Zygarde: Time-Sensitive On-Device Deep Intelligence on Intermittently-Powered Systems

ABSTRACT

In this paper, we propose a time-, energy-, and accuracy-aware scheduling algorithm for intermittently powered systems that execute compressed deep learning tasks that are suitable for MCUs [44] and are powered solely by harvested energy. The sporadic nature of harvested energy, resource constraints of the embedded platform, and the computational demand of deep neural networks (even though compressed) pose a unique and challenging real-time scheduling problem for which no solutions have been proposed in the literature. We empirically study the problem and model the energy harvesting pattern as well as the trade-off between the accuracy and execution of a deep neural network. We develop an imprecise computing-based scheduling algorithm that improves the schedulability of deep learning tasks on intermittently powered systems. We also utilize the dependency of computational need of data samples for deep learning models and propose early termination of deep neural networks. We further propose a semi-supervised machine learning model that exploits the deep features and contributes in determining the imprecise partition of a task. We implement our proposed algorithms on two different datasets and real-life scenarios and show that it increases the accuracy by 9.45% - 3.19%, decreases the execution time by 14% and successfully schedules 33%-12% more tasks.

1 INTRODUCTION

Batteryless Systems. The Internet of Things (IoT) promises to make our lives efficient, productive, enjoyable, and healthier by making everyday objects capable of sensing, computation, and communication. Many of these so-called IoT devices are powered by limited-capacity batteries—which makes them mobile, small, and lightweight. Batteries, however, require periodic maintenance (e.g., replacement and recharging) which is an inconvenience at a large scale. To address this practical problem, batteryless IoT devices have been proposed, which harvest energy from ambient sources, e.g., solar, thermal, kinetic, and RF to power up. These devices, in principle, last forever—as long as the energy harvesting conditions are met. They typically consist of low-power sensors, microcontrollers, and energy-harvesting and management circuitry, and their applications are in many deploy-and-forget scenarios, e.g., wildlife monitoring, remote surveillance, environment and infrastructure monitoring, wearables, and implantables.

Time-Aware Inference. Many IoT applications require timely feedbacks. For instance, in an acoustic monitoring system, audio events need to be detected and reported as fast as possible to initiate prompt actions. Similarly, an air-quality monitoring system needs to identify the increase of a certain air component on time for taking proper actions. Likewise, shared resources, such as gym-equipment and shared bikes in a campus, can be monitored in real-time to detect misuse or malfunctions, and to inform the authority about the incidence on time. While a batteryless system is desirable in these real-time sensing and event detection applications, the unpredictability of the harvested energy, combined with the complexity of on-device event detection tasks, complicates timely execution of machine learning-based event detection tasks on batteryless systems.

Prior Work on Timeliness. Prior works on time-aware batteryless computing systems can be broadly categorized into two types. The first category focuses on time-keeping, i.e., maintaining a reliable system clock [53, 80] even when the power is out. The sporadic nature of an energy harvesting system forces it to run intermittently by going through alternating episodes of power ON and OFF phases which disrupts the continuity of the system clock. By exploiting the rate of decay of an internal capacitor and the content of the SRAM, these systems enable time-keeping during the absence of power. The second category proposes runtime systems that consider the temporal aspect of data across power failures [26, 52, 105, 109]. For instance, [52] discards data after a predefined interval and thus saves energy by not processing stale data, and [26, 105, 109] propose energy-aware runtime systems to increase the chances of task completion. However, none of these consider the utility of data or exploit the property of inference tasks to plan real-time deadline-aware execution of tasks.

Real-Time Intermittent Computing. Scheduling time-aware machine learning tasks on a batteryless computing system is an extremely challenging feat. The two main sources of challenges are the intermittent power supply and the computational demand for executing machine learning tasks. These two challenges have been studied extensively in non-real-time settings. For instance, [18, 19, 31, 67, 68, 70, 82] enable seamless execution of non-real-time tasks on intermittently powered systems by proposing techniques that save and restore the program states across power failures. [43, 44, 56, 77] propose lightweight and compressed deep neural network inference for on-device machine learning on batteryless systems. However, none of these works consider the timing constraints of...
the machine learning tasks. In a real-time setting, simply applying these two types of solutions in conjunction with an existing real-time scheduling algorithm does not quite solve the problem at hand, which is illustrated in Figure 1. We consider two tasks, $τ_1$ and $τ_2$, released at time 0 and 25, respectively. Their deadlines are 45 and 56, and both have an execution time of 28. In Figure 1(a), we observe that under the earliest deadline first (EDF) scheduling, both tasks meet the deadlines when the power is uninterrupted. But when power is intermittent, Figure 1(b) shows that task $τ_2$ misses its deadline.

**Observations.** The goal of this paper is to overcome the aforementioned challenges. Towards this end, we study the energy harvesting pattern and the accuracy-execution trade-off of compressed deep neural networks (DNNs) that are executable on small systems [44]. From these studies, we make two observations:

- First, energy generated by a harvester is bursty, and therefore, its energy harvesting pattern can be modeled using a stochastic framework over a short duration. Here, burstiness indicates that energy generation is maintained during a short period.
- Second, since deeper layers of a DNN extract fine-grained and more detailed features of the input, for a given accuracy, the amount of DNN computation required for an input data is dependent on the quality of data itself.

**The Zygarde Approach.** By exploiting these above two observations, we design an imprecise computing [63, 89], online scheduling algorithm that considers both the intermittent nature of the power supply and the accuracy-execution trade-off of the DNN model. This design allows us to increase both accuracy and timeliness of DNN execution in intermittent systems. For example, in Figure 1(c), when a full execution of the task is not necessary, i.e., tasks are imprecise, both tasks can meet the deadline.

