduling for different Systems on MNIST test dataset.

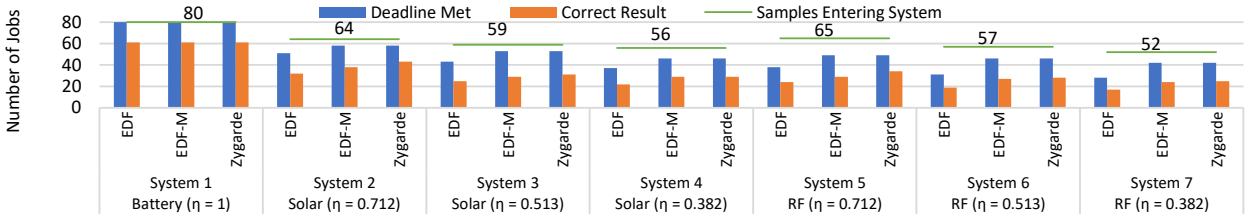


Figure 5.19: Real-time Scheduling for different Systems on ESC-10 test dataset.

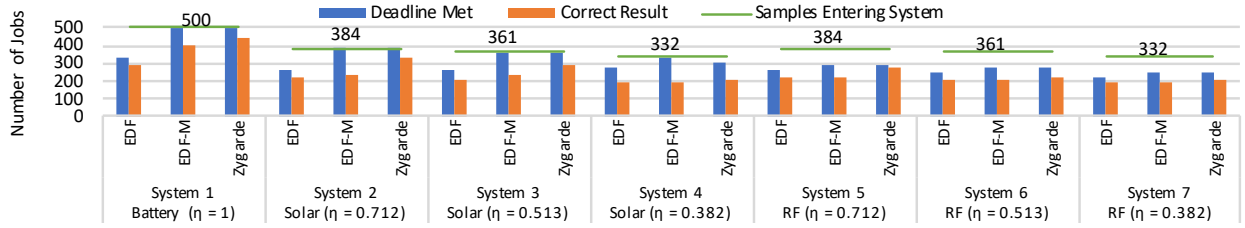


Figure 5.20: Real-time Scheduling for different Systems on CIFAR-100 test dataset.

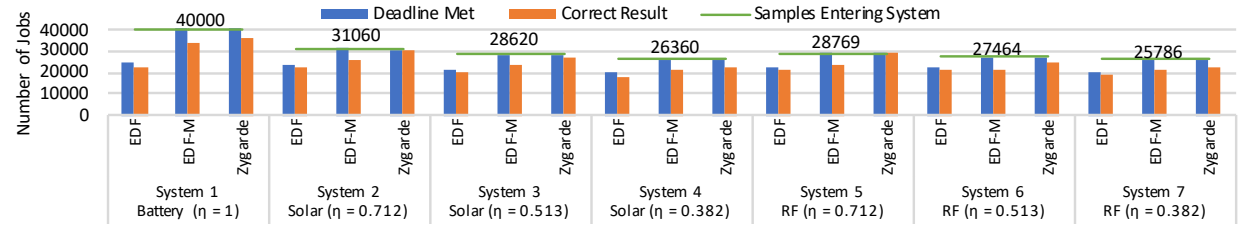


Figure 5.21: Real-time Scheduling for different Systems on Visual Wake Word test dataset.

5.7.4 Effect of Early Termination

In Figure 5.17, I evaluate the proposed utility test by comparing it with a system that does not implement early exit and an oracle that knows the exact number of units needed for each data sample. I use the same persistently-powered system and dataset as in Section 5.7.3. All of these systems use the same trained network with the layer-aware loss function. Utility-based termination (exit) achieves similar accuracy while lowering the average inference time by 4%-26%. The difference in accuracy between these systems is below 2.5%.

5.7.5 Performance of the Real-Time Scheduler

I evaluate the proposed scheduling algorithm for dynamic imprecise tasks in both persistently and intermittently powered systems for four different η-factors and two different CPU utilization described in Table 5.4. To compare our proposed algorithm, I choose earliest deadline first (EDF) and one of its variants– earliest deadline first mandatory (EDF-M). EDF-M schedules only the mandatory portions of the jobs. I choose EDF as a baseline because it is the optimal online scheduling algorithm for sporadic tasks. Here, both Zygarde and EDF-M use the proposed utility test to partition jobs into mandatory and optional units. For the fairness in comparison, successful completion of a job's mandatory units before deadline makes the job schedulable in all algorithms. Note that, I discard a job after its deadline to avoid domino effect [238].

Persistently Powered System. Figure 5.18 shows the performance of proposed scheduling algorithm for MNIST dataset for $T = 3s$ and $D = 6s$. As the CPU utilization (U) is greater than one, none of the schedulers can schedule all the tasks even on persistent power. However, with early termination, EDF-M and Zygarde schedule 17% more jobs. In Figure 5.19, I schedule 80 jobs from the ESC-10 dataset, where $U < 1$, $T = 0.36$ minutes, and $D = 0.72$ minutes. Persistently powered system (System 1) can schedule all the tasks with EDF, EDF-M, and Zygarde. In Figures 5.20 and 5.21, I schedule 500 and 40,000 jobs for CIFAR-100 and visual wake words (VWW) datasets, respectively, where the deadline is twice the period. In both cases, EDF-M and Zygarde schedules all the jobs while EDF fails to do so. As successfully scheduling only the mandatory units of a job before deadline is sufficient to be schedulable, EDF-M schedules similar number of jobs as Zygarde. However, Zygarde achieves higher accuracy by opportunistically executing optional units.

Intermittently Powered Systems. For intermittent systems (Systems 2-7), EDF-M schedules 14.98%-19.51% more jobs for MNIST, 9.44%–20.70% more jobs for ESC, 8.59%–33.59% more jobs for CIFAR, and 16.97%–24.53% more jobs for VWW than EDF. If the utility tests were optimal, EDF-M would have produced correct results for all the scheduled jobs. However, due to the limitation of utility tests, Zygarde increases the number of scheduled jobs that produce the correct results by up to 27.60% by executing some of the optional units. I observe that, Zygarde increases the performance (i.e., the number of scheduled jobs that produce correct results) from EDF-M when η is high. With low η, the performance of Zygarde and EDF-M becomes similar as no optional units are executed.

It is interesting to notice that despite having the same η, solar powered systems schedule 9% - 31% more jobs than RF powered systems due to more available power.

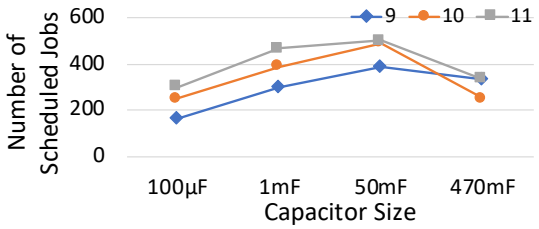


Figure 5.22: Effect of Capacitor

5.7.6 Effect of Capacitor Size

The goal of this experiment is to quantify the effect of the capacitor's size on scheduling. I use the CIFAR-100 dataset and its corresponding DNN (Table 5.3), and power the system from an intermittent RF energy source ($\eta = 0.51$) at around 0.5m distance. The period of the tasks are varied between 9s to 11s and the deadline is set to twice the period. I use four different capacitors: 0.1mF, 1mF, 50mF, and 470mF. This setup and workload stress tests the system and forces the scheduler to miss the deadline when the capacitor values are too small or too high. Figure 5.22 shows that when the capacitor value is below 50mF, more tasks miss their deadlines as they re-execute an atomic fragment when the power goes off before its completion. On the other hand, when the capacitor value is high (e.g., 470mF), tasks miss deadline due to the extra time required to charge such a large capacitor. Hence, I choose to use a 50mF capacitor for the rest of the experiments. Note that although I empirically determine a suitable capacitor for our experiments, one can roughly estimate the optimal value of the system capacitor, C by using a capacitor's energy equation, when the average input power, P, voltage across the capacitor, V, and the difference between the deadline and the total execution time of the task, δT, is known: $C = \sqrt{\frac{2P\delta T}{V^2}}$.

5.7.7 Effect of Remanence Clock

Keeping track of the time is crucial for a real-time scheduler and it is a hard problem, in general, for batteryless systems. To keep track of time reliably across power failures, recently, a batteryless remanence clock, namely the *Cascaded Hierarchical Remanence Timekeeper* (CHRT) [3] has been proposed for intermittently-powered systems. The CHRT clock has three modes or tiers. Its tier-1 yields near-perfect time-keeping accuracy, but has a range of only 100ms. On the other hand, the tier-3 offers 1s resolution, 100s range, and reports accurate time 80% of the cases, while reporting +1s

error for the rest of the time and rarely shows +2s, -1s or -2s error. I implement this clock following the open source hardware design (see Figure 5.13) and use it to power Zygarde – to implement a completely batteryless system. I evaluate the effect of batteryless CHRT clock on Zygarde's scheduler and compare it to the performance of Zygarde when it uses battery-powered RTC.

Table 5.5: Effect of Cascaded Hierarchical Remanence Timekeeper

System	Reboots	Power On Time	Scheduled Tasks using RTC	Scheduled Tasks with CHRT
2	67	77.67%	29989	29980
3	1252	71.48%	27401	27390
4	1820	65.83%	24921	24897

Table 5.5 shows the number of tasks meeting deadlines for both types of clocks for the systems 2–4 (see Table 5.4 for definitions). I do not show results for systems 5–7, which are powered by RF harvesters and require using CHRT tier-1 (which is optimized for RF), since the results are identical for both CHRT and RTC. I observe that the number of missed jobs increases with the number of reboots due to intermittent energy. Upon investigating the cause I find that during the positive error of the CHRT clock, the scheduler either reports the missed deadlines or terminates a job early, as it mistakenly thinks that the deadline has passed, and thus, continuing to execute these tasks is a waste of time. During negative error of the CHRT clock, the scheduler schedules a job despite the fact that it missed the actual deadline and triggers a domino effect that results in more tasks missing their deadlines. However, CHRT shows negative error $< 3\%$ time and often it compensates for a positive error. Overall, the loss of schedulable tasks due to the use of a batteryless clock is below 0.1%.

5.8 Real-World Application Evaluation

In the previous section, I compared the performance of Zygarde with different baseline algorithms. In this section, I observe Zygarde in two real-world applications. In the first application, I perform acoustic sensing and show how different scenarios affect the system performance. In the second application, I compare the performance of Zygarde with a state-of-the-art intermittent DNN inference system [8] for visual sensing tasks in a real-world setting.

5.8.1 Acoustic Sensing

Experimental Setup. This section evaluates Zygarde in real-world uncontrolled experiments using six audio event detection applications. Due to the presence of background noise and multiple audio

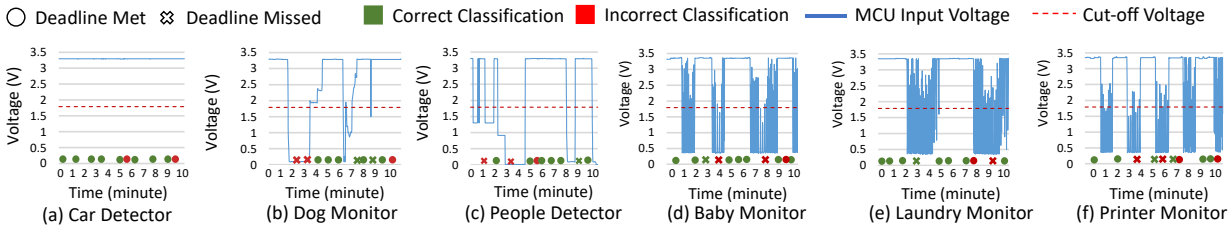


Figure 5.23: Real-life evaluation of Zygarde for acoustic event detection.

events in the data, these applications require DNN-based features for audio event representation and classification. Existing works show that DNN performs significantly better than threshold or classic machine learning-based audio event detectors in real-life noisy environments [239, 240].

Table 5.6: Real-life evaluation setup.

Application	Energy Source	Harvester Placement	Cause of Intermittence	Target Event	Other Events
Car Detector	Solar (74Klx to 111Klx)	Pavement	Vehicle on the closest lane	Car Honk	Silence, Dog, Human Voice, Car
Barking Dog	Solar (2Klx to 18Klx)	Under the Tree	People, objects and cloud	Dog Bark	Silence, Car, Car Honk, Voice
People Detector	Solar (1Klx to 5 Klx)	Edge of the Railing	People and cloud	Voice	Silence, Car, Honk, Dog Bark
Baby Monitor	RF (-0.48dB to -1.66dB)	On the Desk	Change of Distance	Crying Baby	Silence, Voice, Washer, Printer
Laundry Monitor	RF (-0.48dB to -1.91dB)	On the Counter	Change of Distance	Washer Status	Silence, Voice, Cry, Printer
Printer Monitor	RF (-1.59dB to -1.91dB)	On the Desk	Change of Distance	Printer Status	Silence, Voice, Crying Baby, Printer

Table 5.6 shows the the application environment, energy source, harvester placement, cause of energy intermittence, target event and other events present in the environment for the six applications. Each applications runs for 10 minutes and the audio sensor samples every two seconds. I play recorded sound, that are not used during training, 10 times from a speaker as the positive example. The relative deadline of the jobs are 3s which is the required execution time for the whole model. The agile DNN, consisting of a convolution layer and two fully connected layers, has an execution time that varies between 1.7s and 3s, depending on early termination. As it is not possible to ensure that each audio event of the target classes falls neatly into one second buckets, I combine the outputs of two consecutive jobs by taking their logical OR.

I use two energy harvesters: solar and RF. The solar energy harvester is affected by outdoor influences such as passing vehicles. I vary the distance between the RF transmitter and the receiver to test the applications under different levels of noise and interference.

Results. Figures 5.23(a)-(f) shows the MCU's input voltage, the cut-off voltage, the classifier's output, and deadline misses for the six applications over time. Findings from this experiments are as follows.

The car detector in Figure 5.23(a) always harvests sufficient energy from the sun and meets the

deadline for all jobs. However, it misclassifies twice when both pedestrians (talking) and cars are in the scene due to the limitations of the classifier. The dog monitor in Figure 5.23(b) experience intermittency due to people blocking the sun. It misses two target events due to the lack of sufficient energy to read the sensor data and misclassifies one event due to the limitation of the classifier. For two audio events, the applications experience deadline misses despite doing accurate classification because of the limitation of the utility test. For similar reasons, the people detector in Figure 5.23(c) fails to sense two events and misclassifies one.

The baby monitor in Figure 5.23(d), powered with an RF harvester, does not harvest enough energy to read audio samples during one audio event. It also fails to finish execution of mandatory units within the deadline for one event. Due to the limitation of the utility test, it misclassifies one event and misses deadline of another. The Laundry monitor in Figure 5.23(e) misclassifies one event and misses the deadline for two. The printer monitor in Figure 5.23(f) experiences the highest intermittence, misses four deadlines and misclassifies three events.

A number of interesting observations from these experiments are: (1) a shorter power-off period decreases the number of event misses, e.g., the solar powered dog monitor misses more events than the laundry monitor despite having less frequent reboots due to insufficient power supply; (2) a shorter continuous energy results in more deadline misses, as evident in dog monitor and printer monitor applications; (3) deadline and target event misses depend on the harvested energy and the accuracy of the utility test, whereas the classification accuracy relies on the competence of the classifier and the accuracy of the utility test, e.g., the car detector misclassifies due to the limitation of the classifier, whereas the dog monitor misses the deadline of two correctly classified samples due to the inaccuracy of the utility test.

5.8.2 Visual Sensing

Experimental Setup. I evaluate the performance of Zygarde in a multi-tasking scenario having two visual recognition tasks: traffic sign recognition and shape recognition. Both DNNs have two convolution layers and two fully-connected layers, but the convolution layers of the sign recognizer has 8 and 16 filters, whereas the convolution layers of the shape recognizer has 4 and 8 filters. The shape recognizer's execution time is about half of sign recognizer's execution time, and hence, it has a smaller relative deadline. After capturing an image, a sign detection job is created and inserted

into the job queue, followed by a shape detection job.

Figure 5.24 shows the setup for this experiment. I use a 2MP OV2640 camera sensor and capture the test images from the GTSB [241] dataset displayed on the screen of a laptop. I use 80% of the dataset for training and the remaining 20% for testing. I annotate the dataset to label the shape of the sign. I use a solar energy harvester to power the system and acquire 5V and 3V power lines for the camera and the MSP430, respectively, by using two voltage regulators. The camera module requires 4s to capture an event but works in parallel with the MSP430 which uses DMA.

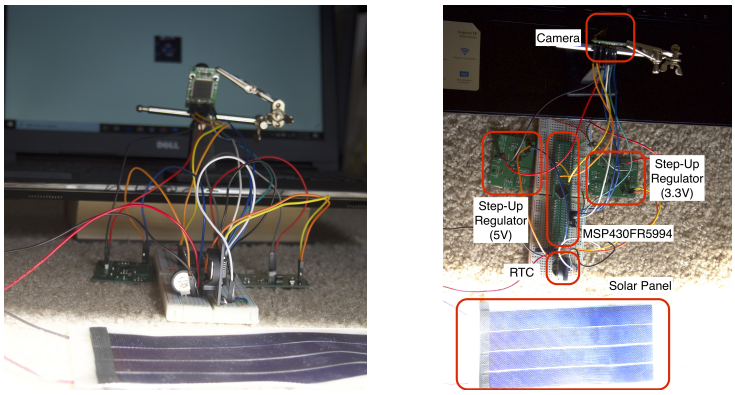


Figure 5.24: Experimental setup for visual sensing application.

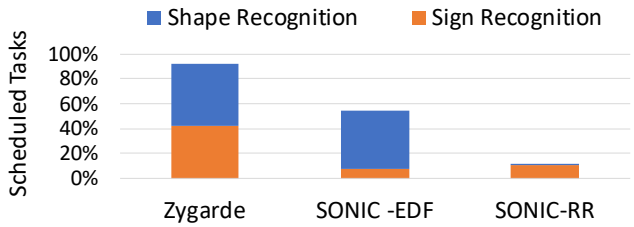


Figure 5.25: Percentage of captured events that meet the deadline.

Results. Figure 5.25 compares the performance of Zygarde against SONIC's [8], which does not implement early termination and uses either an EDF or a round-robin (RR) scheduler. I observe that due to the high energy demand of the camera, 37% of the events are missed and do not enter any of the three systems. Although SONIC-EDF schedules 55% of the jobs that enter the system, it is partial towards the shape recognition jobs since they have earlier deadlines. By choosing the sign recognition job, which has higher execution time, SONIC-RR does not spare sufficient time to execute shape recognition job. SONIC-RR schedules only 11% jobs that enter the system in total, among which, only 1% are shape recognition jobs due to the shorter relative deadline of the

shape recognition task. By performing imprecise computing with early termination, Zygarde assigns different priority to the same job at different units. Thus, Zygarde switches between jobs from different tasks and enables fairness. Zygarde schedules 93% of the jobs that enter the system, where 43% are sign recognition jobs and 50% are shape recognition jobs. Zygarde achieves 61% and 85% classification accuracy for sign and shape recognition, respectively, which is is within 2% of the baselines' that execute the DNNs end-to-end.

5.9 Discussion

5.9.1 Importance of DNNs

For batteryless sensing systems, the inference accuracy dictates the response time and the energy-efficiency of the system [8]. Due to very high energy cost of wireless communication, these systems have to implement a large capacitor – which takes several minutes to charge – in order to send just one data packet. Hence, every false positive wastes significant amount of energy and time. This is why, DNNs are preferred over less accurate traditional classifiers such as Support Vector Machines (SVM), K-Nearest Neighbours (KNN), k-means, and Random Forest. Table 5.7 shows that DNNs are 1%–15% more accurate than the traditional classifiers.

Table 5.7: Classification Accuracy for Different Models.

Classifier	**MNIST**	**ESC-10**	**CIFAR-100**	**VWW**
KNN	92% [242]	40%	55%	60%
K-means	93% [242]	41%	50%	59%
Random Forest	93% [242]	25%	29%	62%
SVM	96% [242]	50%	51%	69%
CNN (No Early Termination)	98%	75%	78%	84%
CNN (Early Termination)	97%	73%	77%	84%

5.9.2 Performance of η Factor Estimation

I estimate the η-factor offline, using energy harvesting traces of the target system. The modeling accuracy of η largely depends on the length of this empirical study, which must be long enough to capture harvested energy variability. This variability depends on the energy source, the usage by a user, and deployment configuration. Hence, prior knowledge about the system's energy usage pattern and the system designer's experience is crucial to determining a reasonable study duration to obtain an accurate estimate of the η-factor.

The change of the environment or the user behavior might change the predictability of a deployed

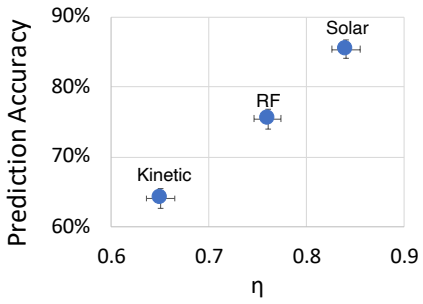


Figure 5.26: Validation of η-Factor.

system. The offline estimation of η causes errors in such scenarios. Assessment of η at runtime could address this problem by updating via online or offline reestimation process. Such assessment is possible as the system uses η to predict the harvestable energy, and the energy measurement at future time slots provides the ground truth. Thus, the system can precisely compute the prediction error at runtime. Depending on this error, η can be adapted to $\eta \pm \delta\eta$, where $\delta\eta$ is proportional to the prediction error.

However, the evaluations done in this chapter does not include adaptations as the empirically derived η values have been reasonably accurate. Here, the accuracy of estimation refers to how closely I can characterize the randomness in intermittent energy instead of accurately predicting the harvested energy. Figure 5.26 shows that the estimated value of η for three harvesters (used in Section 5.2.1) converges to their respective prediction accuracy values. For example, the kinetic energy harvester's estimated η-factor is 0.65, and its (measured) accuracy of predicting the energy state of the next slot is also close to 65%. The convergence of these two values indicates that the estimate is relatively accurate.

5.9.3 Generic Utility Functions

In Zygarde, the utility function provides an estimate of how confident the cluster-based classifier is. For a different type of classifier, although the proposed utility function may not be directly applicable, the general principle behind the utility function remains the same. For some classifiers [243], e.g., support vector machine and K nearest neighbour, the distance of the input data point from the decision boundary or the neighbours can be used to design the utility function that is similar to Zygarde's. For classifiers that provides a probability distribution over all classes as the output [243], e.g., neural networks, naïve bayes, and logistic regression, I recommend using the entropy [244] of

this distribution as the utility function, i.e., $U = -\sum_{i=1}^{c} p_i \log_2 p_i$, where p_i is the probability of the input being in class i and c is the total number of classes. A higher entropy indicates that the probability of the input belonging to some class is higher than the rest of the classes, whereas a lower entropy indicates similar probability across all classes.

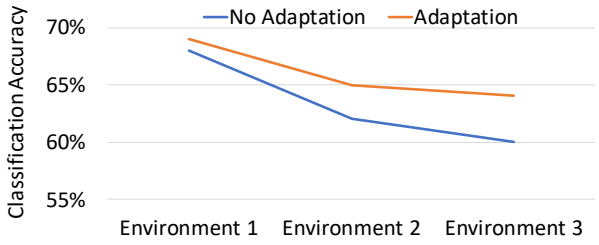


Figure 5.27: Performance gain due to adaptation.

5.9.4 Adapting the k-Means Classifiers

Zygarde adapts the cluster-based classifiers at runtime since a classifier running on a perpetually-powered system is likely to encounter shifts in the input data distribution over its extended lifetime. To enable this, I implement a simple strategy where the cluster centroids are updated by taking the weighted average of the current centroid and the new data point. Assigning more weights to the current centroid ensures that the adaptation process is gradual, and is not affected by a few outliers. This strategy has both benefits and limitations.

The benefit of cluster adaptation is that if a system is trained and tested in different environments, unless measures are taken to adapt the classifier, its accuracy drops. I conduct an experiment to quantify this. I first divide the ESC-10 audio dataset in to 80% training and 20% testing subsets. Then I record only the testing subset in three different environment, i.e., lab, hall, and office. The training subset, 80% data, is recorded only in environment 1 and is used to train the agile DNN and the initial k-means classifier. I test the accuracy of the classifier on the testing subset from environment 1 (lab), followed by testing the accuracy on the testing subset from environment 2 (hall), followed by testing the accuracy on the testing subset from environment 3 (office). I repeat this experiment with and without the cluster adaptation step of Zygarde. Figure 5.27 shows the result. Without the adaptation, Zygarde loses 8% accuracy due to the environment changes. More than half of this lost accuracy is gained back when Zygarde enabled cluster adaptation.

Two major limitations of this approach are: (1) the adaptation process being slow, if the

environment changes rapidly, the system may not be able to adapt fast enough. By adjusting the weights assigned to the new data, this problem can be addressed; (2) the proposed adaptation process is robust to only a certain types of distribution shifts, e.g., translation and rotation of feature spaces, where the relative distances of the cluster heads do not change significantly. However, if the shift in the data distribution in the new environment is complex and/or non-linear, this simple threshold-based cluster adaptation approach may not work. To deal with this, a more sophisticated approach that normalizes the effect of domain shifts in data [245, 246] has to be employed by adding an extra layer of computation prior to the clustering step.

5.9.5 Limitations of the Zygarde Scheduler

Although the scheduler in Zygarde outperforms state-of-the-art scheduling techniques, it has some limitations that need further investigations. First, the scheduler does not provide any guarantee that all the jobs will finish their mandatory part before the deadline. This is primarily due to the uncertainty of the intermittent energy which does not allow us to formally approach the scheduling problem without introducing any probabilistic terms. Besides, in this paper, I only provide a necessary condition for schedulability analysis (Section 5.4.3), while deriving a sufficient condition remains an open problem. Second, the current design of Zygarde does not schedule the wake-up cycles of the system. Instead, it reactively wakes up (and shuts down) based on the harvested energy/input voltage. Because of this, the system often misses capturing the events as it might be in power down state. By learning the event pattern and incorporating the probability of job arrivals into the scheduling framework, this problem can be addressed. Third, the queue size has a significant effect on the scheduler. Due to memory limitations, I cannot implement a longer queue (in Section 5.7, the queue size is 3). If the queue size is smaller (e.g., 1), the scheduler will only schedules the mandatory portions.

5.10 Summary

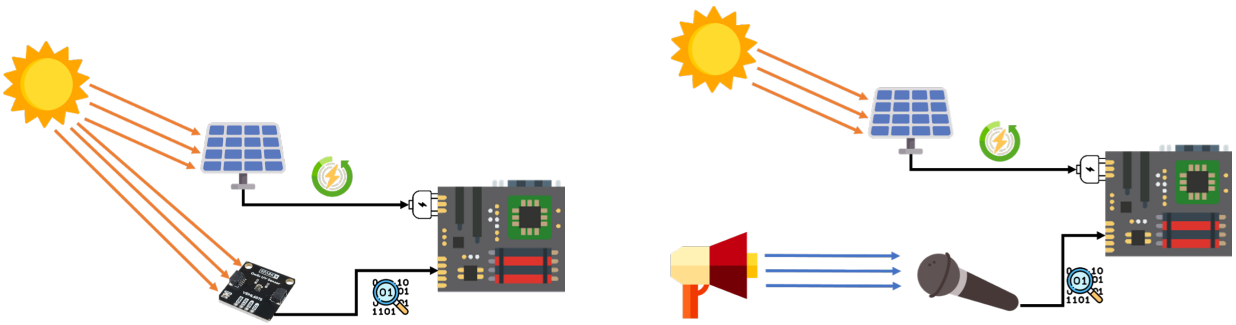
This chapter introduces a deadline-aware DNN runtime framework for intermittent systems. It devises a single metric, η-factor, that demonstrates the probability of a energy harvesting system. If eta is low, then the system is random, and high eta means the system is predictable. In summary, eta measures how close a harvester's harvesting pattern is to a constant energy source. It proposes a DNN construction and execution technique which adapts the DNN inference process at runtime, and decreases the execution time by 5%-26%. Finally, it utilizes the η factor and the adaptive execution

framework of a DNN to devise an online scheduling algorithm for batteryless systems that successfully schedules 9%-34% more tasks than traditional scheduling algorithms. I also derive a necessary condition for scheduling real-time imprecise DNN tasks on intermittently-powered systems.

CHAPTER 6

Persistent System Emulation with Distributed Intermittent System

Despite these commendable efforts to make intermittent systems suitable for time-sensitive applications, these approaches can only guarantee sensing and timely execution when the energy intermittence is not for a prolonged period. Current intermittent systems fail to sense and process target events that occur during a power failure. This drawback limits intermittent systems' potential in continuous monitoring and fault intolerant application domains, e.g., breath monitoring for respiratory diseases, which affects more than one billion people every year.



(a) (b)

Figure 6.1: Correlation between energy source and data source. (a) The source of data and energy is the same, (b) The sources of data and energy are different.

Two primary factors make it challenging for the intermittent systems to guarantee target event sensing and timely execution. The first one is the dynamic and uncontrollable nature of most harvestable energy sources. To reduce this uncertainty, Chapter 5 and Chapter 4 predicts the energy source's predictability and schedules tasks to ensure that the maximum amount of tasks finishes within the deadline. Other works like ACES [52] use reinforcement learning to determine the optimal duty cycle based on the source's harvestable energy. However, none of these works can guarantee target event capture or timely execution when the intermittence duration is high.

A batteryless computing system can effectively learn and classify physical world events when the presence of energy implies (mathematical implication) the presence of data. Though target

events only occur when energy is present, is true for scenarios where the source of energy and source of target event is the same (e.g., solar-power UV ray monitor), this becomes false when the source of energy and target event is different (e.g., solar-powered car detector). The first scenario is comparatively common in passive event detection systems where an external signal wakes up the system and triggers it to start sensing and computing. Most of the existing works on event detection using an energy harvesting system uses such passive event detectors. For example, when a solar-powered device is used to observe the UV emission of the sun, the sun is the source of both power and data, as shown in Figure 6.1a. On the other hand, the second scenario requires active event detection, where the system uses a pooling system to wake itself up to sense and detect the event. Many applications, including air quality monitoring, noise detection, preventive machine maintenance, require active sensing to capture the target system. For instance, Figure 6.1b uses solar energy to analyze an acoustic environment. Here, the source of data and the source of energy are different, but data (sound) is always present, though the energy source (sun) is not. If the active system's wake-up period and event occurrence are not synchronized, the system missed the target event. However, the occurrence of the target event is non-periodic and hard to predict. As none of the existing work takes the event's sporadic nature into account, despite all the efforts, these systems fail to capture all the target events for computation when the energy source and event sources are independent.

To address this limitation, in this chapter, I propose Falinks, which is the first framework that uses a swarm of intermittent nodes to prolong the collective power-on period without communication. Falinks allows each intermittent node of the distributed swarm of intermittent nodes to predict other nodes' environment and behavior without communicating with each other and decide whether to wake up or go to sleep. Falinks aims to mimic a persistently powered node's performance by ensuring that at least one intermittent node is active at any point in time. As a result, it avoids missing any target event without performing any prediction about the event occurrence. Through Falinks, I make four technical contributions:

- First, I propose scheduling algorithms that enable distributed intermittent nodes to emulate a persistently powered node without any communication collaboratively. I propose a Duty-Cycle algorithm when all the swarm nodes get the same energy from a constant energy source. I also

derive the minimum number of required nodes for such a system with proof.

- Second, I devise the Prime-Coprime algorithm that imitates a persistent node when each swarm node gets different energy from a constant energy source. Moreover, I propose a minimal overhead rule-based algorithm (Prior Knowledge algorithm) for scenarios where the different energy levels at each node are known a priori to other nodes. I prove the optimality of the proposed algorithm.

- Third, I formulate variable energy source scenarios as a Partially Observable Markov Decision Process (POMDP). I further propose different heuristics for updating the states with the varying source. I further provide theoretical proofs behind the proposed algorithms.

- Finally, I design a controllable cascading capacitor array where the microcontroller controls each capacitor's charging and discharging. This capacitor array allows each intermittent node to borrow harvested energy from previous charging cycles and eliminates the mutual exclusiveness when the available energy to harvest is smaller than the consumed energy. By doing this, the system harvests maximum energy at any condition.

To evaluate Falinks, I identify three significant performance metrics – schedulability, redundancy, percentage of inactive time, and mean time to non-ovservability. I collect real-life energy traces from solar and RF, event occurrence traces from passing vehicles, and event detection workload traces from a deep neural network-based acoustic event detector running on an MSP430FR5994. Besides executing acoustic sensing and event detection, this microcontroller also executes system tasks, e.g., maintaining clocks and monitoring energy.

I compare FALINKS with a greedy approach and a reinforcement learning-based approach, ACES, that determines the duty cycle for waking up in simulation and trace-based experiments. FALINKS, on average, achieves 58.50% more schedulability than a single intermittent node, 54.40% more schedulability than a greedy swarm of intermittent nodes, and 35.73% more schedulability than a swarm of nodes where each node practices ACES. Though nodes practicing ACES performs better than greedy nodes, it experiences 1.4–12.59 seconds more mean time to non-observability. In real-world scenarios, Falinks achieves 41.17% higher schedulability and 69.7% lower redundancy than a swarm of greedy intermittent nodes.

6.1 Motivation

6.2 Problem Formulation

This section first states the goal of Falinks and then it describes the challenges that hinder the fulfillment of the said goal. Next it establishes different cases based on the different types of energy sources and target events. Finally, it provides the problem statement and assumptions for designing and developing Falinks.

6.2.1 Goal and Challenges

Goal. Our goal is to *develop an intermittent system with similar sensing and on-device computing capabilities of persistently-powered systems and replace battery-powered systems with batteryless systems.*

I identify and describe two most significant challenges for achieving this goal – – (1) high energy consumption and time-synchronization issues of communication, and (2) limitation of capacitor charging.

Communication Challenge. For a swarm of intermittent nodes to operate collaboratively, either a central node or all nodes require to know the harvestable energy status and current action (sleep or awake) of the other nodes to decide for everyone or themselves. The nodes can communicate either with active radio or passive radio, e.g., backscatter [247], to share this knowledge. However, communicating with active radio consumes high energy [8], increasing the energy overhead, and is unsuitable for such frequent knowledge transfer. Moreover, as the intermittent nodes cannot keep the active radio turned on, they require performing duty cycling. Such duty-cycle communication requires precise time synchronization, ranging from nanosecond to millisecond precision [248]. Such precise time synchronization is hard to achieve in intermittent systems despite recent efforts [249]. This obstacle is more significant for intermittent systems as they stay awake for a short time.

One solution might be to use passive backscatter radios, which do not require any energy to communicate. First, let us consider the scenario using the centralized approach where a single intermittent node gains the energy and activity knowledge from the other nodes decides for them. Though intermittent nodes may transmit their status with backscatter, the central node needs to have an active radio to receive the information, and it is unsuitable for an intermittent node. Moreover, each intermittent node also needs to listen to the individual decided action from the

central node, which is not suitable with backscatter radios. For similar reasons, taking a distributed approach where all nodes share their status and decide their action is not viable. Besides, conflict during wireless packet transmission is another major issue for such a system.

To avoid these issues, I aim to develop a framework where the intermittent nodes do not directly share their activity or energy conditions. Instead, they predict it from prior knowledge and environment.

Capacitor Charging Limitation. Capacitor size plays a crucial role in the performance of an intermittent system. Though smaller capacitors are more responsive than larger ones, they are often not sufficient for computationally expensive tasks, e.g., capturing audio, processing data, executing inference, and wireless data transmission. On the other hand, though larger capacitors can fulfill such emergy demands, they require a longer time to reach the operating voltage and thus are less responsive. Moreover, smaller capacitors also experience saturation where the capacitor is already full and can not harvest available energy. To balance between capacity and responsiveness while determining capacitor size reconfigurable capacitor array has been proposed. This capacitor array reconfigures the storage capacitance based on the energy demand of the next task. However, it does not consider (1) how to increase the energy storage even when there is abundant harvestable energy, and (2) the mutual exclusiveness of charging and discharging the capacitor when the harvestable energy is less than the consumed energy. The proposed storage architecture is more suitable for greedy approaches, which discharges immediately after harvesting sufficient energy. As a result, it wastes potential harvestable energy.

Inspired by reconfigurable capacitor storage and cascading remanence timekeeping system, I aim to develop a software-controlled power management system using a capacitor array that can charge the remaining capacitor while draining a subset of capacitors.