Our work complements previous work on time-keeping [53, 80] and intermittent execution of non-real-time tasks [18, 19, 31, 43, 44, 67, 68, 70, 82]. We extend the state-of-the-art intermittent DNN execution framework, called SONIC [44], by implementing a runtime framework that supports intelligent scheduling of real-time DNN tasks. To enable time-aware adaptive deep learning in intermittently powered systems, we make three key technical contributions:

- First, we devise a metric, i.e., the $η$-factor, to model the predictability of an energy harvester’s source. This metric indicates the probability of a harvester maintaining its current state over a short period in time. The introduction of $η$-factor abstracts away the unpredictability of an energy harvesting source and enables development of scheduling algorithms that can make informed decisions based on predicted energy over a short period in the future.
- Second, we redesign the DNN construction and training algorithms to enable early termination of a DNN task based on the quality of the input data. To enable this, we propose a layer-aware loss function to improve the accuracy of a clustering-based, semi-supervised inference algorithm that uses DNN layer outputs as the representation of the input examples.
- Third, we propose an imprecise computing-based, online, scheduling algorithm that improves the timeliness of DNN inference tasks running on energy harvesting systems. This algorithm leverages the $η$-factor of the energy source, along with the properties of the input data to adapt the execution of real-time DNN tasks.

**Main Results.** We implement the system on a TI MSP430FR5994 and evaluate its performance using two datasets: MNIST [61] and ESC-10 [79], as well as in real-world acoustic event detection experiments. We achieve 9.45% - 3.19% higher accuracy than the baseline DNN algorithms for MCUs and achieve 14% reduction in execution time. The proposed scheduling algorithm successfully schedules 33%-12% more tasks than traditional scheduling algorithms.

**2 MOTIVATION**

**On-Device Learning.** Resource-constrained sensing and inference systems enjoy the benefit of machine learning in two ways — either they send raw or partially processed sensor readings to a remote server for inference, or they do everything on device. While most low-power IoT systems largely use the former method [28, 30, 37, 86, 98], in recent years, we see an increasing trend in on-device machine learning on embedded systems [9, 10, 24, 84]. This can partly be attributed to the limitations of server-based systems — which are generally energy-demanding, slow, less reliable, and privacy invasive. The other reason behind this increasing trend is the advancement in hardware and software technologies [9, 11, 14, 15] that are enabling powerful machine learning features into small and low-power systems.

**Learning on Batteryless Systems.** Previous works have addressed the need for computation including inference of deep neural network on energy harvested devices [31, 43, 44, 67, 82]. However, the necessity of updating the models and timeliness is yet to be explored. As no two scenarios or people are genuinely similar, the need to have a customized model is inevitable. Besides, the models of a forever executing batteryless system become outdated with time. To address the same issue, on-device learning (including training) is introduced [9], which improves accuracy and provides a personalized system that ensures privacy, lower latency, and reliability. Moreover, most batteryless systems are deployed at unreachable places (e.g., deep jungles, calamity-prone areas) where the traditional power source is absent and replacing a battery is unrealistic. It is also not feasible to collect data and update the model on-site in such cases. With time, such devices observe a massive amount of information which is likely to be wasted due to the inefficiency of data transmission. To fully utilize the potential of batteryless devices, training the models is needed. However, the small footprint of batteryless devices works against the extra computational and energy load enforced by training models and the unavailability of labeled data. Light-weight semi-supervised learning algorithms need to be considered to evolve previously trained models with the incoming data stream.

**Deep Inference.** Due to its non-linear and parametric model, Deep Neural Network (DNN) exhibits better performance than other traditional models, e.g., Support Vector Machine (SVM). Here, DNN refers to neural networks with more than one hidden layer. Sending the inference result instead of the raw data is more energy efficient and high accuracy is essential to maintain the system’s usability. [44] shows that inference accuracy determines the end-to-end system performance.
Time-Aware Inference. The higher accuracy of DNN is achieved using more rigorous computations that result in higher execution and response time. For a usable system, the response time needs to be tolerable [72, 73, 109]. High delay of a system hampers its usability despite accuracy. Though some works have addressed real-time requirements for batteryless sensor nodes, the accuracy of the system itself has not been considered yet. Precision and responsiveness both are crucial for the usability of a learning system and are contradictory to each other. The goal is to find the sweet spot where the highest accuracy can be achieved with acceptable delay. Such a time-aware application includes event detection and monitoring wildlife, natural calamities, wearable, implantable, infrastructure, and buildings. To illustrate, an acoustic event detector at home enables home activating monitoring, intruder detection, and elderly monitoring. Gym equipment and shared bike usage can be monitored using kinetic energy harvesting systems which can inform the authority about the required maintenance of the system.

3 SYSTEM MODELING

In this section, we study and model the energy harvesting pattern of energy harvesters and the accuracy-execution trade-off of deep neural networks.

3.1 Modeling Energy Harvesting Pattern

Energy Events. Transiently powered systems operate intermittently because energy is not always available to harvest and, even when energy is available, buffering sufficient energy to perform adequate work takes time. In most cases, the pattern of this intermittency is stochastic and thus modeling this pattern is not straightforward. To schedule the workload of an intermittently operating system at run-time, we decide whether to start execution of a task or not at a time instant. This decision heavily depends on the availability of harvestable energy. To model the availability of energy, we define energy event which expresses the availability of sufficient energy during a period. Energy event represents a successful generation of at least K Joules of energy in total during T time slot. Here, K and T are system dependent. In order to better understand the property of energy events, we observe the phenomena causing energy events. For example, in a piezo-electric harvester, taking a minimal number of steps that generates at least K Joule of energy during T time slot is considered equivalent to the occurrence of an energy event. Similarly, we consider a minimal number of packet transmissions per time slot and minimum intensity of solar per time slot as energy events for RF and solar harvesters, respectively.

Properties of Energy Events. We study energy event patterns of three commonly used harvesters – piezo-electric harvester, solar harvester, and RF harvester from datasets [4, 69]. These datasets contain the number of steps taken during every 5-minute time-slot for 61 days, harvested solar energy measurements for three days and outbound packet transmission rate by an RF transmitter for 61 days, harvested solar energy measurements for three days and outbound packet transmission rate by an RF transmitter for 30 days. This study reveals two interesting observation about the pattern of energy events – (1) energy events occur in bursts where burstiness is the intermittent increases and decreases in activity or frequency of an event [17], (2) a probabilistic relation exists among the consecutive energy events during a short period. In other words, the occurrence of an energy event increases the probability of the next energy event during a short period. To illustrate, when a person starts walking, the probability of continuing the walk is high within the first few time slots and it decreases with time. Likewise, when a person is sitting, the probability of remain seated is high immediately, but decreases after a while.