6.2.2 Considered Energy Sources and Target Events

Target Energy Sources. Based on the different categories of energy sources described in Chapter 3, I identify four cases – constant & balanced energy source, variable & balanced energy source, variable & balanced energy source, and variable & unbalanced energy source.

Target Events. I classify target event into two types based on the accessibility of the event to the nodes in the swarm – global event and local event. When an event is global, all the nodes in the

swarm can sense the event. For example, a gunshot is loud enough to be heard by all the sensors in a house. On the contrary, if only a subset of active nodes can sense an event, it is a local event. For example, only nodes near a person can sense his/her irregular breathing.

Note that whether an event is global or local depends not only on the event but also on the node's placement or range. To illustrate, if the sensor nodes are placed in different neighborhoods, then the gunshot in a neighborhood cannot be heard by the sensors in other neighborhoods, and in this case, a gunshot is a local event.

6.2.3 Problem Statement and Assumption

Problem Statement. Given N intermittent nodes, how to schedule the duty cycle of the nodes, such that at least one intermittent node is present at any point in time. Here, N is the optimal number of intermittent nodes needed to satisfy this constraint.

Assumptions.

•*A1.* The target event is global, which means that all the intermittent nodes can sense and process the target event.

•*A2.* The position (location and orientation) of the intermittent nodes are predetermined. At each position, only a limited number of nodes can be placed. The assumption is that there can be only one node at any location due to the physical constraints.

•*A3.* No communication is available among the intermittent nodes.

6.3 Scheduling Algorithms for Collaborative Intermittent Nodes without Communication

This section, first, provides an optimal solution to solve the problem mentioned above. Then, it describes the algorithms with the necessary proofs for each of those cases. These algorithms satisfy all the assumptions mentioned in Section 6.2.3.

6.3.1 Falinks Optimal Algorithm

For deducing the optimal algorithm, I assume that each intermittent node knows the harvestable energy for all other intermittent nodes as prior knowledge or perfect estimation for the optimal solution. The estimation process may include exploration of physical phenomenon (e.g., path loss models) and known or predictive energy source variation. Each node utilizes this knowledge and calculates the action of the other nodes to decide its action. Each node dynamically sleeps or wakes

up using rule-based decision making. The rule is as following – *the node with the maximum total harvestable and stored (in the energy storage) energy is awake at any point in time.*

Theorem 1. *If the harvestable energy for all other nodes is known, only waking up the node with the maximum total harvestable and stored (in the energy storage) energy is optimal.*

Proof of Theorem 1. I proof this using contradiction. I assume that intermittent node I_i ($i = 1, 2, ..., N$) has total energy $E_1 = E_{H_1} + E_{C_1}$, where E_{H_i} and E_{C_i} are the harvestable energy and energy stored for the i^{th} intermittent node. Let I_m be the node with the maximum total energy or $E_m = max(E_i)$, for $\forall i \in N$. Let us assume that waking only I_m up is not optimal. Thus there is another node I_p which is optimal to wake up where $E_{min} < E_p < E_m$, where E_{min} is the energy consumption rate.

$E_m > E_p$ can occur is two ways – either $E_{H_m} > E_{H_p}$ or $E_{C_m} > E_{C_p}$. If $E_{H_m} > E_{H_p}$ and only I_p wakes up while all the other node asleep, $E_p{}' = E_p - E_{min} + E_{H_p}$ and $E_m{}' = E_m + E_{H_m}$. As $E_{H_m} > E_{H_p}$, $E_m{}' > E_p{}'$, E_m will keep increasing and will reach the maximum capacity of the energy storage. Then, I_m will fail to harvest available harvestable energy and as a result it will waste potential energy. Similarly, when $E_{C_m} > E_{C_p}$, the energy storage of I_m will fill up quicker resulting in wastage of harvestable energy. As wastage of potential energy is never optimal, waking up I_p is not optimal. This contradicts our assumption and prove that waking I_m is optimal. □

As prior knowledge or perfect estimation of the harvestable energy at other nodes is realistic, calculating all other nodes' decision is computationally expensive. Therefore, this optimal solution is not feasible.

6.3.2 Case 1: Constant and Balanced Energy Source

Before digging into the problem, I start with the most trivial case where all intermittently powered nodes have the same energy harvestable energy at all the time. In other words, the energy is constant and balanced. To achieve the goal of having at least one active node at any given point in time, I propose using a *Falinks Duty-Cycle* algorithm.

Falinks Duty-Cycle Algorithm. If t_h is the harvesting time and t_e is the execution time, the duty cycle of the nodes are $t_e + t_h$ (if t_h is divisible by t_e) or $t_e + t_h + 1$ (if t_h is not divisible by t_e). The start time of the nodes is determined by an offset: $(n-1)t_e$, here n is the number of node.

Theorem 2. *When the energy source is constant and balanced, at least n nodes are sufficient to*

have at least one node active at any time, where $n = \lceil \frac{t_e}{t_h} \rceil$.

Proof of Theorem 2. I proof this theorem using induction.

Null Hypothesis: $n = 1$ is possible, if and only if $t_h \geq t_e$. When $t_h \geq t_e$, the each node always have sufficient energy and never go through power of period. As a result, a node will always be active in this condition and having one node is sufficient.

Induction Hypothesis: Let us assume that when $n = k$, there exist sufficient number of nodes to have at least one active node at any time.

Inductive Step: Let us assume that when $n = k + 1$, the n nodes are not sufficient to have at least one active node. The value of n increases when either t_e increases by at most t_h units or t_h decreases by at least t_e units. As the task executed by a node is fixed, t_e is constant. There n increase only when t_h decreases by at least t_e units and it introduces t_h amount of time when no node have sufficient energy to be active. As $t_h < t_e$, one node can execute for t_e time and can ensure that a node is active at any time. Therefore, as $n = k$ nodes are sufficient, $n = k + 1$ nodes are also sufficient.

Conclusion: Since both the base case and the inductive step have been proved as true, by mathematical induction $n = \lceil \frac{t_e}{t_h} \rceil$ are sufficient to have at least one node active at any time. □

6.3.3 Case 2: Constant and Unbalanced Energy Source

Similar to the previous case, in this case, the harvestable energy is constant at any time. However, harvestable energy is unbalanced, and thus the amount of harvestable energy at each node varies. This case needs to consider the lowest available harvestable energy as the harvestable energy for all nodes to apply the *Falinks Duty-Cycle* algorithm. However, it is not efficient as I will require a higher number of intermittent nodes.

Theorem 3. *When the energy source is constant but unbalanced,* Falinks Duty-Cycle *algorithm is not optimal.*

Proof of Theorem 3. I proof this theorem using contradiction. Let us assume that the Falinks Duty-Cycle algorithm is optimal when the energy source is constant but unbalanced. In the Falinks Duty-Cycle algorithm for unbalanced harvester, using the lowest value to calculate the duty cycle will ensure that at least one node is active. Taking the highest or average value is not suitable due to the lack of guarantee. However, the nodes with higher harvestable energy will waste energy as

their energy storage will get charged faster. If these nodes have small duty-cycles, the total number of nodes required to ensure one active node at any time can be reduced. It contradicts our earlier assumption that the Falinks Duty-Cycle algorithm is optimal. Thus, when the harvestable energy is constant but unbalanced, the Falinks Duty-Cycle algorithm is not optimal. □

Falinks Prime-CoPrime Algorithm. To address this, instead of using the same duty-cycle for all nodes, I propose a new algorithm that provides the duty cycle based on the prime and co-prime numbers. I name it the *Falinks Prime-CoPrime* algorithm. This algorithm is optimal and can ensure that at least one node is active at any time using the minimum number of nodes.

1. Take all the prime numbers $P = \{p_1, p_2, ..., p_i\}$ where $\forall i \in R$, $T_0 \leq p_i \leq T_H$. Here, T_0 is the lowest duty cycle possible at any location, and T_H is the hyperperiod.

2. Use the Sieve of Eratosthenes to determine the rest of the duty-cycles larger than T_0 and are not divisible by P.

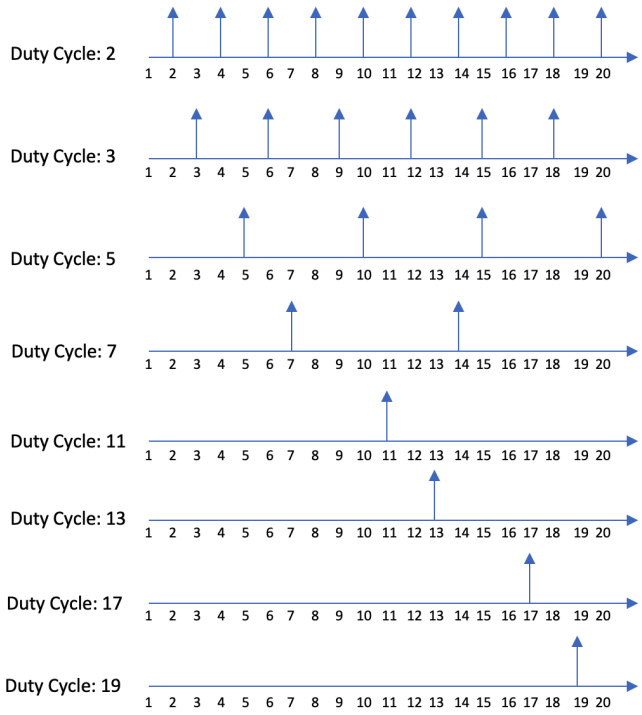


Figure 6.2: Falinks Prime-CoPrime algorithm with duty cycle of prime numbers. Here, the lowest duty-cycle is 2, the hyperperiod is 20.

Theorem 4. *When the energy source is constant and unbalanced, the minimum number of inter-*

mittent nodes required to ensure that at least one intermittent node is present at any time-instance within a hyperperiod, T, can be given by total number of primes smaller than or equals to T. Here, the smallest allowed duty-cycle is 2.

Proof of Theorem 4. Lets assume that for every prime number smaller than or equals to hyperperiod (T), if there exists an intermittent node with prime duty cycle, then not all the time instances does not have an active node.

Each number smaller than or equal to T can either be a prime or non-prime. Moreover, as the nodes are co-prime each duty cycle will be unique. If the time instance represents a prime number then there is an intermittent node with prime duty cycle and thus exactly one node will wake up at those point.

Next, I observe the case where the time instance represents a non-prime number. For a non-prime number, $q = mn$ must be true, where $1 < m, n < q$. By induction, as m, n are smaller than q, they must each be a product of primes. Therefore, q is also a product of prime. Therefore the there will be p actives nodes at q time where p is the number of unique primes factors of q. For example, if the time instance is 12, there will be 2 active nodes having duty-cycle 2 and 3.

This contradicts our assumption. Thus, there is at least one active nodes in hyperperiod T, if N nodes are present where N is the number of primes less than equals to T and each node's duty cycle is a prime number smaller than or equals to T and they are co-prime. □

6.3.4 Case 3: Variable and Equal Energy Source

As the harvestable energy at any node is equal in this scenario, each node knows precisely the other node's harvestable energy. Thus, this case can use the optimal algorithm described in Section 6.3.1.

6.3.5 Case 4: Variable and Unequal Energy Source

In this scenario, the harvestable energy varies with time and location. To address this, I determine all the permutation of the duty-cycles from the Falinks Prime-CoPrime Algorithm. Every node start the process from the same index number of the list. However, depending on the changing harvestable energy they change the index.

A Dec-POMDP is a tuple $\{N, S, A, T, R, \Omega, O, h, b_0\}$, where

- $N = \{1, .., n\}$ is the set of n agents,

- S is the finite set of states s,
- A is the set of joint actions $a = \{a_1, .., a_n\}$,
- T is the transition function that specifies $Pr(s_{t+1}|s_t, a_t)$,
- $R(s, a)$ is the immediate reward function,
- Ω is the set of joint observations $o = \{o_1, ..., o_n\}$,
- O is the observation function: $Pr(o_{t+1}|a_t, s_{t+1})$,
- h is the horizon of the problem,
- $b \in \Delta(S)$, is the initial state distribution at time $t = 0$.

Dec-POMDP aims to find an optimal joint policy $\pi*$ that maximizes the expected sum (over time-step) of rewards. I do not use the discounted summation of the rewards as performing a task sooner does not benefit our goal. Instead, having at least one intermittent node active all the time is more critical. The significant difference between multiagent MDP and Dec-POMDP frameworks is that the joint policy is decentralized. $\pi*$ is a tuple $\{\pi_i, ..., pi_n\}$ where the individual policy π_i of every agent i maps individual observations histories $o_{i,t} = \{o_{i,1}, ..., o_{i,t}$ to action $\pi_i(o_{i,t} = a_{i,t})$.

Though this is the most optimal method, this DEC-POMDP is a *NEXP-complete* problem that is not suitable for an intermittent system. Therefore, I provide different suboptimal lightweight heuristic to update the state diagram.

To develop our heuristics, I determine all the permutations of the duty-cycles from the Falinks Prime-CoPrime algorithm. Every node starts the process from the same index number of the list. However, depending on the changing harvestable energy, they change the index. Following are some heuristics to select the next index.

Ideal Index Selection. All nodes choose the same index of the list. For a known or predictable system, this can be achieved using a modified prior knowledge algorithm.

Random Index Selection. When a node experiences changing harvestable energy, it will randomly choose a higher or lower index achievable by the harvestable energy. I perform this selection using a variant of the binary search method where instead of selecting the middle point, selects a random point in the range.

Incremental Index Selection. In this method, using the local harvestable energy, the node chooses the immediately higher or lower index.

Suboptimal Reinforcement Learning. In this method, each node tracks the change in the harvestable energy and takes the benefit of the prior knowledge about how the energy change across locations are related. This prior knowledge will come as an offline state diagram denoting how indexes can be changed based on energy. Each node will use the same state diagram to select the next index. When the environment can provide feedback based on a node's actuation, the node exploits it to update the offline state diagram. This algorithm is more suitable for scenarios where the system monitors a continuous variable and failing to have any node awake at any time slot results in a change in that variable. In such a scenario, each node will have its private Q-Table with its energy status as a state and sleep or wake up as action. Each node takes decisions from the Q-table and updates the table using the feedback from the continuous variable. Note that this is a suboptimal solution.

6.4 Software Controlled Cascading Capacitor Array

The cascading capacitor array has a switch between the harvester and each supercapacitor of the capacitor array for controlling the supercapacitor charging cycle. It also has a switch between each capacitor and the processing unit to control the supercapacitor discharging cycle. With these switches, the system controls which supercapacitors to charge and discharge at any point in time and ensure that charging (harvesting) and discharging (computing) can coincide when energy allows. The microcontrollers use GPIO pins to control the switches.

This cascading capacitor array has two types of switches – default-On and default-Off. The default-On switch is initially on to allow the circuit to work from a cold start by enabling at least one charging capacitor when the microcontroller fails to power up. This switch uses a P-Channel Metal Oxide Silicon Field Effect Transistor (MOSFET) [250] that activates immediately without any control signal from the microcontroller. The microcontroller can deactivate the p-MOSFET with digital signals from the GPIO pin.

The default-Off switch follows a similar design on SmartOn [251]. This switch disconnects the supercapacitor from the harvester or the processor unit when the system restarts. Only the microcontroller can activate these switches using the GPIO pins.

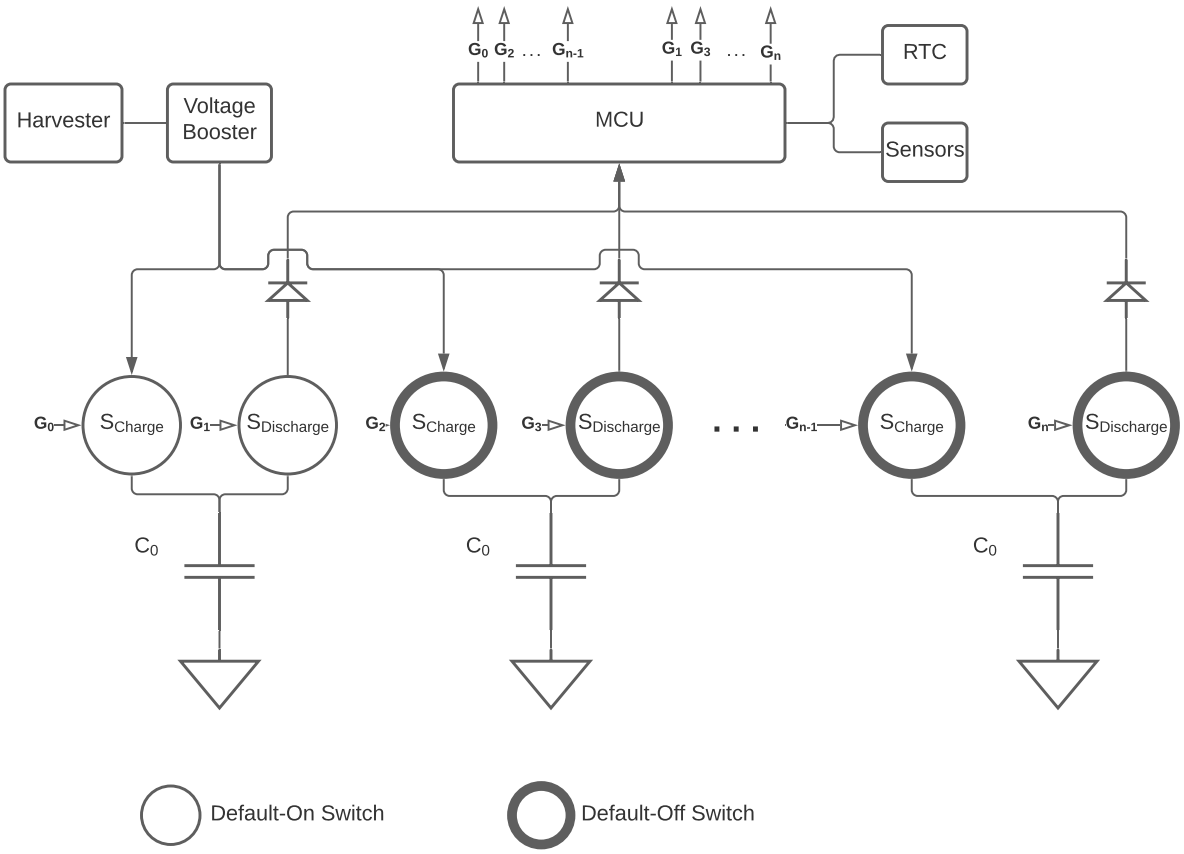


Figure 6.3: System with Software Controlled Cascading Capacitor Array.