Conditional Energy Event. We define conditional energy event (CEE) that represents the conditional probability of an energy event occurrence based on the occurrence/absence of previous consecutive energy events. CEE(N) is the probability that an energy event will occur given immediately preceding N consecutive energy events occurred (for N > 0) or not occurred (for N < 0). The following equation expresses CEE.

\[
CEE(N) = \begin{cases} 
\frac{p(\text{occurrence} \mid \text{N consecutive occurrence})}{p(\text{occurrence} \mid \text{N consecutive non-occurrence})}, & \text{if } N > 0 \\
\frac{p(\text{occurrence} \mid \text{N consecutive occurrence})}{1}, & \text{if } N < 0 
\end{cases}
\]

To illustrate CEE(10) = 90% implies that the next energy event will occur with 90% probability if 10 immediately preceding consecutive energy event occurred. Similarly, CEE(-15) = 5% indicates the probability of an energy event at the current time slot is 5%, given that there were no energy events in the last 15 slots.

The CEE of a system powered by a persistently power supply or an ideal harvester that has no intermittence looks like Figure 2(a). Figure 2(b-d) shows the CEE of three energy harvested systems. From these figures, we observe that for a small value of N these systems demonstrate similarity with the ideal correlative harvester. We measure the similarity of CEE of a harvested system with persistent powered or ideal harvested system using Kantorovich-Wasserstein distance [81]. Through out this paper, we use KW to express the Kantorovich-Wasserstein distance between the CEE a system (H) and the CEE of persistently powered system (P) as given in Equation 2. We also observe that for a large [N] the CEE drops significantly because when the interval time between the first and
current event increases their probabilistic relation decreases. For example, a person is walking for a long time has a high probability of stopping.

\[ KW(H, P) = \int_{-\infty}^{+\infty} |CDF(H) - CDF(P)| \]  

\( \eta \) Factor. To quantify the predictability of a harvester, we take inspiration from [91] and define a predictability metric \( \eta \). Despite being informative, distance alone is not sufficient to measure predictability. Because distance does not address the imbalance between the number of elements in CEE with positive and negative N values. To address this, we quantify \( \eta \) as the distance of the harvester (H) from a persistently powered system compared to the distance of the random harvester from a persistently powered system with same energy event rate. Random harvester (R) refers to harvesters where energy events are completely independent. \( \eta \) is expressed as follows.

\[ \eta = \frac{KW(R, P) - KW(H, P)}{KW(R, P)} \]  

\( \eta = 1 \) indicates that the power is persistent, while \( \eta = 0 \) indicates energy events are totally random. This metric also depends on the number of consecutive energy events. As CEE(N) is close to zero for high \( |N| \), we consider small \( |N| \) throughout the paper. Figure 2(b-d) shows the \( KW \) and \( \eta \) where \( |N| \) is 20.

\( \eta \) not only varies across different harvesters but also changes for a specific harvester. For transiently powered devices this change over time depends on different parameters, e.g., change of human weight, change of seasons or locations, the distance between RF transmitter and receiver. However, due to the infrequency of these changes, they can be ignored.

### 3.2 Modeling Deep Neural Network

In order to execute machine learning tasks (e.g., deep neural network (DNN) inference with convolution and fully connected layers) in a resource-constrained batteryless system, we need to minimize memory and computation costs. To achieve this goal, we study several attributes that are unique to deep learning processes.

**Significance of Depth.** DNN have layered structures where the input of the first layer is from an external source, e.g., sensors. The output of a layer is fed as the input of the next layer, and it goes on until the end of the network. The total number of layers in a DNN is called depth. A shallow neural network, e.g., first perceptrons are composed of one input layer, at most one hidden layer and one output layer. A neural network having more than three layers (including input and output) qualifies as “deep” neural network. Increased number of layers and neurons both contribute to more complicated calculation resulting in higher accuracy [47]. However, a shallow network requires width exponential to that of a deeper network to achieve similar accuracy. Therefore, the performance of a neural network depends on not only the number of parameters but also depth. For example, VGGNet has 16 layers with 140M parameters, while ResNet beats it with 152 layers but only 2M parameters.

The depth of a DNN is significant because at each layer nodes train on a distinct set of features based on the output of the previous layer. The complexity and abstraction of features increase with the depth of layers, and it is known as the feature hierarchy. Deep learning can extract features from data without human intervention. This automatic feature extraction is known as representation learning. To understand the effect of the depth of a DNN, let us consider a face detector in Figure 3(a). The first layer of this deep learning based face detector learns basic features, e.g., edges. In the next layer, a collection of edges, e.g., nose, is learned. A deeper layer learns a higher level feature, e.g., face abstraction.

**Required Depth.** To decrease the execution time we observe the fact that depth is highly data-dependent [25]. If target classes are profoundly distinctive, then simple features can be used to distinguish them. For example, in Figure 4, audio of cat and water (easy data) are very distinguishable; thus a single layer CNN achieves 93% classification accuracy. On the other hand, similar classes, e.g., train and helicopter (hard data) needs more complex representations to be distinct and thus require five layers of CNN to achieve 81% accuracy. Representation learning uses deep neural network to extract features from raw data. By executing only necessary layers based on the complexity of data we can achieve similar accuracy with decreased execution time. We consider execution of each data sample as a task and introduce the requirement of depth as a imprecise task model [27, 63, 89]. Each imprecise task consists of two portions – mandatory and optional. Mandatory portion of a task is necessary to achieve required accuracy while executing optional portions further improves the performance. We consider the required depth for a data sample as mandatory portion and rest as optional.

**DNNs for Ultra-Low-Power Systems.** Fitting neural networks in resource-constrained energy harvesting systems is a challenge especially due to limited memory capacity. Most commonly used
processors in existing intermittent systems are TI MSP430 low-power microcontrollers [33, 44, 49–51] that include 1-4KB of SRAM and 32-256KB of FRAM. Therefore, only small compressed networks are suitable for these systems. Compressed networks mentioned in Table 1 and previous work [44] have 48,136-10,411 parameters. For 16-bit fixed point data type these parameter requires 96.3-20KB of memory. On the other hand, same networks require 943.66-180KB of memory without compression. Such networks can be executed in batteryless systems only after compression. However, larger networks e.g., Resnet (image) and EnvNet (audio) requires 5MB and 94MB of memory which is not suitable for small memory footprint devices even after compression.

We execute MNIST(compressed) in TI MSP430 to estimate the size of code and other variables in the FRAM. We observed that instructions and other parameters (including a buffer to perform matrix operations) require around 128KB of memory. Though the increasing number of parameters increases the buffer size, for simplicity we ignore it. The remaining memory can be used for storing approximately 64,000 16 bit fixed point parameters. A deep neural network that requires less than 64,000 parameters can execute in a microcontroller having 256KB of FRAM. The number of hidden layers and the number of neurons at each layer depend on the number of input and output [48]. Therefore, very large dataset e.g., Imagenet is not suitable for such systems. These conditions are sufficient for executing 4-6 layer networks depending on the network configuration [44].

This paper considers two networks summarized in Table 1. MNIST [61] represents the image-based application, and ESC-10 [78] represents audio applications. We use two known techniques – rank decomposition or separation [22, 29, 38, 39, 95, 99] and pruning [46, 75] to compress each layer of the networks. Note that, our semi-supervised models avoid the last layers for inference.

| Dataset | Layer | Uncompressed Size | Compressed Size |
|---------|-------|-------------------|-----------------|
| MNIST   | Convolution 20×1×5×5 | 3x1D Conv | 1253 |
|         | Convolution 100×20×5×5 | | 5456 |
|         | Fully Connected 200×1600 | | 1892 |
|         | Fully Connected 500×200 | | – |
|         | Fully Connected 10×500 | | – |
| ESC-10  | Convolution 16×1×5×5 | 3x1D Conv | 1280 |
|         | Convolution 32×16×5×5 | | 5068 |
|         | Convolution 64×32×5×5 | | 2703 |
|         | Fully Connected 96×256 | | – |
|         | Fully Connected 10×96 | | – |

Table 1: Networks considered in this paper.

4 ZYGARDE SYSTEM DESIGN

Zygarde is a system architecture that executes semi-supervised deep learning with timing constraints in an intermittently powered system. It uses a deep neural network to extract complex features from data samples, where the deep neural network is pre-trained on a high-end device. Zygarde adapts to new unlabeled incoming data by using semi-supervised models, e.g., seed-based k-means [29]. In seed-based k-means, initial centroids are defined from labeled data in training phase and at runtime the unlabeled data update these centroids. Zygarde relies on imprecise computing to maximize the number of samples meeting timing constraints in a batteryless platform.

Zygarde aims to achieve three goals simultaneously – (1) minimize classification error, (2) maximize the number of samples meeting timing constraints, and (3) minimize energy waste. Zygarde addresses this by terminating the deep feature extraction network early when necessary and scheduling the samples with a special online scheduling algorithm for batteryless system that considers time, classification error and availability of energy simultaneously.

4.1 System Components

Zygarde consists of five major components – task generator, energy manager, agile DNN model, scheduler and adaptive models.

- Task Generator. Zygarde gathers data from sensors (e.g. microphone, camera) and considers each data sample as a task. A task includes inference of agile DNN model and semi-supervised learning using adaptive model with adaptation for a data sample. To sense, it takes advantage of the analog to digital converter (ADC) and direct memory access (DMA) which writes sensor data to non-volatile memory without occupying the CPU. Each task (data sample) is pushed in the task queue upon arrival. Each task contains two portions – mandatory and optional. Only tasks in the task-queue are considered for execution and a task leaves the queue at the end of execution or at deadline.

- Energy Manager. The energy manager monitors the status of the energy storage (e.g. capacitor) and the energy harvesting rate. To measure the energy harvesting rate it relies on the system operating voltage and the voltage across the capacitor. These parameters are fed to the scheduler that determines whether to execute a task depending on these parameters. When energy is less than a minimum threshold ($E_{man}$), power failure occurs and nothing gets executed. We use SONIC [44] to handle intermittent execution of tasks in this system.

- Agile DNN Model. Agile DNN model is a pre-trained feature extraction deep neural network. This deep neural network is trained with labeled data to extract the features from data for semi-supervised learning. This network is trained in a high-end device (e.g. server or GPU). We compress the trained network using rank decomposition and separation to fit in memory-constrained systems. To achieve better classification/clustering accuracy in the earlier layer of the network, we propose a layer-aware loss function. The goal of this loss function is to extract distinctive features in the early stages of the network if possible.

- Adaptive Models. Adaptive models are a set of seed-based k-means models. These models classify the sensor data by using the
features extracted from the Agile DNN models. To select only useful features and to decrease the model size, these models use features with highest Chi-squared stats. Adaptive models are incrementally updated to evolve with new data and adapt to dynamic environment. These models are utilized to determine confidence of accuracy (utility) to exit the DNN layers.

- **Scheduler.** The scheduler decides which task to process and partitions the task based on utility. This utility decides if the data sample requires further processing for more confident decision making. The scheduler uses this mandatory and optional segments, achieved confidence (utility) and energy status from the energy manager to decide which data sample to process. When a function of current energy of the system ($E_{curr}$) and $\eta$ is less than a threshold ($E_{opt}$), Zygarde becomes conservative in its choice for execution and considers only mandatory portions of tasks. The high probability for low energy harvesting and possible power failure in the near future leads to this decision. Otherwise Zygarde considers both mandatory and optional portions for execution.