6.5 Simulation-Based Evaluation

This section first defines the baseline algorithms and performance metrics. Then, it describes the synthetic dataset. Next, it compares Falniks algorithms' performance against baseline algorithms using the described synthetic taskset and harvestable energy patterns. For each scenario, this evaluation reports the average of 1,000 iterations for different time duration – one day, one week, two weeks, and one month.

6.5.1 Baseline Algorithms

Two baseline scheduling algorithms are – greedy scheduling algorithm and automatic configuration of energy harvesting sensors (ACES) [52] algorithm. This evaluation considers two different variants of each of these algorithms – a single intermittent node and a swarm of intermittent nodes. In the second scenario, each code executes a local scheduling algorithm due to the lack of communication. To preserve the evaluation's integrity, every algorithm with a swarm of nodes has the same number

of member nodes.

Greedy Scheduling Algorithm. In the greedy scheduling algorithm, the node wakes up whenever it accumulates enough energy to power the microcontroller. This intermittent node then executes as long as sufficient energy is present in the energy storage. Then the node goes to sleep or low-power mode until the energy storage stores the required energy.

Automatic Configuration of Energy Harvesting Sensors (ACES). Automatic Configuration of Energy Harvesting Sensors (ACES) [52] uses reinforcement learning to maximize each intermittent nodes sensing performance. Using Q-learning at each node for determining their duty-cycle. In this method, each node chooses between four duty-cycle periods – 15 seconds, 1 minute, 5 minutes, and 15 minutes. Every 15 minutes, the reinforcement algorithm observes and changes the duty-cycle if needed.

6.5.2 Performance Metrics

The performance of Falinks depends not only on the number of sensed events but also on the number of computed events within the deadline. Besides, having multiple nodes active at the same time wastes resources. Thus, I propose four different performance metrics to compare Falinks against the baseline algorithms – (1) schedulability, (2) redundancy percentage, (3) redundancy degree, and (4) mean time to non-observability.

Schedulability. In real-time systems, schedulability is a scheduling algorithm's ability to schedule all the tasks in the taskset. As the performance also depends on the number of sensed events, I redefine *schedulability* as follows– *Given a scheduling algorithm that schedules wake-up and sleep of intermittent sensor nodes, if all energy and event combinations can be sensed and inferred, I achieve 100% schedulability.* Though I desire 100% schedulability, if a scheduler can sense and infer 50% of the combined set events, the scheduler has 50% schedulability.

Redundancy Percentage. To optimally simulate a persistently powered system with multiple intermittently powered system, only one intermittent node needs to be active at any point in time. For designing an optimal system, I also want to measure the percentage of time more than one node was active. I define *redundancy degree* as the percentage of active time more than one nodes are active.

Redundancy Degree. Along with the percentage of redundancy, the performance metric includes the degree of redundancy, demonstrating a system's efficiency. The *redundancy degree* is the maximum number of nodes active at the same time within a duration. Here, the considered durations are – one day, one week, two weeks, and one month.

Mean Time to Non-observability. The final performance metric for assessing Falinks is mean time to non-observability. Non-observability means that the intermittent node was not able to sense any event due to power failure. Mean time to non-observability denotes the average value of the continuous non-observability time or the continuous power-off time.

6.5.3 Source, Event, and Taskset

Synthetic Energy Source. For evaluating the performance of Falinks, this section considers three types of energy sources – constant and balanced, constant and unbalanced, and variable and unbalanced. Despite defining four types of energy sources in Section 6.2.2, this evaluation does not consider the variable and balanced energy sources because it is similar to the constant and unbalanced with prior knowledge. For all the sources, the randomly selected harvestable energy ensures the presence of three cases –

1. when the harvestable energy is greater than the energy consumption rate of the task,
2. when the harvestable energy is smaller than the energy consumption rate of the task, and
3. the harvestable energy is equal to the energy consumption rate of the task.

For variable energy sources moves in different patterned paths in different speed. As a result, the harvestable energy varies for different nodes based on various path loss models.

Synthetic Events. The synthetic event dataset contains 1,000 randomly generated sporadic events. The maximum allowed period, the minimum allowed period, the maximum duration of an event, and the event's minimum duration are the input to the random event generator. After getting the input, the random event generator generates events with a random period, which is the minimum difference between the beginning of two consecutive events and random event duration. The randomly chosen event duration is always at least equal to the period to avoid the occurrence of two events simultaneously. This generator also provides each event's start time by adding a random variable

between zero and the chosen period with the chosen period to introduce sporadicity. Note that all events are considered global, and thus all the intermittent nodes can sense them.

Synthetic Computing Tasks. All intermittent nodes in the swarm perform the same task when it captures the event. The synthetic computing task dataset consists of 1,000 randomly generated computing tasks. This random task generator determines the random energy consumption rate and random execution time at every 1,000 iterations. During one iteration, this value is constant and the same for all the intermittent nodes. The random execution time is between one second and the period. The period is the upper bound as the execution time can not be greater than the period in an implicit deadline system. The random task generator randomly chooses one of the three levels of predefined energy consumption rates. The first two levels correspond to the two levels of power consumption of an MSP430FR5994 microcontroller in its active mode. Additional sensors consume more energy to operate; hence, it adds the third level to support these sensors' activation.

6.5.4 Performance of Falinks with Constant and Balanced Energy Source

Figure 6.4 shows the schedulability of different scheduling algorithms when the energy source is constant and balanced. In other words, the harvestable energy at each intermittent node of the swarm is the same and does not vary with time. The x-axis of Figure 6.4 represents the number of days, and the y-axis represents the schedulability (defined in Section 6.5.2). I observe that the schedulability of a single intermittent node and a swarm of intermittent nodes executing the greedy algorithm have similar performance. As the nodes have no notion of collective goal in the greedy algorithm, multiple nodes fail to contribute to the performance. Moreover, Table 6.1 shows that the redundancy percentage and degree of intermittent nodes' swarm with the greedy algorithm are high. Thus more nodes are active at the same time and consuming unnecessary energy due to lack of collaboration.

Table 6.1: Redundancy percentage & degree and mean time to non-observability for constant and balanced energy sources.

	Redundancy Percentage	Redundancy Degree	Mean Time to Non-Observability
Greedy Single	0.00%	1	2.48 s
Greedy Swarm	96.31%	5	2.31s
ACES Single	0.00%	1	19.07 s
ACES Swarm	38.05%	3	17.36 s
Falinks Optimal	0.00%	1	0.13 s
Falinks Duty-Cycle	0.00%	1	0.17 s

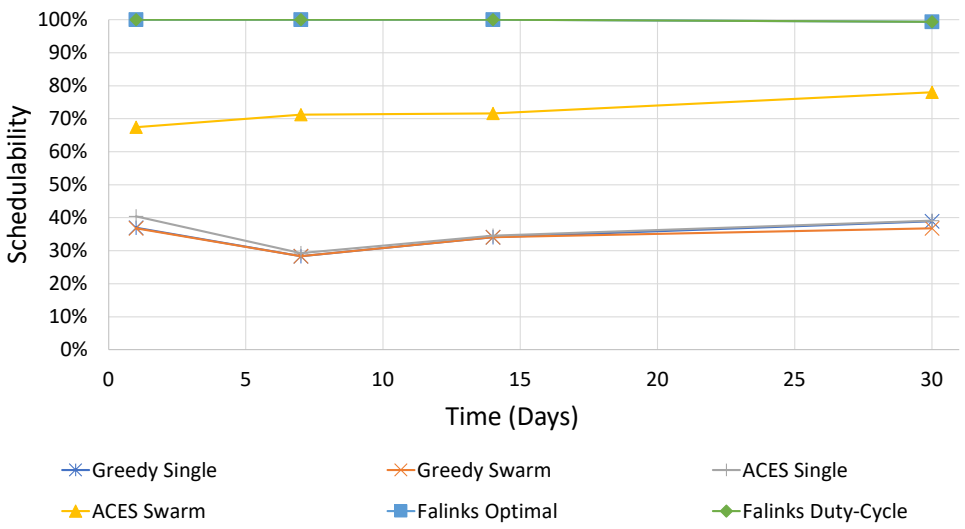


Figure 6.4: Schedulability comparison for constant and balanced energy source.

On the other hand, though a single node with the ACES algorithm has similar performance, Figure 6.4 shows that a swarm of intermittent nodes executing ACES has higher schedulability as each node updates their action (duty-cycle) using reinforcement learning. The schedulability also increases with time as the reinforcement learner learns. However, the mean time to non-observability is high when executing the ACES algorithm because of the predefined duty cycles. Finally, the swarm of intermittent nodes with Falinks Optimal and Falinks Duty-Cycle algorithms has 99.97% schedulability, where some of the loss of schedulability happens due to the failure in processing all the sensed tasks rather than missing the events. They also demonstrate 0% redundancy and minimal mean time to non-observability, as shown in Table 6.1.

Table 6.2: Redundancy percentage & degree and mean time to non-observability for constant and unbalanced energy sources.

	Redundancy Percentage	Redundancy Degree	Mean Time to Non-Observability
Greedy Single	0.00%	1	1.48 s
Greedy Swarm	96.71%	5	1.31s
ACES Single	0.00%	1	13.07 s
ACES Swarm	38.04%	4	12.75 s
Falinks Optimal	0.00%	1	0 s
Falinks Duty-Cycle	0.00%	1	1.58 s
Falinks PrimeCoPrime	5.45%	3	0.17 s

6.5.5 Performance of Falinks with Constant and Unbalanced Energy Source

Figure 6.5 shows the schedulability of different scheduling algorithms when the energy source is constant and unbalanced. Along with the scheduling algorithms shown in Figure 6.4, Figure 6.5

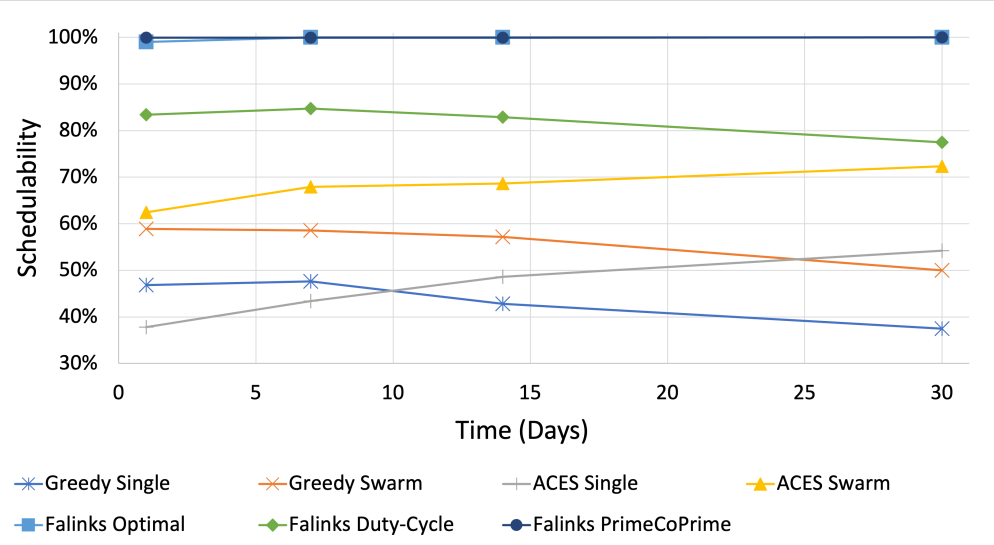


Figure 6.5: Schedulability comparison for constant and unbalanced energy source.

shows the Falinks PrimeCoPrime algorithm's schedulability. The increase of schedulability of nodes with ACES is significantly higher in Figure 6.5 than in Figure 6.4. This increase happens because each node learns its harvestable energy and chooses its duty cycle, which might differ. Though the Falinks Duty-Cycle's redundancy is 0% in Table 6.2, the schedulability decreases as each node's harvestable energy varies. Note that every system has the same number of nodes (besides the single cases) determined by the Falinks Optimal scheduler for a fair comparison. The Falinks Duty-Cycle algorithm can achieve optimal schedulability if it considers the minimum available harvestable energy to determine the duty-cycle and number of nodes at the cost of a higher number of nodes. The Falinks PrimeCoPrime algorithm achieves 1% less schedulability and 0.17 seconds more mean time to non-observability than Falinks Optimal on average due to insufficient time to execute a task in nodes with lower duty-cycle. The Falinks PrimeCoPrime algorithm has higher redundancy percentage and degree as several numbers have multiple prime numbers as factors.

6.5.6 Performance of Falinks with Variable and Unbalanced Energy Source

Figure 6.6 shows the schedulability of different systems executing greedy, ACES, Falinks Optimal, Falinks Duty-Cycle, Falinks PrimeCoPrime, and three online variants of Falinks PrimeCoPrime algorithms. These online variants are – Random Index Selection for Falinks PrimeCoPrime (Falinks PrimeCoPrime Random), Incremental Index selection for Falinks PrimeCoPrime (Falinks PrimeCoPrime Incremental), and Suboptimal Reinforcement Learning based Index Selection for Falinks

PrimeCoPrime (Falinks PrimeCoPrime RL). In Figure 6.6, the schedulability of the Falinks Optimal algorithm decreases compared to the previous scenarios. The unknown variation of the harvestable energy causes jobs to miss their deadline, resulting in this decrement in schedulability. This unknown variation also causes energy scarcity, resulting in the increment in the mean time to non-observability of Falinks Optimal in Table 6.3.

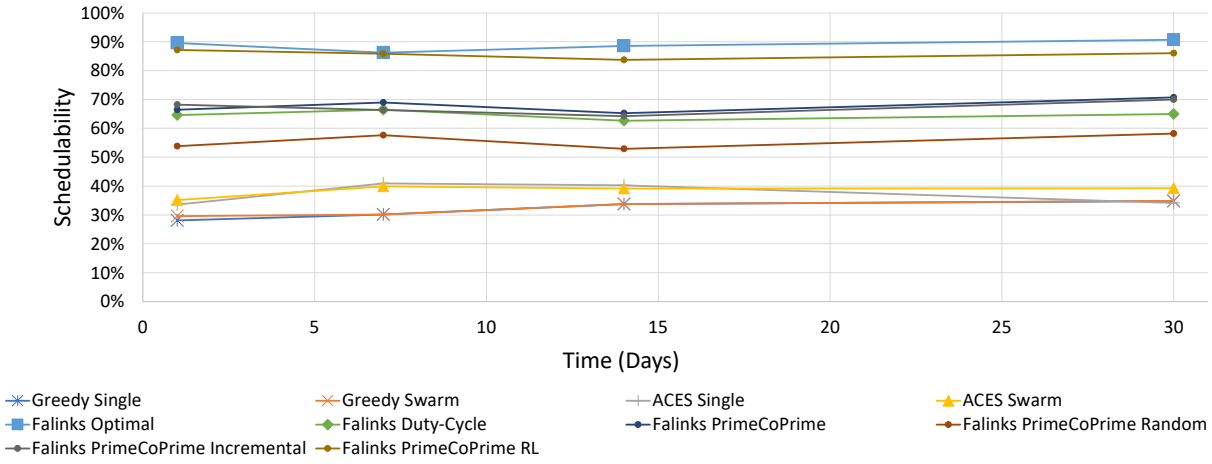


Figure 6.6: Schedulability comparison for constant and unbalanced energy source.

Table 6.3: Redundancy percentage & degree and mean time to non-observability for variable and unbalanced energy sources.

	Redundancy Percentage	Redundancy Degree	Mean Time to Non-Observability
Greedy Single	0.00%	1	1.36 s
Greedy Swarm	70.53%	7	1.29s
ACES Single	0.00%	1	11.01 s
ACES Swarm	40.80%	4	13.88 s
Falinks Optimal	0.00%	1	1.2 s
Falinks Duty-Cycle	0.00%	1	2.45 s
Falinks PrimeCoPrime	5.49%	3	1.81 s
Falinks PrimeCoPrime Random	21.71%	4	2.78 s
Falinks PrimeCoPrime Incremental	23.00%	3	1.96 s
Falinks PrimeCoPrime RL	6.41%	3	1.91 s

As the harvestable energy varies for each data point, Falinks PrimeCoPrime does not achieve similar schedulability as the Falinks Optimal, as shown in Figure 6.6. However, when each node independently performs random index selection to accommodate this changing harvestable energy, the schedulability decreases by 12.21% on average, and the redundancy degree increases to 4. Though Incremental changes of the indexes introduce minimal performance, Falinks PrimeCoPrime RL

achieves only 3% less schedulability, similar to the Falinks Optimal on average. Using the prior knowledge about the relation of the harvestable energy among the nodes and continuously learning from the feedback, Falinks PrimeCoPrime RL achieves this significant improvement. Despite having a higher redundancy degree than Falinks Optimal and 45% more mean time to non-observability, Falins PrimeCoPrime RL shows redundancy only 6.41% of the time.

6.6 Real world Evaluation

This section evaluates the performance of Falinks in a real world setting using energy and event traces from real datasets.

6.6.1 Experimental Setup

Baseline Algorithms and Performance Metric. I use the same baseline algorithms and performance metric as Section 6.5.2. However, this section does not include the Falinks Prior algorithm for evaluation as it requires power-hungry communication, which will result in lower schedulability in real-world scenarios.

Source. The solar energy trace from the Section 4.6.1 of Chapter 4 is reused as the source energy trace in this evaluation.

Events. This evaluation uses the Detection and Classification of Acoustic Scenes and Events (DCASE) 2018 challenge "*Large-scale weakly labeled semi-supervised sound event detection in domestic environments*" evaluation dataset. This dataset contains more than 2 hours of acoustic data with ten types of events. These ten types of events are – alarm bell ringing, dogs, cats, dishes, speech, frying, running water, blender, vacuum, and electric shaver. There is 3328 occurrence of events during this 2.14 hours. The maximum duration of an event is 10 seconds, and the minimum duration is 0.25 seconds. Multiple events coincide in this dataset, and this evaluation only considers the six types of events whose average durations are less than five seconds to reduce the concurrent occurrence of multiple events. The five considered events are – alarm bell ringing, dogs, cats, dishes, speech, and blender. These events' average duration is – 2.12 seconds, 1.50 seconds, 1.60 seconds, 0.64 seconds, 1.49 seconds, and 4.97 seconds. This subset has 3003 events with 307 alarm bells ringing, 450 dogs, 246 cats, 445 dishes, 1499 speech, and 56 blender audios.