### Table 2: Task Description

| Task | Total Layers | Mandatory Layers | Optional Layers | Release Time | Deadline |
|------|--------------|------------------|-----------------|--------------|----------|
| $\tau_1$ | 4            | 1                | 3               | $t_1$        | $t_7$    |
| $\tau_2$ | 4            | 2                | 2               | $t_3$        | $t_9$    |

| Time | Reason of the Action |
|------|----------------------|
| $t_0$ | No task in the system. |
| $t_1$ | $\tau_1^1$ (the only task) gets scheduled. |
| $t_2$ | Since $E_{curr} < E_{opt}$, optional $\tau_2^1$ is not scheduled. |
| $t_3$ | System prioritized $\tau_2^2$ over $\tau_1^3$. (See: Section 5). |
| $t_4$ | Since $E_{curr} < E_{man}$ no task is scheduled. |
| $t_5$ | System prioritized mandatory $\tau_2^2$ over optional $\tau_1^1$. |
| $t_6$ | Only optional tasks remain and $E_{curr} > E_{opt}$. The system prioritizes $\tau_3^1$ over $\tau_1^3$ due to its tighter deadline. |
| $t_7$ | $\tau_2^3$ (the only task) gets scheduled. |
| $t_8$ | $\tau_2^3$ (the only task) gets scheduled. |

### Table 3: Description of Figure 6

| Time | Reason of the Action |
|------|----------------------|
| $t_0$ | No task in the system. |
| $t_1$ | $\tau_1^1$ (the only task) gets scheduled. |
| $t_2$ | Since $E_{curr} < E_{opt}$, optional $\tau_2^1$ is not scheduled. |
| $t_3$ | System prioritized $\tau_2^2$ over $\tau_1^3$. (See: Section 5). |
| $t_4$ | Since $E_{curr} < E_{man}$ no task is scheduled. |
| $t_5$ | System prioritized mandatory $\tau_2^2$ over optional $\tau_1^1$. |
| $t_6$ | Only optional tasks remain and $E_{curr} > E_{opt}$. The system prioritizes $\tau_3^1$ over $\tau_2^3$ due to its tighter deadline. |
| $t_7$ | $\tau_2^3$ (the only task) gets scheduled. |
| $t_8$ | $\tau_2^3$ (the only task) gets scheduled. |

### 4.2 Example Execution

We describe a simple workload consisting of real-time inference of two examples and demonstrate how Zygarde executes the workload. We describe the tasks in Table 2, where $\tau_i$ refers to task $i$. Figure 6 demonstrates execution of task $\tau_1$ and $\tau_2$ along with energy status. Here, $\tau_i^j$ refers to $j^{th}$ layer of task $i$. Table 3 provides the actions and reasons of the actions taken at each time step. Note that this example uses simplified assumptions (e.g., each layer requires single time unit to execute). The algorithm mentioned in Section 5 handles complexities (e.g., different execution time for each layer, multiple time units per layer, unknown mandatory layer number and, power failure during layer execution).

### 5 REAL-TIME SCHEDULER

This section describes a generic real-time task scheduler for intermittently powered systems where each task executes as a chain of sequential subtasks that can be partitioned into mandatory and optional parts. We define the task model and task prioritization metric, and describe the scheduling algorithm.

#### 5.1 Task Model

**Define Task.** We define each data sample entering Zygarde as an imprecise task$^1$ [63, 88], $\tau$. Data samples enter Zygarde in a sporadic manner and multiple data samples/tasks can exist at any time point. The $i^{th}$ task is defined at $t_{\tau_i}$ where $D_i, e_i^j$ are the deadline of task $\tau_i$ and execution time of $j^{th}$ subtask of task $\tau_i$. The subtasks maintain a strict precedence order. An imprecise task $\tau_i$ is divided into two portions – mandatory and optional. According to the definition of imprecise scheduling, successful execution of the mandatory portion within deadline is considered as schedulable [63, 88]. Figure 7 shows the task model of a task $\tau_i$. Each subtask consists of multiple units that execute atomically in a batteryless system that maintain the precedence order. A unit is similar to a task in task-based intermittently powered models mentioned in [32, 43, 44, 67] that needs to be restarted if there is a power down before it finishes. The scheduler of Zygarde works with the subtasks and the units are maintained by SONIC [44].

**Utility and Runtime Task Partitioning.** The utility is an application specific parameter that indicates the system’s goal. For example, in a control system, the completion of a correct control task results in maximum utility. The utility of a task has a non-decreasing and non-linear correspondence with execution of each subtask. Mandatory portion of a task contains the subtasks which need to be executed to achieve a minimum utility. The rest of the subtasks belong to the optional portion. Unlike traditional imprecise computing [63, 88] where the number of subtasks in the mandatory

$^1$ According to the definitions in the real-time systems community, we should call each data sample a job. However, as each task has only one job in this system, we use task for each data sample for simplicity.
portion is a pre-knowledge, the number of subtasks in the mandatory portion of Zygarde is determined at run-time. We define such imprecise computing model as Dynamic Imprecise Computing.

**Preemption and Task Switching.** Each subtask performs a semantically integrated operation and cannot be preempted by the scheduler. However, the scheduler is allowed to preempt a task at the end of each subtask. This task-model follows the cooperative preemption task model at the subtask level [74]. Note that, a subtask can be preempted due to a power failure by the energy manager, but this is not related to the scheduler.

### 5.2 Scheduling for Persistent Systems

Before we introduce the scheduling algorithm for an intermittently powered system, we discuss the scheduling algorithm for a persistently powered system. We consider that the CPU utilization [106] can be higher than one because in the intermittent system the power failure virtually blocks the CPU and increases the CPU utilization. As theoretically no scheduler can schedule all the tasks when the CPU utilization greater than one [106], our goal is to maximize the number of tasks that can be scheduled.

In order to schedule dynamic imprecise tasks online, we propose a priority function ($\zeta$) which considers both the deadline and the utility of tasks. It also considers the effect of mandatory and optional portions which are crucial for imprecise scheduling. We define the priority function as following:

$$
\zeta = (1 - \alpha(D_t - t_c)) + (1 - \beta U_i) + \gamma 
$$

Here, $D_t$ and $U_i$ is the deadline and current utility of the task $t_i$ respectively, $t_c$ is the current time, and $\alpha$ and $\beta$ are the scaling factors. Finally, $\gamma$ is the imprecise factor which defines if a task $t_i$ is currently executing a mandatory subtask or an optional one. The following equation expresses the imprecise factor ($\gamma$).

$$
\gamma = \begin{cases} 
1, & \text{mandatory portion (} U_i < U_t \text{)} \\
0, & \text{optional portion (} U_i \geq U_t \text{)} 
\end{cases}
$$

Here, $U_t$ is the utility threshold that indicates the end of the mandatory portion. Imprecise factor guarantees the precedence of the mandatory portions before the optional portions.

| Task | Release Time | Total Layers | Task Set 1 | Task Set 2 |
|------|-------------|-------------|------------|------------|
|      |             |             | Mandatory Layers | Deadline | Mandatory Layers | Deadline |
| $t_1$ | 0           | 7           | 2           | 45        | 5           | 59       |
| $t_2$ | 25          | 7           | 4           | 56        | 2           | 69       |
| $t_3$ | 50          | 7           | 3           | 93        | 1           | 84       |

**Table 4: Task Description**

![Figure 8: Subtask execution time](image)

**Example.** We consider two sets of three tasks described in Table 4. We consider accumulated accuracy as utility function. Each task has

![Figure 9: (a) EDF fails to meet imprecise deadline. (b) Priority function meets imprecise deadline (c) EDF meets deadline with 80% accuracy (d) Priority function meets deadline with 88.7% accuracy.](image)

seven subtasks with execution time shown in Figure 8. Figure 9(a-b) shows the execution of tasks from set1 using EDF and priority based scheduling algorithm. In Figure 9(a), EDF fails to schedule the second task; however, priority based scheduler can schedule all three tasks in Figure 9(b). In Figure 9(c-d) we consider tasks from set2. Even though both EDF and priority function succeeds to schedule all the tasks, the accumulated accuracy of the EDF schedule (80%) is higher than that of priority function schedule (88%).

### 5.3 Scheduling for Intermittent System

Scheduling in intermittently powered system is challenging due to the occurrence of power failure. In Section 3.1, we introduce a probability of metric $\eta$ to measure predictability of an energy harvester. We use $\eta$ influenced priority function ($\zeta_I$) to schedule an intermittent system.

$$
\zeta_I = \begin{cases} 
(1 - \alpha(D_t - t_c)) + (1 - \beta U_i) + \gamma, & \eta E_{curr} \geq E_{opt} \\
\gamma((1 - \alpha(D_t - t_c)) + (1 - \beta U_i)), & \eta E_{curr} < E_{opt} 
\end{cases}
$$

Here, $E_{curr}$ is the current energy generation rate and $E_{opt}$ is the threshold energy generation rate. For an energy harvester with high $\eta$, we leverage the predictability of energy generation and boost the utilization. When the $E_{curr}$ is high, we schedule both mandatory and optional subtasks, opportunely taking advantage of correlated energy event occurrence. Otherwise, the scheduler considers the high probability of energy event non-occurrence and schedules conservatively. In this case we only schedule the mandatory subtasks. When $\eta$ is low, the predictability of energy generation is minimal and thus the system schedules conservatively unless the energy generation is very high.

$\zeta_I$ minimizes two types of energy waste in energy harvesting systems [26]. The first one is running unnecessary tasks which we avoid when $\eta E_{curr} < E_{opt}$. The second wasteage occurs by not running tasks when the harvester is getting continuous energy from the source to keep the capacitor charged. We handle this waste by running optional subtasks when $\eta E_{curr} \geq E_{opt}$.
5.4 A Special Case: Scheduling Deterministic Intermittent System.

Deterministic intermittent power source is a special type of intermittent source where energy harvesting pattern is known, e.g., RF harvester with periodic signal transmission. We propose a more simplistic approach for such systems. To schedule in such a system, we pretend this energy intermittence as a hypothetical periodic task (termed energy task) with the highest priority, and it is prescheduled offline. This assumption allows us to schedule the tasks with energy constraints taken in to account. Note that we assume that energy task can preempt the conditional-preemptive tasks at any time instance.

![Energy Task Diagram](image)

Figure 10: (a) Missed a imprecise deadline of the job by not taking the deterministic intermittence power into account. (b) Successfully scheduled two tasks by considering deterministic energy pattern as energy tasks.

In Figure 10, we consider the first two tasks ($\tau_1$ and $\tau_2$) from Figure 9(a-b). We consider a periodic power source with a period of 8 time units. In Figure 10(a), the system does not consider the energy intermittence and misses the imprecise deadline for $\tau_2$. However, by considering the deterministic power source as energy tasks, the system schedules both tasks successfully.

6 AGILE DNN AND MODEL ADAPTATION

In this section, we first describe the task model for semi-supervised learning with deep neural features. Then we describe the construction of agile DNN Model and approximate adaptation of adaptive models.

6.1 Agile DNN Task Model

Define Task. We define each data sample entering Zygarde as a task. The execution of one layer of agile DNN along with corresponding k-means model is defined as a subtask. In a subtask, Zygarde extracts features of the data sample from a specific layer of the agile DNN model, use those features to execute a semi-supervised k-means model and update the centroids. Figure 11 shows the task model of a task. $\tau_i$, $\tau_{i-1}$, $\tau_i$ and $\tau_{i+1}$ are the subtasks of $\tau_i$ that represent the $k - 1$, $k$ and $k + 1$ layers of the DNN with corresponding semi-supervised k-means clustering, respectively.

Figure 11: Flow of agile DNN task $\tau_i$ where $k - 1$, $k$ and $k + 1$ are different DNN layers (subtasks). $U_{i-1}^k$, $U_{i}^k$ and $U_{i+1}^k$ are the utility of $\tau_i$ after execution of $k - 1$, $k$ and $k + 1$ layers.

Early Termination. For an energy constraint system, computing accuracy of unlabeled data is expensive. Popular model validation techniques including inter-intra cluster distance measure are computation heavy [56]. Therefore, instead of using accuracy as utility, we propose a light weight utility function. We define the difference between the distances of the data point from two nearest centroids as utility.

$$U_i = |d_a - d_b|$$ (7)

Here, $d_a$ and $d_b$ are the distance of the data sample from the closest and the second closest centroid. The intuition behind this definition is that a data point at similar distances from two centroids is not confident enough regarding which cluster it belongs to. Therefore, more complex representation is required to determine the cluster with confidence. In Figure 12, the distance between the data sample and two nearest centroids, $c_1$ and $c_3$ are $d_1$ and $d_3$ respectively. In Figure 12(a) the difference between $d_1$ and $d_3$ are very small and thus the confidence of the data sample being a member of cluster 3 is low. Therefore more complex representation is needed to provide more confident result and further execution of agile DNN is needed. On the other hand, in Figure 12(b) the difference between $d_1$ and $d_2$ are prominent and thus the data sample belongs to cluster 3 with high confidence. Thus further execution is not needed. Figure 11, shows the early termination policy and the execution of subtasks. Here, $U_{i}^{k-1}$, $U_{i}^{k}$ and $U_{i}^{k+1}$ are the utility of $\tau_i$ after execution of $k - 1$, $k$ and $k + 1$ layers. $U_t$ is the threshold utility.