Taskset. The taskset of this evaluation consists of a deep neural network-based acoustic event detector running in an MSP430FR5994 microcontroller. This detector consists of one convolution

layers and two fully-connected layers along with max pool and batch normalization in between. The runtime of this acoustic event detector is 3.89 seconds, and it consumes 26.72 mJ energy.

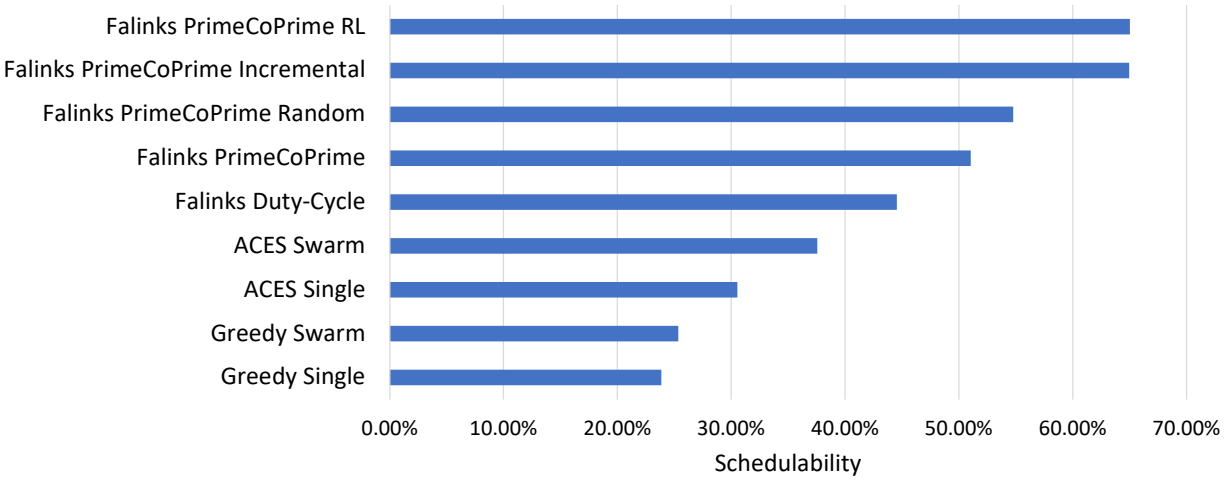


Figure 6.7: Schedulability comparison in real world scenario.

Table 6.4: Redundancy percentage & degree and mean time to non-observability for real world scenario.

	Redundancy Percentage	Redundancy Degree	Mean Time to Non-Observability
Greedy Single	0.00%	1	2.69 s
Greedy Swarm	90.82%	4	2.19 s
ACES Single	0.00%	1	12.5 s
ACES Swarm	80.78%	4	12.94 s
Falinks Duty-Cycle	2.6%	2	1.16 s
Falinks PrimeCoPrime	2.1%	2	1.27 s
Falinks PrimeCoPrime Random	30.08%	3	1.86 s
Falinks PrimeCoPrime Incremental	27.51%	3	1.60 s
Falinks PrimeCoPrime RL	21.12%	3	1.60 s

6.6.2 Performance

Figure 6.7 demonstrates the schedulability of Falinks PrimeCoPrime RL and Falinks PrimeCoPrime Incremental is almost identical with less than 1% difference. Falinks PrimeCoPrime Incremental performs better than Section 6.5 here due to the incremental changes in the harvestable energy trace. Despite the similar schedulability, redundancy degree, and mean time to non-observability (as shown in Table 6.4), the redundancy percentage of these two algorithms due to the more tailored increment by the Falinks PrimeCoPrime RL algorithm. Though these two algorithms' average schedulability is 65%, they only missed 24.91% of events on average. The lower schedulability occurred due to failure in completing jobs within the deadline. Though Falniks Duty Cycle did not have any redundancy

percentage in the simulation-based evaluation in Section 6.5, Table 6.4 shows some redundancy percentage in this scenario. It occurs due to the synchronization error of the timekeepers.

6.7 Discussion

6.7.1 Example Use-case Scenarios of Falinks

This section describes three example applications of Falinks: (1) preventive machine maintenance in extensive facilities, (2) methane detection and monitoring in mines, and (3) humidity temperature control in warehouses. All these applications continuously monitor and control a variable sensed by the sensor systems.

Preventive Machine Maintenance. Hundred million rotating machines in factories consume 53% of the world's electricity [252], and manufacturers lose $260,000 per hour due to machine failure and unplanned downtime. Preventive maintenance try to maintain the machine's optimum working condition and prevent any unplanned downtime due to breakdown. Besides factories, continuous monitoring and preventive maintenance of centralized HVAC systems reduce cost and save up to 30% of energy [253, 254]. Preventive maintenance of large-scale machines demands multiple sensor nodes which can operate 24/7. Low-cost batteryless sensor nodes are perfect for such a scenario without requiring any upfront capital expenditure or ongoing battery maintenance. These sensor nodes will monitor machine sound, vibration, temperature, and magnetic field data and perform on-device computing to determine the machine's current status.

Methane Detection in Mines. By 2020, global methane emissions from coal mines are estimated to reach 9% of the total global methane emission [255]. Methane levels can rise and fall rapidly, and informing miners about this changing condition on time allows them to respond quickly. An effective methane monitoring system will indicate a methane concentration of 1% before the surface's methane levels reach 5% [256]. I envision using batteryless methane sensor nodes, which will continuously collect methane gas readings and calculate the concentration. If the methane levels are 1% or higher, they will inform the miners, who will remove methane through ventilation systems [255].

Humidity and Temperature Control in Warehouses. Temperature and humidity can have a significant impact on the condition of stored goods in a warehouse. It is necessary to maintain an optimal temperature and humidity level (40%-50% RH) to reduce stock damage. An effective way of

monitoring should be cost-effective, easy-to-install, maintenance-free, and continuously monitoring. Batteryless sensor nodes are cheap, need no additional wiring, and require no battery maintenance. Falinks provide continuous monitoring to these batteryless sensor nodes, making them suitable for continuous temperature and humidity monitoring in every corner of a warehouse.

6.7.2 Handling Local Events

Though this chapter assumes that all events are global, in real-world scenarios, events might be local. When events are local, a subset of intermittent nodes is capable of sensing the target event. This intermittent node subset selection can be formulated as a wireless sensor network formation problem, such as a topology-based network with cluster-based formation. Here the cluster formation is done based on the predetermined subset of the nodes for the event type. Note that a single node can belong to multiple clusters. After formulating the clusters, our proposed Falinks algorithms apply to each cluster. It is more beneficial to select the nodes belonging to the maximum number of clusters to select the minimum number of nodes while having at least one node from each cluster. However, if the nodes belonging to maximum clusters is always chosen, it will soon exhaust these nodes' harvested energy and end up with only the non-overlapped nodes. In summary, along with the Falinks PrimeCoPrime algorithm, which uses the node with the highest energy harvesting rate, using the node that belongs to most clusters more frequently is effective.

6.7.3 Position of the Intermittent Nodes

As the nodes' placement may affect their energy harvesting and sensing capabilities, it is essential to ensure that a target event is not missed due to out of range. Intermittent nodes' placement depends on three factors – energy source, event source, and physical constraints. The intermittent node placement concerning the energy source affects the amount of harvestable energy available to that node. It directly contributes to the harvesting energy of the sensor nodes. This dissertation focuses on scheduling pre-positioned intermittent nodes instead of looking at placing the intermittent nodes. However, using the Falinks PrimeCoPrime algorithm, the intermittent nodes' energy source dependant placement can be determined. The placement of intermittent nodes for the event source contributes to its capability of sensing the event. If the sensor is out of the sensing range, then despite being powered on, it will fail to sense and infer the event. If all the nodes can be at the same place as the persistent system, this might not have been an issue, but it is not physically possible. This problem can be formed as the art gallery problem [257], a well-studied visibility problem in

computational geometry. Due to physical and environmental constraints, nodes might have a given placement. In such scenarios, the relation between the energy and event source is predetermined. This chapter focuses on this last scenario.

6.8 Summary

This chapter studies the unique problem of intermittently powered systems, where the systems fail to observe or sense the target event due to lack of sufficient energy to turn on. This chapter takes a unique approach to the problem by considering a swarm of intermittent nodes as an entity that collaboratively addresses the non-observability problem. However, the high communication cost hinders this collaborative behavior by imposing no communication rule for efficiency. This chapter proposes a Falinks scheduling algorithm that schedules the sleeping and waking up of these swarm of intermittent nodes to keep at least one intermittent node active at any time. This way, it emulates a persistently powered system with a swarm of intermittently powered sensor nodes.

CHAPTER 7

Conclusion and Future Works

7.1 Conclusion

This thesis focuses on enhancing time-sensitive and inference capable batteryless mobile computing devices; and addresses the growing need for sustainable sensing and control solutions. While existing works on batteryless computers primarily concentrate on harvesting units, real-time clock, intermittence management, and memory management, their timeliness, data-processing efficiency, and self-adaptation are still unexplored. Without time-sensitivity and an efficient processing layer, these systems fail to capture and process data within the deadline and generate irrelevant and ineffective results. It makes batteryless systems unsuitable for a wide range of necessary applications, from infrastructure monitoring to wildlife tracking, medical implants, and long-term health monitoring. For example, the long-life and low maintenance of batteryless systems make them perfect for tracking endangered wildlife (e.g., IBM's Project Rhino [23] monitors a herd of impalas for early rhino poacher detection). However, such a system is futile if it fails to notify the forest rangers before the poachers reach the rhino. Besides, these extreme edge devices require sophisticated inference and self-adaptation techniques for accurate and relevant outcomes.

This dissertation focuses on the intersection of systems and machine learning and ensure timely response in batteryless systems while maintaining high output quality by focusing on the control and processing layers. Chapter 5 is the first to propose adaptable machine learning approaches for time-aware ultra-constrained and intermittently powered hardware. This dissertation achieve this using a three-step approach: (1) understanding the physical phenomena and unique characteristics of the application domain (e.g., studying harvestable energy patterns in Chapter 5), (2) developing novel frameworks that leverage application-specific characteristics (Chapter 3), and (3) designing scheduling algorithms for deadline-aware task execution in intermittent systems (Chapter 5 and Chapter 4) and reducing non-obserservability time of intermittent systems (Chapter 6).

7.2 Future Directions

The future of computing will reshape how everyday objects will behave and influence human life by continuously learning human behavior, action, and environment. These everyday objects will beneficiate healthcare, environment monitoring, wildlife tracking, agriculture, and infrastructure maintenance. They will require life-long sensing and computing while having a small footprint, long life, and easy maintenance. My research seeks to develop these sustainable tiny computing systems by bringing out batteryless IoT devices' full potential. I aim to use them to design "deploy and forget" medical wearables and implantable that will continuously monitor health biomarkers. Another goal focuses on biodiversity and environmental preservation by the large-scale remote deployment of "never dying" self-sufficient sensor systems. I will split my effort into three directions – (1) integrate artificial intelligence, and machine learning with batteryless systems; (2) make batteryless systems adaptive to domain-specific needs; and (3) study the usability and impact of intelligent intermittent systems on the users. My future research will address high impact topics (e.g., remote health, biodiversity, urban infrastructure, and environment) and inspire multidisciplinary collaboration.

Integrate Artificial Intelligence with Intermittent Systems. The advancement in the intermittent systems has mostly emerged from the programming language and system architecture perspectives. However, the opportunities that lie within exploring different core machine learning algorithms for optimized performance are still unexplored. I have started investigating this avenue with my work, Zygarde, where I proposed adaptive convolution neural networks for intermittent systems. I want to optimize different deep neural network architectures, such as residual networks and inception networks, for intermittent systems. Such optimization will enable the inference of countless existing machine learning algorithms in tiny intermittent systems. Besides optimization, I aim to explore neuromorphic computing for intermittent systems. Moreover, I see the vast potential of self-supervised learning, active learning, and federated learning in intermittent systems that will utilize the massive data sensed by these lifelong devices to their maximum potential.

Adapt to Domain-Specific Necessities. I aim to extend the application of intermittent systems, which brings a new set of challenges from both the application domain and the batteryless domain. I foresee myself working in collaboration with experts in other disciplines like health and medicine. I aim to develop batteryless passive health monitoring wearables and implantable that can continuously

monitor post-operative patients, elderly, and chronic patients with minimal maintenance. One of my research goals is to collaborate with the experts in agriculture and environmental science; and develop a network of distributed batteryless intelligent computers for smart agriculture and biodiversity preservation by exploring distributed intermittent systems, opportunistic networking, hybrid architecture, and organic energy sources, e.g., microbial fuel. My short-term goal is to take my experience in respiratory monitoring, HVAC maintenance, and pedestrian safety one step further by investigating the requirements.

Study Usability and Impact. My research goal involves developing novel intermittent systems to improve human lives. For achieving this goal, understanding human needs and their perspective towards these developed systems is required. Thus far, no literature exists on understanding human perspective or studying human interaction with intermittent systems. For example, an intermittently powered, continuous blood sugar monitoring system will be hugely beneficial for a large population, and how this population interacts and trusts such wearables has a high impact on their health. I aim to perform detailed studies on the user experience, domain specifications before and after the development and deployment stages. It will pave the path to develop practical and impactful intermittently powered systems for our society and environment. This direction of my research will build a connection between human-computer interaction and batteryless systems. I am excited and committed to bringing together both sides' efforts to make intermittent systems have an immediate and lasting impact on the real-world.

APPENDIX A

NOMENCLATURE

A popular gameplay and anime named "Pokémon" [258] inspires the name of the algorithms in this dissertation. Pokémon refers to 898 fictional species who live in the wild or alongside humans and have extraordinary capabilities. This chapter describes the reasons behind each algorithm's name.

A.1 Zygarde

Zygarde is a dual-type Legendary Pokémon introduced in Generation VI of these anime series. This unique Pokémon does not eat any food and harvests solar energy, which is the most common renewable energy source. Zygarde has a Core which gathers different amount of its Cells to create one of the three alternate forms depending on the requirements (shown in Figure A.1) – (1) Zygarde 10% Forme occurs when Zygarde Core gathers 10% of the Cells nearby, (2) Zygarde 50% Forme occurs when Zygarde Core gathers 50% of the Cells nearby, and (3)Zygarde Complete Forme, which is the more powerful forme where it uses all its cells.

10% Forme. 50% Forme. Complete Forme.

Figure A.1: Different Forms of Zygarde

Similar to this Pokémon, our proposed Zygarde framework in Chapter 5 harvests renewable energy for execution and flexibly infers the necessary Deep Neural Network or DNN layers based on the available time and energy to ensure timeliness and increase inference capability.

A.2 Celebi

Celebi is a Mythical event Pokémon introduced in Generation II (shown in Figure A.2). This Pokémon can travel through time and exist simultaneously throughout time. It senses temporal anomalies and resolves temporal conflicts.

Figure A.2: Celebi.

Like its namesake, the Celebi scheduling algorithms in Chapter 4 resolve the conflicting cases where harvesting and computing can not coincide. These algorithms determine when to harvest and when to compute to maximize the number of computing jobs meeting deadlines.

A.3 Falinks

Falinks is a Fighting-type Pokémon introduced in Generation VIII. Falinks is a group of small, bipedal Pokémon. A single Falinks is a formation of six individuals, who usually march in a single file line, giving their formation the appearance of a caterpillar (shown in Figure A.3). A single Falinks individual is not effective in battle, and thus the six members rely on teamwork as their strategy to win battles and constantly change formation when they fight.

Figure A.3: Falinks.

Like the Pokémon Falinks, a single intermittent node is not sufficient for observing all the target events due to the lack of sufficient energy. However, like this Pokémon, the Falinks algorithms in Chapter 6 proposes to use a swarm of intermittent nodes who collaborative works towards emulating a persistently powered system that can observe all target events.

APPENDIX B

IMPLEMENTATION DETAILS

This section describes some key implementation techniques used throughout this thesis. It will include five key implementation components.

B.1 Installing Linux Tool Chain for MSP430

The microcontroller used in this thesis is an MSP430FR5994, which has MSP430 architecture and belongs to a microcontroller family from Texas Instrument (TI). Building code for the target device requires toolchain which comprises of a compiler with necessary headers, libraries and C runtime. The details of the used toolchain is described here: `https://github.com/CMUAbstract/releases/blob/master/Toolchains.md`. Following are the highlights of the steps described in the linked document.

1. Install the **Maker** dependency build system from here: `https://github.com/CMUAbstract/maker`.

2. Download and install the **TI GCC for MSP430**, a pre-built toolchain based on GCC.

3. Download and install **LLVM/Clang** toolchain, which compiles C down to MSP430 assembly using LLVM's MSP430 backend.

4. Download and install **mspdebug** and **tilib in mspdebug** to flash the binary code to microcontroller.

5. Download **Screen** using a command line instruction (*sudo apt install screen*) to allow displaying the output using serial port. More details about using Screen can be found here: `https://www.hostinger.com/tutorials/how-to-install-and-use-linux-screen/`.

B.2 Building, Flashing and Monitoring Code

The developed C code first needs to be built to generate application file (.out) and then flashed to the microcontroller for execution. For debugging the serial outputs (as PRINTF) also needs to be monitored. Following are the steps to achieve this.

1. Clean dependencies: *make <directory to code>/bld/gcc/depclean*

2. Build dependencies: *make <directory to code>/bld/gcc/dep*

3. Build target: *make <directory to code>/bld/gcc/all*

4. Open the Debugger: *mspdebug -v 3300 -d /dev/ttyACM0 tilib*

5. Flash the Code: *prog <directory to code>/bld/gcc/<output filename>.out*

6. Execute the Code: *run*

7. Monitor Serial Output: *screen /dev/ttyACM0 9600* [execute this command in a new terminal window.]

B.3 Intermittency Management Frameworks

This thesis uses two intermittency management frameworks ALPACA [13] and SONIC [8]. The details description of executing these frameworks are can be in the following link.

1. ALPACA: It is used for all general task loads. Code repository – `https://github.com/CMUAbstract/alpaca-oopsla2017`

2. SONIC: It is built on top of ALPACA and provides specific extensions for deep neural network (DNN) inference. Please download the dependancies individually as some of the linked urls in the repository in outdated. Code repository – `https://github.com/CMUAbstract/SONIC`

B.4 Reading Sensor Data

This section includes the details of how to read data from different types of sensors:

1. Acoustic Sensor Reading: The acoustic sensing requires one ADC and two buffers in the FRAM to store the audio signal. These two buffers work as a dual buffer or flip-flop buffer to avoid missing data. First download the software example from `https://software-dl.ti.com/msp430/msp430_public_sw/mcu/msp430/MSP-EXP430FR5994/latest/index_FDS.html`. Then in the *Firmware/Source/BOOSTXL-AUDIO_RecordPlayback_MSP430FR5994/* directory you will find the necessary codes to read audio sensor. *main.c* first shows the necessary circuit diagram to connet the microphones. If you do not want to use the Audio Boosterpack or the speaker in it, you will not need the SPI connection.

2. Camera Sensor Reading: To read the camera sensor, we need to use both the I2C and the SPI communications. The camera reading code can be found here: `https://github.com/cjosephson/backcam/tree/master/camera-mcu`.

B.5 Timekeeping

The details of the batteryless timekeeping system can be found here: `https://github.com/TUDSSL/Botoks#documentation`.

REFERENCES

[1] A. Rahmati, M. Salajegheh, D. Holcomb, J. Sorber, W. P. Burleson, and K. Fu, "Tardis: Time and remanence decay in sram to implement secure protocols on embedded devices without clocks," in *Proceedings of the 21st USENIX conference on Security symposium*, pp. 36–36, USENIX Association, 2012.

[2] J. Hester, N. Tobias, A. Rahmati, L. Sitanayah, D. Holcomb, K. Fu, W. P. Burleson, and J. Sorber, "Persistent clocks for batteryless sensing devices," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 15, no. 4, p. 77, 2016.

[3] J. de Winkel, C. Delle Donne, K. S. Yildirim, P. Pawełczak, and J. Hester, "Reliable timekeeping for intermittent computing," in *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*, pp. 53–67, 2020.

[4] J. Hester, K. Storer, and J. Sorber, "Timely execution on intermittently powered batteryless sensors," in *Proceedings of the 15th ACM Conference on Embedded Network Sensor Systems*, p. 17, ACM, 2017.

[5] P. Sparks, "The route to a trillion devices," *White Paper, ARM*, 2017.

[6] "Overcoming iot's battery barrier with self-powered sensors," Nov 2018. https://www.iottechexpo.com/2018/11/iot/overcoming-iots-battery-barrier-with-self-powered-sensors/.

[7] X. Zeng, J. Li, and Y. Ren, "Prediction of various discarded lithium batteries in china," in *2012 IEEE International Symposium on Sustainable Systems and Technology (ISSST)*, pp. 1–4, IEEE, 2012.

[8] G. Gobieski, B. Lucia, and N. Beckmann, "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*, pp. 199–213, ACM, 2019.

[9] B. Lucia, V. Balaji, A. Colin, K. Maeng, and E. Ruppel, "Intermittent computing: Challenges and opportunities," in *LIPIcs-Leibniz International Proceedings in Informatics*, vol. 71, Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2017.

[10] B. Ransford, J. Sorber, and K. Fu, "Mementos: System support for long-running computation on rfid-scale devices," *Acm Sigplan Notices*, vol. 47, no. 4, pp. 159–170, 2012.

[11] 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 Embedded Systems Letters*, vol. 7, no. 1, pp. 15–18, 2015.

[12] K. S. Yıldırım, A. Y. Majid, D. Patoukas, K. Schaper, P. Pawelczak, and J. Hester, "Ink: Reactive kernel for tiny batteryless sensors," in *Proceedings of the 16th ACM Conference on Embedded Networked Sensor Systems*, ACM, 2018.

[13] K. Maeng, A. Colin, and B. Lucia, "Alpaca: Intermittent execution without checkpoints," *Proceedings of the ACM on Programming Languages*, vol. 1, no. OOPSLA, p. 96, 2017.

[14] E. Bäumker, F. Schüle, and P. Woias, "Development of a batteryless vhf-beacon and tracker for mammals," in *Journal of Physics: Conference Series*, vol. 1052, p. 012005, IOP Publishing, 2018.

[15] F. Spruyt, "Project rhino," *IBM*, 2011. https://www.projectrhinokzn.org.

[16] A. Dewan, S. U. Ay, M. N. Karim, and H. Beyenal, "Alternative power sources for remote sensors: A review," *Journal of power sources*, vol. 245, pp. 129–143, 2014.

[17] G. Werner-Allen, J. Johnson, M. Ruiz, J. Lees, and M. Welsh, "Monitoring volcanic eruptions with a wireless sensor network.," in *EWSN*, vol. 5, pp. 108–120, 2005.

[18] "Battery-free wireless sensing for smart agriculture." https://www.onsemi.com/blog/iot/Battery-Free-Wireless-Sensing-for-Smart-Agriculture?canonicalURL=/blog/iot/Battery-Free-Wireless-Sensing-for-Smart-Agriculture/.

[19] https://tinyurl.com/y2abguyf.

[20] H. Truong, S. Zhang, U. Muncuk, P. Nguyen, N. Bui, A. Nguyen, Q. Lv, K. Chowdhury, T. Dinh, and T. Vu, "Capband: Battery-free successive capacitance sensing wristband for hand gesture recognition," in *Proceedings of the 16th ACM Conference on Embedded Networked Sensor Systems*, pp. 54–67, ACM, 2018.

[21] D. C. Ranasinghe, R. L. S. Torres, A. P. Sample, J. R. Smith, K. Hill, and R. Visvanathan, "Towards falls prevention: a wearable wireless and battery-less sensing and automatic identification tag for real time monitoring of human movements," in *2012 Annual International Conference of the IEEE Engineering in Medicine and Biology Society*, pp. 6402–6405, IEEE, 2012.

[22] J. D. Hester and J. Sorber, "New directions: The future of sensing is batteryless, intermittent, and awesome," in *15th ACM Conference on Embedded Networked Sensor Systems (SenSys 2017)*, 2017.

[23] R. L. S. Torres, R. Visvanathan, D. Abbott, K. D. Hill, and D. C. Ranasinghe, "A battery-less and wireless wearable sensor system for identifying bed and chair exits in a pilot trial in hospitalized older people," *PloS one*, vol. 12, no. 10, p. e0185670, 2017.

[24] https://www.fierceelectronics.com/components/battery-less-sensors-make-clothing-smart-and-autonomous.

[25] A. Abedi, A. Razi, and F. Afghan, "Small battery-free wireless sensor networks for structural health monitoring," in *8th International Workshop on Structural Health Monitoring, Stanford, CA, September*, pp. 13–15, 2011.

[26] T. N. Gia, M. Ali, I. B. Dhaou, A. M. Rahmani, T. Westerlund, P. Liljeberg, and H. Tenhunen, "Iot-based continuous glucose monitoring system: A feasibility study," *Procedia Computer Science*, vol. 109, pp. 327–334, 2017.

[27] A. Haeberlin, A. Zurbuchen, S. Walpen, J. Schaerer, T. Niederhauser, C. Huber, H. Tanner, H. Servatius, J. Seiler, H. Haeberlin, *et al.*, "The first batteryless, solar-powered cardiac pacemaker," *Heart rhythm*, vol. 12, no. 6, pp. 1317–1323, 2015.

[28] M. B. Hoy, "Alexa, siri, cortana, and more: an introduction to voice assistants," *Medical reference services quarterly*, 2018.

[29] M. T. Islam, B. Islam, and S. Nirjon, "Soundsifter: Mitigating overhearing of continuous listening devices," in *Proceedings of the 15th Annual International Conference on Mobile Systems, Applications, and Services*, pp. 29–41, ACM, 2017.

[30] D. de Godoy, B. Islam, S. Xia, M. T. Islam, R. Chandrasekaran, Y.-C. Chen, S. Nirjon, P. R. Kinget, and X. Jiang, "Paws: A wearable acoustic system for pedestrian safety," in *2018 IEEE/ACM Third International Conference on Internet-of-Things Design and Implementation (IoTDI)*, pp. 237–248, IEEE, 2018.

[31] R. Srinivasan, M. T. Islam, B. Islam, Z. Wang, T. Sookoor, O. Gnawali, and S. Nirjon, "Preventive maintenance of centralized hvac systems: use of acoustic sensors, feature extraction, and unsupervised learning," in *Proceedings of the 15th IBPSA Conference*, pp. 2518–2524, 2017.

[32] D. Patoukas, K. S. Yildirim, A. Y. Majid, J. Hester, and P. Pawełczak, "Feasibility of multi-tenancy on intermittent power," in *Proceedings of the 6th International Workshop on Energy Harvesting & Energy-Neutral Sensing Systems*, pp. 26–31, 2018.

[33] S. Lee and S. Nirjon, "Neuro.ZERO: A Zero-Energy Neural Network Accelerator for Embedded Sensing and Inference Systems," in *The 17th ACM Conference on Embedded Networked Sensor Systems (SenSys 2019)*, ACM, 2019.

[34] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding," *arXiv preprint arXiv:1510.00149*, 2015.

[35] T. M. Nabhan and A. Y. Zomaya, "Toward generating neural network structures for function approximation," *Neural Networks*, vol. 7, no. 1, pp. 89–99, 1994.

[36] S. Yao, Y. Zhao, A. Zhang, L. Su, and T. Abdelzaher, "Deepiot: Compressing deep neural network structures for sensing systems with a compressor-critic framework," in *Proceedings of the 15th ACM Conference on Embedded Network Sensor Systems*, 2017.

[37] S. Yao, Y. Zhao, H. Shao, A. Zhang, C. Zhang, S. Li, and T. Abdelzaher, "Rdeepsense: Reliable deep mobile computing models with uncertainty estimations," *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, vol. 1, no. 4, p. 173, 2018.

[38] S. Yao, Y. Zhao, H. Shao, C. Zhang, A. Zhang, D. Liu, S. Liu, L. Su, and T. Abdelzaher, "Apdeepsense: Deep learning uncertainty estimation without the pain for iot applications," in *2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS)*, pp. 334–343, IEEE, 2018.

[39] S. Yao, Y. Zhao, H. Shao, S. Liu, D. Liu, L. Su, and T. Abdelzaher, "Fastdeepiot: Towards understanding and optimizing neural network execution time on mobile and embedded devices," *arXiv preprint arXiv:1809.06970*, 2018.

[40] B. Ransford, S. Clark, M. Salajegheh, and K. Fu, "Getting things done on computational rfids with energy-aware checkpointing and voltage-aware scheduling," *Proceedings of the 2008 Conference on Power Aware Computing and Systems*, pp. 5–5, 2008. http://dl.acm.org/citation.cfm?id=1855610.1855615.

[41] D. Balsamo, A. S. Weddell, A. Das, A. R. Arreola, D. Brunelli, B. M. Al-Hashimi, G. V. Merrett, and L. Benini, "Hibernus++: a self-calibrating and adaptive system for transiently-powered embedded devices," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 35, no. 12, pp. 1968–1980, 2016.

[42] K. Maeng and B. Lucia, "Adaptive dynamic checkpointing for safe efficient intermittent computing," in *13th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 18)*, 2018.

[43] S. Ahmed, N. A. Bhatti, M. H. Alizai, J. H. Siddiqui, and L. Mottola, "Efficient intermittent computing with differential checkpointing," 2019.

[44] B. Lucia and B. Ransford, "A simpler, safer programming and execution model for intermittent systems," *ACM SIGPLAN Notices*, vol. 50, no. 6, pp. 575–585, 2015.

[45] A. Colin and B. Lucia, "Chain: tasks and channels for reliable intermittent programs," *ACM SIGPLAN Notices*, 2016.

[46] A. Colin and B. Lucia, "Termination checking and task decomposition for task-based intermittent programs," in *Proceedings of the 27th International Conference on Compiler Construction*, pp. 116–127, ACM, 2018.

[47] A. Y. Majid, C. D. Donne, K. Maeng, A. Colin, K. S. Yildirim, B. Lucia, and P. Pawełczak, "Dynamic task-based intermittent execution for energy-harvesting devices," *ACM Transactions on Sensor Networks (TOSN)*, vol. 16, no. 1, pp. 1–24, 2020.

[48] M. Buettner, B. Greenstein, and D. Wetherall, "Dewdrop: an energy-aware runtime for computational rfid," in *Proc. USENIX NSDI*, 2011.

[49] T. Zhu, A. Mohaisen, Y. Ping, and D. Towsley, "Deos: Dynamic energy-oriented scheduling for sustainable wireless sensor networks," in *INFOCOM, 2012 Proceedings IEEE*, pp. 2363–2371, IEEE, 2012.

[50] C. Moser, D. Brunelli, L. Thiele, and L. Benini, "Lazy scheduling for energy harvesting sensor nodes," in *IFIP Working Conference on Distributed and Parallel Embedded Systems*, Springer, 2006.

[51] C. Moser, D. Brunelli, L. Thiele, and L. Benini, "Real-time scheduling for energy harvesting sensor nodes," *Real-Time Systems*, 2007.

[52] F. Fraternali, B. Balaji, Y. Agarwal, and R. K. Gupta, "Aces: Automatic configuration of energy harvesting sensors with reinforcement learning," *ACM Transactions on Sensor Networks (TOSN)*, vol. 16, no. 4, pp. 1–31, 2020.

[53] A. Colin, E. Ruppel, and B. Lucia, "A reconfigurable energy storage architecture for energy-harvesting devices," in *Proceedings of the Twenty-Third International Conference on Architectural Support for Programming Languages and Operating Systems*, pp. 767–781, ACM, 2018.

[54] J. Hester, L. Sitanayah, and J. Sorber, "Tragedy of the coulombs: Federating energy storage for tiny, intermittently-powered sensors," in *Proceedings of the 13th ACM Conference on Embedded Networked Sensor Systems*, pp. 5–16, ACM, 2015.

[55] A. Crovetto, F. Wang, and O. Hansen, "Modeling and optimization of an electrostatic energy harvesting device," *journal of microelectromechanical systems*, vol. 23, no. 5, pp. 1141–1155, 2014.

[56] H. Sharma, A. Haque, and Z. Jaffery, "Modeling and optimisation of a solar energy harvesting system for wireless sensor network nodes," *Journal of Sensor and Actuator Networks*, vol. 7, no. 3, p. 40, 2018.

[57] J. Jia, X. Shan, D. Upadrashta, T. Xie, Y. Yang, and R. Song, "Modeling and analysis of upright piezoelectric energy harvester under aerodynamic vortex-induced vibration," *Micromachines*, vol. 9, no. 12, p. 667, 2018.

[58] D. R. Cox, "Prediction by exponentially weighted moving averages and related methods," *Journal of the Royal Statistical Society. Series B (Methodological)*, pp. 414–422, 1961.

[59] J. S. Hunter, "The exponentially weighted moving average," *Journal of quality technology*, vol. 18, no. 4, pp. 203–210, 1986.

[60] C. M. Vigorito, D. Ganesan, and A. G. Barto, "Adaptive control of duty cycling in energy-harvesting wireless sensor networks," in *2007 4th Annual IEEE Communications Society Conference on Sensor, Mesh and Ad Hoc Communications and Networks*, pp. 21–30, IEEE, 2007.

[61] A. Kansal, J. Hsu, S. Zahedi, and M. B. Srivastava, "Power management in energy harvesting sensor networks," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 6, no. 4, p. 32, 2007.

[62] J. R. Piorno, C. Bergonzini, D. Atienza, and T. S. Rosing, "Prediction and management in energy harvested wireless sensor nodes," in *Wireless Communication, Vehicular Technology, Information Theory and Aerospace & Electronic Systems Technology, 2009. Wireless VITAE 2009. 1st International Conference on*, pp. 6–10, IEEE, 2009.

[63] H. Suehrcke and P. McCormick, "A performance prediction method for solar energy systems," *Solar energy*, vol. 48, no. 3, pp. 169–175, 1992.

[64] J. San Miguel, K. Ganesan, M. Badr, and N. E. Jerger, "The eh model: Analytical exploration of energy-harvesting architectures," *IEEE Computer Architecture Letters*, vol. 17, no. 1, pp. 76–79, 2018.

[65] V. Sze, Y.-H. Chen, T.-J. Yang, and J. S. Emer, "Efficient processing of deep neural networks: A tutorial and survey," *Proceedings of the IEEE*, vol. 105, no. 12, pp. 2295–2329, 2017.

[66] B. Hanin, "Universal function approximation by deep neural nets with bounded width and relu activations," *arXiv preprint arXiv:1708.02691*, 2017.

[67] Z. Lu, H. Pu, F. Wang, Z. Hu, and L. Wang, "The expressive power of neural networks: A view from the width," in *Advances in Neural Information Processing Systems*, pp. 6231–6239, 2017.

[68] K. Hornik, "Approximation capabilities of multilayer feedforward networks," *Neural networks*, vol. 4, no. 2, pp. 251–257, 1991.

[69] G. Cybenko, "Approximation by superpositions of a sigmoidal function," *Mathematics of control, signals and systems*, 1989.

[70] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

[71] Y. Guo, M. Tan, Q. Wu, J. Chen, A. V. D. Hengel, and Q. Shi, "The shallow end: Empowering shallower deep-convolutional networks through auxiliary outputs," *arXiv preprint arXiv:1611.01773*, 2016.

[72] S. Sun, W. Chen, L. Wang, X. Liu, and T.-Y. Liu, "On the depth of deep neural networks: A theoretical view," in *Thirtieth AAAI Conference on Artificial Intelligence*, 2016.

[73] T. He, Y. Fan, Y. Qian, T. Tan, and K. Yu, "Reshaping deep neural network for fast decoding by node-pruning," in *Acoustics, Speech and Signal Processing (ICASSP), 2014 IEEE International Conference on*, pp. 245–249, IEEE, 2014.

[74] J. Van Leeuwen, "On the construction of huffman trees.," in *ICALP*, pp. 382–410, 1976.

[75] Y. Guo, A. Yao, and Y. Chen, "Dynamic network surgery for efficient dnns," in *Advances In Neural Information Processing Systems*, pp. 1379–1387, 2016.