6.2 Agile DNN Construction

Our termination policy utilizes the output vector at each layer as learned representation rather than solving a joint optimization problem to perform both classification and clustering at the last layer [100]. To ensure higher utility, we need to learn a representation which maximizes the distance between the representation of different classes and also minimizes the space between the representation of the same class. Contrastive loss function obtain this at the
final [57]. However, easier samples can be distinguished with simpler representation from previous layers. Thus, we need to achieve distinctive representation in earlier layers.

**Layer-Aware Loss Function.** In order to accomplish this, we propose a layer-aware loss function inspired by the contrastive loss [57]. We use a convex combination of contrastive loss at each layer as a loss function that allows the system to have better distinguishable representation at the preceding layer. To ensure distinctive representations at earlier layers, early layers have more weights than the deeper layers. Therefore, easy samples get enough distinguishable representation at an earlier layer of a network migrating their need to execute all the layers. Note that this loss function is used during the training of the agile DNN model in a traditional deep learning computing system.

The layer-aware loss function, $l$ is represented as follows.

$$l = \sum_{i=1}^{L} a_i \times l_i(W_i, X^1_i, X^2_i, ..., X^N_i); \text{where}, \sum_{i=1}^{L} a_i = 1 \quad (8)$$

Here, $a_i$ is the convex coefficient at layer $i$. $L$ and $N$ represent the total number of layers and classes in the network respectively. $W_i$ is the learnable weights of the network at layer $i$. $X^1_i, X^2_i, ..., X^N_i$ are the vectors of each class at layer $i$. $l_i$ is the contrastive loss function and for two classes $X^1_i$ & $X^2_i$ at layer $i$ it is defined as following.

$$l_i(W_i, X^1_i, X^2_i) = \frac{(1-Y)}{2} G_W(X^1_i) \Delta G_W(X^2_i)$$

$$+ \frac{Y}{2} \max(0, \Delta - G_W(X^1_i) \Delta G_W(X^2_i)) \quad (9)$$

Here, $G_W(X^j_i)$ is the representation output of a member of class $j$ where $j = 1, 2, ..., N$, at layer $i$. Coefficient $Y = 0$ if $X^1$ and $X^2$ belong to the same class and $Y = 1$ otherwise. The term $\Delta$ represents the distance margin that is maintained between the representation of different classes.

**Adaptation.** Due to the prior termination in previous layers, a group of data fails to affect the clustering model with complex representation from deeper layers. It hinders the adaptation of cluster model with more complex features. One solution is to execute the non-linear calculation for each sample to get more complex representation and update the models. This calculation includes matrix multiplications, addition, and a non-linear function; e.g., RELU activation. However, this execution contradicts to our goal of avoiding unnecessary computations for simple data. This dispute imposes an exciting challenge of updating the centroid of layer $L_i \Delta k$ from the centroid of layer $L_i$ with light-weight calculation, where $k = 1, 2, 3, ..., n$. Such a challenge never occurred before as this is the first work combining model update with prior termination.

Let $C_i^j$ be a centroid of the clustering model at layer $i$ where $j$ is the number of members in the cluster where $C_i^j = \frac{\sum_{k=1}^{j} X_k^j}{j}$. For the next layer, $i + 1$, the centroid is

$$C_{i+1}^j = \frac{\sum_{k=1}^{j} \sigma(W_{i+1}^j \times X_k^j)}{j} \quad (10)$$

Here, $W_{i+1}^j$ and $B_{i+1}^j$ are weight and bias for layer $i + 1$ respectively. $\sigma$ is the non-linear function; e.g., RELU. This formula requires, at least $j$ multiplication. As multiplication is an expensive function, our goal is to avoid using them. Therefore, we approximate Equation 10 using the following equation and reduce the number of multiplication by $j - 1$.

$$C_{i+1}^j = \frac{\sigma(W_{i+1}^j \times \sum_{k=1}^{j} X_k^j)}{j} \quad (11)$$

We assume that the non-linear function is RELU activation function. So, $\sigma(x) = \max(0, x) = \frac{x+|x|}{2}$. After applying the RELU function, we observe that the error, $\epsilon$ is

$$\epsilon_{i+1} = \frac{\sum_{k=1}^{j} |W_{i+1}^j \times X_k^j| - |W_{i+1}^j \times \sum_{k=1}^{j} X_k^j|}{2j} \quad (12)$$

7 EVALUATION

7.1 Experimental Setup

**Computational Device.** For evaluation we use TI-MSP430FR5994 [3] at 16MHz. This micro-controller is equipped with 256KB of FRAM, 8KB of SRAM, 16-channel 12-bit ADC, 6-channel direct memory access (DMA) and a Low Energy Accelerator (LEA). It has an operating voltage range of 1.8V to 3.6V. To program this device we use the Linux distribution and GCC compiler with an ez-FET programmer. Zygarde uses fixed point calculation and flip-flop buffers to enable DNN execution in MSP430.

To train the agile DNN model mentioned in Table 1 we use an Intel Core i7 PC with RTX2080 GPU. We train the network offline and compress it with rank decomposition or separation [22, 29, 38, 39, 59, 99] and pruning [46, 75]. We execute this compressed trained network for inference in our target device TI-MSP430FR5994.

**Energy Harvester.** We harvest energy from two ambient sources – solar and RF. We use a flexible Ethylene Tetrafluoroethylene (ETFE) based solar panel [1] that outputs 6V at 1W. We use a LTC3105 step up DC/DC converter with a start up voltage of 250mV [2] to charge the capacitor with the solar panel. A Powercast P2210B [5] is used to harvest RF energy from a 3W Power-caster transmitter [6], Figure 13. For all the experiments with intermittent power we use a 50mF capacitor. For persistently powered experiments we rely on the power supply from the ez-FET of the MSP430 launchpad.