[76] N. D. Lane and P. Georgiev, "Can deep learning revolutionize mobile sensing?," in *Proceedings of the 16th International Workshop on Mobile Computing Systems and Applications*, pp. 117–122, ACM, 2015.

[77] J. Johnson, "Rethinking floating point for deep learning," *arXiv preprint arXiv:1811.01721*, 2018.

[78] C. De Sa, M. Feldman, C. Ré, and K. Olukotun, "Understanding and optimizing asynchronous low-precision stochastic gradient descent," in *ACM SIGARCH Computer Architecture News*, vol. 45, pp. 561–574, ACM, 2017.

[79] S. Han, X. Liu, H. Mao, J. Pu, A. Pedram, M. A. Horowitz, and W. J. Dally, "Eie: efficient inference engine on compressed deep neural network," in *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*, 2016.

[80] V. Vanhoucke, A. Senior, and M. Z. Mao, "Improving the speed of neural networks on cpus," in *Proc. Deep Learning and Unsupervised Feature Learning NIPS Workshop*, vol. 1, p. 4, Citeseer, 2011.

[81] K. Hwang and W. Sung, "Fixed-point feedforward deep neural network design using weights+ 1, 0, and- 1," in *Signal Processing Systems (SiPS), 2014 IEEE Workshop on*, pp. 1–6, IEEE, 2014.

[82] E. L. Denton, W. Zaremba, J. Bruna, Y. LeCun, and R. Fergus, "Exploiting linear structure within convolutional networks for efficient evaluation," in *Advances in neural information processing systems*, pp. 1269–1277, 2014.

[83] M. Lin, Q. Chen, and S. Yan, "Network in network," *arXiv preprint arXiv:1312.4400*, 2013.

[84] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1–9, 2015.

[85] S. Bhattacharya and N. D. Lane, "Sparsification and separation of deep learning layers for constrained resource inference on wearables," in *Proceedings of the 14th ACM Conference on Embedded Network Sensor Systems CD-ROM*, pp. 176–189, ACM, 2016.

[86] F. Chollet, "Xception: Deep learning with depthwise separable convolutions," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1251–1258, 2017.

[87] P. Nakkiran, R. Alvarez, R. Prabhavalkar, and C. Parada, "Compressing deep neural networks using a rank-constrained topology," 2015.

[88] C. Szegedy, S. Ioffe, V. Vanhoucke, and A. A. Alemi, "Inception-v4, inception-resnet and the impact of residual connections on learning," in *Thirty-First AAAI Conference on Artificial Intelligence*, 2017.

[89] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 2818–2826, 2016.

[90] M. Wang, B. Liu, and H. Foroosh, "Factorized convolutional neural networks.," in *ICCV Workshops*, pp. 545–553, 2017.

[91] J. Xue, J. Li, and Y. Gong, "Restructuring of deep neural network acoustic models with singular value decomposition.," in *Interspeech*, pp. 2365–2369, 2013.

[92] N. D. Lane, S. Bhattacharya, P. Georgiev, C. Forlivesi, L. Jiao, L. Qendro, and F. Kawsar, "Deepx: A software accelerator for low-power deep learning inference on mobile devices," in *Proceedings of the 15th International Conference on Information Processing in Sensor Networks*, p. 23, IEEE Press, 2016.

[93] C. Bucilua, R. Caruana, and A. Niculescu-Mizil, "Model compression," in *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 535–541, ACM, 2006.

[94] G. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network," *arXiv preprint arXiv:1503.02531*, 2015.

[95] M. Courbariaux, Y. Bengio, and J.-P. David, "Binaryconnect: Training deep neural networks with binary weights during propagations," in *Advances in neural information processing systems*, pp. 3123–3131, 2015.

[96] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio, "Binarized neural networks," in *Advances in neural information processing systems*, pp. 4107–4115, 2016.

[97] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi, "Xnor-net: Imagenet classification using binary convolutional neural networks," in *European Conference on Computer Vision*, pp. 525–542, Springer, 2016.

[98] M. Courbariaux and Y. Bengio, "Binarynet: Training deep neural networks with weights and activations constrained to+ 1 or- 1. arxiv: 1602.02830, 2016," 2017.

[99] S. Teerapittayanon, B. McDanel, and H.-T. Kung, "Branchynet: Fast inference via early exiting from deep neural networks," in *2016 23rd International Conference on Pattern Recognition (ICPR)*, pp. 2464–2469, IEEE, 2016.

[100] T. Bolukbasi, J. Wang, O. Dekel, and V. Saligrama, "Adaptive neural networks for efficient inference," *arXiv preprint arXiv:1702.07811*, 2017.

[101] M. Figurnov, M. D. Collins, Y. Zhu, L. Zhang, J. Huang, D. P. Vetrov, and R. Salakhutdinov, "Spatially adaptive computation time for residual networks.," in *CVPR*, vol. 2, p. 7, 2017.

[102] S. Leroux, S. Bohez, E. De Coninck, T. Verbelen, B. Vankeirsbilck, P. Simoens, and B. Dhoedt, "The cascading neural network: building the internet of smart things," *Knowledge and Information Systems*, vol. 52, no. 3, pp. 791–814, 2017.

[103] R. Teja Mullapudi, W. R. Mark, N. Shazeer, and K. Fatahalian, "Hydranets: Specialized dynamic architectures for efficient inference," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 8080–8089, 2018.

[104] A. Odena, D. Lawson, and C. Olah, "Changing model behavior at test-time using reinforcement learning," *arXiv preprint arXiv:1702.07780*, 2017.

[105] G. Huang, D. Chen, T. Li, F. Wu, L. van der Maaten, and K. Q. Weinberger, "Multi-scale dense networks for resource efficient image classification," *arXiv preprint arXiv:1703.09844*, 2017.

[106] A. Veit and S. Belongie, "Convolutional networks with adaptive inference graphs," in *Proceedings of the European Conference on Computer Vision (ECCV)*, pp. 3–18, 2018.

[107] G. Huang, Y. Sun, Z. Liu, D. Sedra, and K. Q. Weinberger, "Deep networks with stochastic depth," in *European conference on computer vision*, pp. 646–661, Springer, 2016.

[108] N. J. Harvey, J. Dunagan, M. Jones, S. Saroiu, M. Theimer, and A. Wolman, "Skipnet: A scalable overlay network with practical locality properties," 2002.

[109] J. Yu and T. S. Huang, "Universally slimmable networks and improved training techniques," in *Proceedings of the IEEE International Conference on Computer Vision*, pp. 1803–1811, 2019.

[110] S. Karayev, T. Baumgartner, M. Fritz, and T. Darrell, "Timely object recognition," in *Advances in Neural Information Processing Systems*, pp. 890–898, 2012.

[111] S. Karayev, M. Fritz, and T. Darrell, "Anytime recognition of objects and scenes," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 572–579, 2014.

[112] H. Hu, D. Dey, M. Hebert, and J. A. Bagnell, "Learning anytime predictions in neural networks via adaptive loss balancing," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, pp. 3812–3821, 2019.

[113] E. Kim, C. Ahn, and S. Oh, "Nestednet: Learning nested sparse structures in deep neural networks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 8669–8678, 2018.

[114] W.-K. Shih and J. W.-S. Liu, "On-line scheduling of imprecise computations to minimize error," in *Real-Time Systems Symposium, 1992*, pp. 280–289, IEEE, 1992.

[115] Y.-X. Zhang, C.-H. Fang, and Y. Wang, "A feedback-driven online scheduler for processes with imprecise computing," *Journal of Software*, 2004.

[116] Y. WANG and Y. ZHANG, "Feedback driven online scheduling of processes with imprecise computation," in *Algorithms And Architectures For Parallel Processing: ICA3PP 2000*, World Scientific, 2000.

[117] H. Chishiro, A. Takeda, K. Funaoka, and N. Yamasaki, "Semi-fixed-priority scheduling: New priority assignment policy for practical imprecise computation," in *2010 IEEE 16th International Conference on Embedded and Real-Time Computing Systems and Applications*, pp. 339–348, IEEE, 2010.

[118] R. Rajkumar, C. Lee, J. Lehoczky, and D. Siewiorek, "A resource allocation model for qos management," in *Proceedings Real-Time Systems Symposium*, pp. 298–307, IEEE, 1997.

[119] C. Lee, J. Lehoczky, D. Siewiorek, R. Rajkumar, and J. Hansen, "A scalable solution to the multi-resource qos problem," in *Proceedings 20th IEEE Real-Time Systems Symposium (Cat. No. 99CB37054)*, pp. 315–326, IEEE, 1999.

[120] S. K. Baruah, A. Burns, and R. I. Davis, "Response-time analysis for mixed criticality systems," in *2011 IEEE 32nd Real-Time Systems Symposium*, pp. 34–43, IEEE, 2011.

[121] S. Baruah, H. Li, and L. Stougie, "Towards the design of certifiable mixed-criticality systems," in *2010 16th IEEE Real-Time and Embedded Technology and Applications Symposium*, pp. 13–22, IEEE, 2010.

[122] S. Vestal, "Preemptive scheduling of multi-criticality systems with varying degrees of execution time assurance," in *28th IEEE International Real-Time Systems Symposium (RTSS 2007)*, pp. 239–243, IEEE, 2007.

[123] H. Li and S. Baruah, "An algorithm for scheduling certifiable mixed-criticality sporadic task systems," in *2010 31st IEEE Real-Time Systems Symposium*, pp. 183–192, IEEE, 2010.

[124] F. Dorin, P. Richard, M. Richard, and J. Goossens, "Schedulability and sensitivity analysis of multiple criticality tasks with fixed-priorities," *Real-Time Systems*, vol. 46, no. 3, pp. 305–331, 2010.

[125] G. Buttazzo, M. Spuri, and F. Sensini, "Value vs. deadline scheduling in overload conditions," in *Proceedings 16th IEEE Real-Time Systems Symposium*, pp. 90–99, IEEE, 1995.

[126] S. Bateni and C. Liu, "Apnet: Approximation-aware real-time neural network," in *2018 IEEE Real-Time Systems Symposium (RTSS)*, pp. 67–79, IEEE, 2018.

[127] H. Zhou, S. Bateni, and C. Liu, "Sˆ 3dnn: Supervised streaming and scheduling for gpu-accelerated real-time dnn workloads," in *2018 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pp. 190–201, IEEE, 2018.

[128] F. B. Yahya, C. J. Lukas, and B. H. Calhoun, "A top-down approach to building batteryless self-powered systems for the internet-of-things," *Journal of Low Power Electronics and Applications*, vol. 8, no. 2, p. 21, 2018.

[129] S. Baruah, "The limited-preemption uniprocessor scheduling of sporadic task systems," in *17th Euromicro Conference on Real-Time Systems (ECRTS'05)*, pp. 137–144, IEEE, 2005.

[130] A. Asaro, I. Laksono, J. Doyle, and G. F. Grigor, "Method and apparatus for improved double buffering," 2000. US Patent 6,100,906.

[131] J. R. Smith, A. P. Sample, P. S. Powledge, S. Roy, and A. Mamishev, "A wirelessly-powered platform for sensing and computation," in *International Conference on Ubiquitous Computing*, pp. 495–506, Springer, 2006.

[132] A. P. Sample, D. J. Yeager, P. S. Powledge, A. V. Mamishev, and J. R. Smith, "Design of an rfid-based battery-free programmable sensing platform," *IEEE transactions on instrumentation and measurement*, vol. 57, no. 11, p. 2608, 2008.

[133] V. Talla, B. Kellogg, S. Gollakota, and J. R. Smith, "Battery-free cellphone," *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, vol. 1, no. 2, p. 25, 2017.

[134] S. Sasaki, T. Seki, and S. Sugiyama, "Batteryless accelerometer using power feeding system of rfid," in *SICE-ICASE, 2006. International Joint Conference*, pp. 3567–3570, IEEE, 2006.

[135] J. Sosa, S. Garcia-Alonso, J. Monzon-Verona, J. Montiel-Nelson, A. Beriain, E. Fernandez, H. Solar, A. Garcia-Alonso, and I. Gutierrez, "Wireless and batteryless blood pressure sensor," in *Circuits and Systems (MWSCAS), 2011 IEEE 54th International Midwest Symposium on*, pp. 1–4, IEEE, 2011.

[136] S. Naderiparizi, A. N. Parks, Z. Kapetanovic, B. Ransford, and J. R. Smith, "Wispcam: A battery-free rfid camera," in *RFID (RFID), 2015 IEEE International Conference on*, pp. 166–173, IEEE, 2015.

[137] C.-C. Chen, G.-N. Sung, W.-C. Chen, C.-T. Kuo, J.-J. Chue, C.-M. Wu, and C.-M. Huang, "A wireless and batteryless intelligent carbon monoxide sensor," *Sensors*, vol. 16, no. 10, p. 1568, 2016.

[138] "Max44007 low-power digital ambient light sensor." `https://goo.gl/9cZyC7`.

[139] "Mcp9700-e/to temperature and humidity sensors." `https://goo.gl/HSZZg8`.

[140] "Inmp510 mems analog microphone." `https://goo.gl/FAM3kF`.

[141] "Low power accelarometer." `https://goo.gl/RAoXnF`.

[142] C. Perez, A. Santibanez, C. Holzmann, P. Estevez, and C. Held, "Power requirements for vibro-tactile piezo-electric and electromechanical transducers," *Medical and Biological Engineering and Computing*, vol. 41, no. 6, pp. 718–726, 2003.

[143] "Hm01b0 ultra low power cis." `https://goo.gl/hSg9y2`.

[144] "Ultra-low power analog sensor module for carbon monoxide." `https://goo.gl/YEYBvd`.

[145] P. Bhuvaneswari, R. Balakumar, V. Vaidehi, and P. Balamuralidhar, "Solar energy harvesting for wireless sensor networks," in *2009 First International Conference on Computational Intelligence, Communication Systems and Networks*, pp. 57–61, IEEE, 2009.

[146] C. Alippi and C. Galperti, "An adaptive system for optimal solar energy harvesting in wireless sensor network nodes," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 55, no. 6, pp. 1742–1750, 2008.

[147] N. S. Shenck and J. A. Paradiso, "Energy scavenging with shoe-mounted piezoelectrics," *IEEE micro*, vol. 21, no. 3, pp. 30–42, 2001.

[148] J. Kymissis, C. Kendall, J. Paradiso, and N. Gershenfeld, "Parasitic power harvesting in shoes," in *Wearable Computers, 1998. Digest of Papers. Second International Symposium on*, pp. 132–139, IEEE, 1998.

[149] T. Starner, "Human-powered wearable computing," *IBM systems Journal*, vol. 35, no. 3/4, pp. 618–629, 1996.

[150] C. Park and P. H. Chou, "Ambimax: Autonomous energy harvesting platform for multi-supply wireless sensor nodes," in *Sensor and Ad Hoc Communications and Networks, 2006. SECON'06. 2006 3rd Annual IEEE Communications Society on*, vol. 1, pp. 168–177, IEEE, 2006.

[151] M. A. Weimer, T. S. Paing, and R. A. Zane, "Remote area wind energy harvesting for low-power autonomous sensors," in *Power Electronics Specialists Conference, 2006. PESC'06. 37th IEEE*, pp. 1–5, IEEE, 2006.

[152] G. Xu, Y. Yang, Y. Zhou, and J. Liu, "Wearable thermal energy harvester powered by human foot," *Frontiers in Energy*, vol. 7, no. 1, pp. 26–38, 2013.

[153] E. Lawrence and G. Snyder, "A study of heat sink performance in air and soil for use in a thermoelectric energy harvesting device," in *Thermoelectrics, 2002. Proceedings ICT'02. Twenty-First International Conference on*, pp. 446–449, IEEE, 2002.

[154] X. Lu, P. Wang, D. Niyato, D. I. Kim, and Z. Han, "Wireless networks with rf energy harvesting: A contemporary survey," *IEEE Communications Surveys & Tutorials*, vol. 17, no. 2, pp. 757–789, 2015.

[155] S. Sudevalayam and P. Kulkarni, "Energy harvesting sensor nodes: Survey and implications," *IEEE Communications Surveys & Tutorials*, vol. 13, no. 3, pp. 443–461, 2011.

[156] "Fundamentals of ambient energy transducers in energy harvesting systems," 2015. https://www.ecnmag.com/article/2012/03/fundamentals-ambient-energy-transducers-energy-harvesting-systems.

[157] X. Jiang, J. Polastre, and D. Culler, "Perpetual environmentally powered sensor networks," in *Proceedings of the 4th international symposium on Information processing in sensor networks*, p. 65, IEEE Press, 2005.

[158] V. Raghunathan, A. Kansal, J. Hsu, J. Friedman, and M. Srivastava, "Design considerations for solar energy harvesting wireless embedded systems," in *Proceedings of the 4th international symposium on Information processing in sensor networks*, p. 64, IEEE Press, 2005.

[159] J. Taneja, J. Jeong, and D. Culler, "Design, modeling, and capacity planning for micro-solar power sensor networks," in *Proceedings of the 7th international conference on Information processing in sensor networks*, pp. 407–418, IEEE Computer Society, 2008.

[160] R. Riemer and A. Shapiro, "Biomechanical energy harvesting from human motion: theory, state of the art, design guidelines, and future directions," *Journal of neuroengineering and rehabilitation*, vol. 8, no. 1, p. 22, 2011.

[161] H. Kulah and K. Najafi, "Energy scavenging from low-frequency vibrations by using frequency up-conversion for wireless sensor applications," *IEEE Sensors Journal*, vol. 8, no. 3, pp. 261–268, 2008.

[162] G. Xu, Y. Yang, Y. Zhou, and J. Liu, "Wearable thermal energy harvester powered by human foot," *Frontiers in Energy*, vol. 7, no. 1, pp. 26–38, 2013.

[163] "Nxp semiconductors spi real time clock/calendar," 2012. http://www.nxp.com/documents/data sheet/ PCF2123.

[164] "Timer module," 2015. https://datasheets.maximintegrated.com/en/ds/DS3231.pdf.

[165] "Abracon corporation ab08x5real-time clock family," 2015. http://abracon.com/Precisiontiming/AB08X5-RTC.

[166] H. W. Ng, X. Jiang, L.-K. Merhi, and C. Menon, "Investigation of the feasibility of strain gages as pressure sensors for force myography," in *International Conference on Bioinformatics and Biomedical Engineering*, pp. 261–270, Springer, 2017.

[167] V. Talla, B. Kellogg, B. Ransford, S. Naderiparizi, S. Gollakota, and J. R. Smith, "Powering the next billion devices with wi-fi," in *Proceedings of the 11th ACM Conference on Emerging Networking Experiments and Technologies*, p. 4, ACM, 2015.

[168] U. Olgun, C.-C. Chen, and J. L. Volakis, "Design of an efficient ambient wifi energy harvesting system," *IET Microwaves, Antennas & Propagation*, vol. 6, no. 11, pp. 1200–1206, 2012.

[169] S. Kim, R. Vyas, J. Bito, K. Niotaki, A. Collado, A. Georgiadis, and M. M. Tentzeris, "Ambient rf energy-harvesting technologies for self-sustainable standalone wireless sensor platforms," *Proceedings of the IEEE*, vol. 102, no. 11, pp. 1649–1666, 2014.

[170] R. J. Vyas, B. B. Cook, Y. Kawahara, and M. M. Tentzeris, "E-wehp: A batteryless embedded sensor-platform wirelessly powered from ambient digital-tv signals," *IEEE Transactions on microwave theory and techniques*, vol. 61, no. 6, pp. 2491–2505, 2013.

[171] M. Pinuela, P. D. Mitcheson, and S. Lucyszyn, "Ambient rf energy harvesting in urban and semi-urban environments," *IEEE Transactions on Microwave Theory and Techniques*, vol. 61, no. 7, pp. 2715–2726, 2013.

[172] P. Nintanavongsa, U. Muncuk, D. R. Lewis, and K. R. Chowdhury, "Design optimization and implementation for rf energy harvesting circuits," *IEEE Journal on emerging and selected topics in circuits and systems*, vol. 2, no. 1, pp. 24–33, 2012.

[173] G. Ottman, A. Bhatt, H. Hofmann, and G. Lesieutre, "Adaptive piezoelectric energy harvesting circuit for wireless, remote power supply," in *19th AIAA Applied Aerodynamics Conference*, p. 1505, 2002.

[174] B. Islam and S. Nirjon, "Poster abstract: On-device training from sensor data onbatteryless platforms," in *The 18th ACM/IEEE Conference on Information Processing in Sensor Networks*, ACM/IEEE, 2019.

[175] "Msp430fr5994," 2018. http://www.ti.com/lit/ds/symlink/ msp430fr5994.pdf.

[176] D. J. Yeager, J. Holleman, R. Prasad, J. R. Smith, and B. P. Otis, "Neuralwisp: A wirelessly powered neural interface with 1-m range," *IEEE Transactions on Biomedical Circuits and Systems*, vol. 3, no. 6, pp. 379–387, 2009.

[177] J. Holleman, D. Yeager, R. Prasad, J. R. Smith, and B. Otis, "Neuralwisp: An energy-harvesting wireless neural interface with 1-m range," in *Biomedical Circuits and Systems Conference, 2008. BioCAS 2008. IEEE*, pp. 37–40, IEEE, 2008.

[178] G. Gobieski, N. Beckmann, and B. Lucia, "Intermittent deep neural network inference," *SysML*, 2018.

[179] R. Iqdour and A. Zeroual, "Prediction of daily global solar radiation using fuzzy systems," *International Journal of Sustainable Energy*, vol. 26, no. 1, pp. 19–29, 2007.

[180] F. K. Shaikh and S. Zeadally, "Energy harvesting in wireless sensor networks: A comprehensive review," *Renewable and Sustainable Energy Reviews*, vol. 55, pp. 1041–1054, 2016.

[181] https://www.gurobi.com.

[182] K. G. Murty, *Linear programming*. Springer, 1983.

[183] G. Chartrand, A. D. Polimeni, and P. Zhang, "Mathematical proofs," *Kanada: Aptara Inc*, 2013.

[184] A. M. Van Tilborg and G. M. Koob, *Foundations of real-time computing: Scheduling and resource management*, vol. 141. Springer Science & Business Media, 2012.

[185] N. Dang, E. Bozorgzadeh, and N. Venkatasubramanian, "Energy harvesting for sustainable smart spaces," in *Advances in Computers*, vol. 87, pp. 203–251, Elsevier, 2012.

[186] S. K. Baruah, A. K. Mok, and L. E. Rosier, "Preemptively scheduling hard-real-time sporadic tasks on one processor," in *[1990] Proceedings 11th Real-Time Systems Symposium*, pp. 182–190, IEEE, 1990.

[187] G. Wang, W. Gong, and R. Kastner, "Operation scheduling: algorithms and applications," in *High-Level Synthesis*, pp. 231–255, Springer, 2008.

[188] P. Warden, "Speech commands: A dataset for limited-vocabulary speech recognition," *arXiv preprint arXiv:1804.03209*, 2018.

[189] K. J. Piczak, "ESC: Dataset for Environmental Sound Classification," in *Proceedings of the 23rd Annual ACM Conference on Multimedia*, pp. 1015–1018, ACM Press. http://dl.acm.org/citation.cfm?doid=2733373.2806390.

[190] F. Paci, L. Baraldi, G. Serra, R. Cucchiara, and L. Benini, "Context change detection for an ultra-low power low-resolution ego-vision imager," in *European Conference on Computer Vision*, pp. 589–602, Springer, 2016.

[191] N. Twomey, T. Diethe, X. Fafoutis, A. Elsts, R. McConville, P. Flach, and I. Craddock, "A comprehensive study of activity recognition using accelerometers," in *Informatics*, vol. 5, p. 27, Multidisciplinary Digital Publishing Institute, 2018.

[192] "Powercast p2210b," 2016. https://www.powercastco.com/wp-content/uploads/2016/12/P2110B-Datasheet-Rev-3.pdf.

[193] "Powercaster transmitter," 2016. http://www. powercastco.com/wp-content/uploads/2016/11/User-Manual-TX-915-01-Rev-A-4.pdf.

[194] https://tinyurl.com/UAHWiFiDataset.

[195] https://www.adafruit.com/product/700.

[196] "Ltc3105 step up dc/dc converter," 2015. https://www.analog.com/media/en/technical-documentation/data-sheets/3105fb.pdf.

[197] "Adafruit Electret Microphone," 2018. shorturl.at/yIX03.

[198] P. Wägemann, C. Dietrich, T. Distler, P. Ulbrich, and W. Schröder-Preikschat, "Whole-system worst-case energy-consumption analysis for energy-constrained real-time systems," *Leibniz International Proceedings in Informatics, LIPIcs 106 (2018)*, vol. 106, p. 24, 2018.

[199] P. Wägemann, T. Distler, T. Hönig, H. Janker, R. Kapitza, and W. Schröder-Preikschat, "Worst-case energy consumption analysis for energy-constrained embedded systems," in *2015 27th Euromicro Conference on Real-Time Systems*, pp. 105–114, IEEE, 2015.

[200] TexasInstruments, "Energy trace." `http://www.ti.com/tool/ENERGYTRACE`, 2018.

[201] M. R. Guthaus, J. S. Ringenberg, D. Ernst, T. M. Austin, T. Mudge, and R. B. Brown, "Mibench: A free, commercially representative embedded benchmark suite," in *Proceedings of the Fourth Annual IEEE International Workshop on Workload Characterization. WWC-4 (Cat. No. 01EX538)*, pp. 3–14, IEEE, 2001.

[202] S. Lee, I. Bashima, L. Yubo, and S. Nirjon, "Intermittent Learning: On-Device Machine Learning on Intermittently Powered System," in *Proceedings of ACM on Interactive, Mobile, Wearable, and Ubiquitous Computing, 2020.*, ACM, 2020.

[203] M. Yang, S. Wang, J. Bakita, T. Vu, F. D. Smith, J. H. Anderson, and J.-M. Frahm, "Re-thinking cnn frameworks for time-sensitive autonomous-driving applications: Addressing an industrial challenge," in *2019 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pp. 305–317, IEEE, 2019.

[204] S. Lee and N. Shahriar, "Subflow: A dynamic induced-subgraph strategy toward real-time dnn inference and training," in *26TH IEEE REAL-TIME AND EMBEDDED TECHNOLOGY AND APPLICATIONS SYMPOSIUM, SYDNEY, AUSTRALIA*, IEEE, 2020.

[205] B. Islam and N. Shahriar, "Scheduling computational and energy harvesting tasks in deadline-aware intermittent systems," in *26TH IEEE REAL-TIME AND EMBEDDED TECHNOLOGY AND APPLICATIONS SYMPOSIUM, SYDNEY, AUSTRALIA*, IEEE, 2020.

[206] J. W.-S. Liu, K.-J. Lin, W. K. Shih, A. C.-s. Yu, J.-Y. Chung, and W. Zhao, "Algorithms for scheduling imprecise computations," in *Foundations of Real-Time Computing: Scheduling and Resource Management*, pp. 203–249, Springer, 1991.

[207] A. Mirhoseini, E. M. Songhori, and F. Koushanfar, "Automated checkpointing for enabling intensive applications on energy harvesting devices," in *Low Power Electronics and Design (ISLPED), 2013 IEEE International Symposium on*, pp. 27–32, IEEE, 2013.

[208] Y. LeCun, C. Cortes, and C. J. Burges, "The mnist database of handwritten digits, 1998," vol. 10, p. 34, 1998. http://yann. lecun. com/exdb/mnist.

[209] K. J. Piczak, "Environmental sound classification with convolutional neural networks," in *2015 IEEE 25th International Workshop on Machine Learning for Signal Processing (MLSP)*, pp. 1–6, IEEE, 2015.

[210] A. Krizhevsky and G. Hinton, "Learning multiple layers of features from tiny images," tech. rep., Citeseer, 2009.

[211] A. Chowdhery, P. Warden, J. Shlens, A. Howard, and R. Rhodes, "Visual wake words dataset," 2019.

[212] Z. Zhang and M. Sabuncu, "Generalized cross entropy loss for training deep neural networks with noisy labels," in *Advances in Neural Information Processing Systems*, pp. 8778–8788, 2018.

[213] M. T. Islam and S. Nirjon, "Soundsemantics: exploiting semantic knowledge in text for embedded acoustic event classification," in *Proceedings of the 18th International Conference on Information Processing in Sensor Networks*, pp. 217–228, ACM, 2019.

[214] Y. Bengio, A. Courville, and P. Vincent, "Representation learning: A review and new perspectives," *IEEE transactions on pattern analysis and machine intelligence*, vol. 35, no. 8, pp. 1798–1828, 2013.

[215] "Gcc instruction timing." `http://mspgcc.sourceforge.net/manual/x528.html`, 2020.

[216] "Msp mcu iqmathlib users guide," 2015. http://www.ti.com/tool/MSP-IQMATHLIB.

[217] D. Arthur and S. Vassilvitskii, "k-means++: The advantages of careful seeding," tech. rep., Stanford, 2006.

[218] P. E. Black, "Dictionary of algorithms and data structures," tech. rep., 1998.

[219] E. Meyer, M. C. Greenwood, and T. Tran, "Daily step count profile data for 61 days [dataset]," 2016.

[220] L. Kantorovich, "On translation of mass," in *Dokl. AN SSSR*, vol. 37, p. 20, 1942.

[221] M. Moallemi and G. Wainer, "I-devs: imprecise real-time and embedded devs modeling," in *Proceedings of the 2011 Symposium on Theory of Modeling & Simulation: DEVS Integrative M&S Symposium*, Society for Computer Simulation International, 2011.

[222] B. Yang, X. Fu, N. D. Sidiropoulos, and M. Hong, "Towards k-means-friendly spaces: Simultaneous deep learning and clustering," *arXiv preprint arXiv:1610.04794*, 2016.

[223] B. George, "A study of the effect of random projection and other dimensionality reduction techniques on different classification methods," *Baselius Researcher*, p. 201769, 2017.

[224] I. S. Dhillon, Y. Guan, and J. Kogan, "Iterative clustering of high dimensional text data augmented by local search," in *2002 IEEE International Conference on Data Mining, 2002. Proceedings.*, pp. 131–138, IEEE, 2002.

[225] H. Cheng, K. A. Hua, and K. Vu, "Constrained locally weighted clustering," *Proceedings of the VLDB Endowment*, 2008.

[226] S. Basu, A. Banerjee, and R. Mooney, "Semi-supervised clustering by seeding," in *In Proceedings of 19th International Conference on Machine Learning (ICML-2002*, Citeseer, 2002.

[227] J. Schmidhuber, "Deep learning in neural networks: An overview," *Neural networks*, vol. 61, pp. 85–117, 2015.

[228] C. M. Krishna, "Real-time systems," *Wiley Encyclopedia of Electrical and Electronics Engineering*, 2001.

[229] Q. Shi, J. Petterson, G. Dror, J. Langford, A. Smola, and S. Vishwanathan, "Hash kernels for structured data," *Journal of Machine Learning Research*, vol. 10, no. Nov, pp. 2615–2637, 2009.

[230] L. De Lathauwer, B. De Moor, and J. Vandewalle, "A multilinear singular value decomposition," *SIAM journal on Matrix Analysis and Applications*, vol. 21, no. 4, pp. 1253–1278, 2000.

[231] L. De Lathauwer, B. De Moor, and J. Vandewalle, "On the best rank-1 and rank-(r 1, r 2,..., rn) approximation of higher-order tensors," *SIAM journal on Matrix Analysis and Applications*, vol. 21, no. 4, pp. 1324–1342, 2000.

[232] L. R. Tucker, "Some mathematical notes on three-mode factor analysis," *Psychometrika*, vol. 31, no. 3, pp. 279–311, 1966.

[233] "Flexible ethylene tetrafluoroethylene (etfe) based solar panel," 2020. shorturl.at/almrC.

[234] "Arducam mini," 2015. https://www.arducam.com/product/arducam-2mp-spi-camera-b0067-arduino/.

[235] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, M. Kudlur, J. Levenberg, R. Monga, S. Moore, D. G. Murray, B. Steiner, P. Tucker, V. Vasudevan, P. Warden, M. Wicke, Y. Yu, and X. Zheng, "Tensorflow: A system for large-scale machine learning," in *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, 2016.

[236] T.-Y. Lin, M. Maire, S. Belongie, L. Bourdev, R. Girshick, J. Hays, P. Perona, D. Ramanan, C. L. Zitnick, and P. DollÃ¡r, "Microsoft coco: Common objects in context," 2014.

[237] S. Xie and Z. Tu, "Holistically-nested edge detection," in *Proceedings of the IEEE international conference on computer vision*, pp. 1395–1403, 2015.

[238] R. Sharma *et al.*, "Performance evaluation of new joint edf-rm scheduling algorithm for real time distributed system," *Journal of Engineering*, vol. 2014, 2014.

[239] Z. Kons, O. Toledo-Ronen, and M. Carmel, "Audio event classification using deep neural networks.," in *Interspeech*, pp. 1482–1486, 2013.

[240] "Evaluation of esc dataset with different algorithms.," 2018. https://github.com/karoldvl/ESC-50.

[241] J. Stallkamp, M. Schlipsing, J. Salmen, and C. Igel, "Man vs. computer: Benchmarking machine learning algorithms for traffic sign recognition," *Neural Networks*, no. 0, pp. –, 2012. http://www.sciencedirect.com/science/article/pii/S0893608012000457.

[242] R. Ebrahimzadeh and M. Jampour, "Efficient handwritten digit recognition based on histogram of oriented gradients and svm," *International Journal of Computer Applications*, vol. 104, no. 9, 2014.

[243] E. Alpaydin, *Introduction to machine learning*. MIT press, 2020.

[244] C.-H. Cheng, A. W. Fu, and Y. Zhang, "Entropy-based subspace clustering for mining numerical data," in *Proceedings of the fifth ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 84–93, 1999.

[245] J. Hoffman, E. Tzeng, T. Park, J.-Y. Zhu, P. Isola, K. Saenko, A. A. Efros, and T. Darrell, "Cycada: Cycle-consistent adversarial domain adaptation," *arXiv preprint arXiv:1711.03213*, 2017.

[246] A. Mathur, A. Isopoussu, F. Kawsar, N. Berthouze, and N. D. Lane, "Mic2mic: using cycle-consistent generative adversarial networks to overcome microphone variability in speech systems," in *Proceedings of the 18th International Conference on Information Processing in Sensor Networks*, pp. 169–180, 2019.

[247] V. Liu, A. Parks, V. Talla, S. Gollakota, D. Wetherall, and J. R. Smith, "Ambient backscatter: wireless communication out of thin air," in *ACM SIGCOMM Computer Communication Review*, vol. 43, pp. 39–50, ACM, 2013.

[248] J. Elson and D. Estrin, "Time synchronization for wireless sensor networks," in *Parallel and Distributed Processing Symposium, International*, vol. 4, pp. 30186b–30186b, IEEE Computer Society, 2001.

[249] J. de Winkel, C. Delle Donne, K. S. Yıldırım, P. Pawełczak, and J. Hester, "Reliable timekeeping for intermittent computing,"

[250] "P-channel mosfet irlml5103trpbf," 2020. https://www.digikey.com/product-detail/en/infineon-technologies/IRLML5103TRPBF/IRLML5103PBFCT-ND/812496.

[251] Y. Luo and S. Nirjon, "Smarton: Just-in-time active event detection on energy harvesting systems," *arXiv preprint arXiv:2103.00749*, 2021.

[252] "https://empoweringpumps.com/everactive-batteryless-machine-health-monitoring-cost-effective-24-7-insight-for-all-rotating-equipment/."

[253] M. Masoero, C. Silvi, and J. Toniolo, "Energy performance assessment of hvac systems by inspection and monitoring," in *Proc. of 10 th REHVA World Congress Clima2010, Antalya*, pp. 09–12, 2010.

[254] J. Toniolo, C. Silvi, and M. Masoero, "Energy savings in hvac systems by continuous monitoring. results of a long term monitoring campaign on buildings.," in *International Conference on Renewable Energies and Power Quality (ICREPQ 2013)*, pp. 1–6, 2013.

[255] "https://www.epa.gov/cmop/frequent-questions."

[256] "https://www.cdc.gov/niosh/mining/topics/methanedetectionandmonitoring.html."

[257] M. Abrahamsen, A. Adamaszek, and T. Miltzow, "The art gallery problem is r-complete," in *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing*, pp. 65–73, 2018.

[258] “Pokemon.” https://en.wikipedia.org/wiki/Pokemon.