**Sensor Peripheral.** For audio sensor, we use an Electret microphone [7] that draws 3.1mA current and has a start up time of 125ms. We utilize the built in ADC and internal clocks to read data from the microphone. To calculate the FFT we use the LEA and use DMA to write data to the FRAM without occupying the CPU.

**Time Keeping Peripheral.** Like [52, 105], we use a real-time clock, DS3132 [8] via I2C, for time keeping. This choice is made

---

2The implementation of Zygarde is available on https://github.com/zygarde-sensys/Zygarde.git
for the ease of implementations. Note that, we only use this clock during power up for syncing up and maintaining time with the internal clock of the MCU. This real-time clock is easily replaceable with SRAM and capacitor based time-keeping system during power off periods [53, 80]. Note that both time-keeping and intermittent execution in batteryless system is out of the scope of this paper.

Libraries. We use MSP430 libraries provided by Texas Instrument. For maintaining intermittent execution Zygarde uses SONIC [44] and the dependant libraries (e.g. ALPACA [67]) of this project. For training the agile DNN model we use Google’s Tensorflow [16].

Datasets. To evaluate our algorithms we use two popular datasets MNIST [61] and ESC-10 [78]. MNIST is a image based dataset which consists of $28 \times 28$ pixel images and 10 classes. ESC-10 consists environmental sounds of 10 classes. Each audio clip is 5s long and has a sampling rate of 44KHz. To accommodate this dataset with our resource constrained device, we take the middle 1s audio and down-sampled it to 8KHz.

Controlled Energy Source. To evaluate the system with different $\eta$, we perform controlled experiments with the energy sources. For RF, we vary the distance between the harvester and the transmitter from 1 feet to 5 feet. For solar, we simulate the sun with three dimmable bulbs with varying intensity (5.6 Klx - 35 Klx) as shown in Figure 13. Note that, for the real-life experiment we use outdoor scenarios and windowed rooms to get the sunlight.

7.2 Algorithm Evaluation

Effect of Layer-Aware Loss Function and Early Termination. In this section, we evaluate our proposed layer-aware loss function, early termination and adaptation. We train all models with same hyper-parameters and only use the training dataset. While inference we rely on the testing dataset provided by MNIST and ESC-10 (we use fold 1 only) containing 10,000 and 80 samples respectively. Note that, layer-aware loss function and early termination are applicable for any system, not just intermittently powered ones. Therefore, we do not bring the energy harvesting aspect in this evaluation and perform the evaluation on a persistently powered system. We compare our layer-aware loss function (L) and layer-aware loss function with adaptation (AL) with cross-entropy loss (CE) [107] and contrastive loss (C) [57]. We also consider a network trained with cross-entropy loss which only exits in the last layer similar to the one shown in SONIC [44].

For MNIST dataset in Figure 14, we observe that for cross-entropy loss (CE) with early termination decreases the accuracy along with the execution time. However, with the layer-aware loss(L) we increase the accuracy by 9.45% from cross-entropy loss (CE) and 3.19% from contrastive loss (C) while keeping similar execution time. Layer-aware loss function with adaptation (AL) increases the accuracy by 1% and allows more samples to terminate in the early layers.

ESC-10 is a complex dataset where previous works achieves 73% accuracy using models with three convolution layers, data augmentation and, 44KHz sampling rate [79]. Our network takes downsampled audio data of 1s duration and achieves 70% accuracy. In Figure 15, layer-aware loss function(L) achieves 76.25% accuracy and decreases the execution time by 1.22 minutes. Note that, layer-aware loss function with approximation (AL) does not show any improvement over layer-aware loss in this scenario because all data samples of several classes exit the network in earlier layers minimizing the need of approximate adaptation for deeper layers.

Performance of Real-Time Scheduler. In this section, we evaluate our proposed scheduling algorithm for dynamic imprecise tasks that is applicable to both persistently (Section 5.2) and intermittently (Section 5.3) powered systems for different $\eta$. We evaluate the system with two different CPU utilization.

For MNIST dataset in Figure 16, we consider a CPU utilization $\geq 1$. Thus even with persistent power we can not schedule all tasks. Without early termination the majority of samples could not be scheduled as they keep on waiting in the queue. But, with early exit 75% of samples can be scheduled. We consider sporadic tasks with a period of 3 seconds and the deadline is twice the period. According to the definition of imprecise computing, we consider completion of mandatory portion as a successful scheduling [89]. For ESC-10 dataset in Figure 17, we consider that CPU utilization $\leq 1$, the period is 0.36 minute and the deadline is twice the period.

We compare our approach with earliest deadline first (EDF) algorithm and a variation of EDF that only executes the mandatory portion (EDF-M). We consider various $\eta$ and use three systems that persistently powered, solar powered and RF powered. We notice that in all the cases Zygarde can successfully schedule higher number of tasks with higher accuracy. There are two major things to notice in Figure 16 and Figure 17.

- When $\eta$ is high Zygarde increases the number of tasks with correct result by executing some of the optional portion. For a 100% accurate utility function the performance of Zygarde and EDF-M would be same.
- For small $\eta$ the performance of Zygarde and EDF-M is the same because it satisfies the second condition of Equation 6 which only executes mandatory portion.
- The number of tasks that can be scheduled does not depend only on $\eta$ but also on the harvested energy.
7.3 Real-World Application Evaluation

**Experimental Setup.** In this section, we evaluate your system in an uncontrolled environment. We present an acoustic event detector as the application. We choose six scenarios with seven acoustic events. In three of these scenarios we use a solar harvester and for the rest we use a RF harvester. Table 5 describes the different scenarios along with the target events. We rely on the natural events for hindrance in power generation for the solar harvested systems. For example in the street scenario, the harvester with solar panel is kept on the edge of the pavement. The vehicles (cars, buses) passing through the nearest lane to the pavement is blocking the sun and thus introducing hindrance in energy generation. However, due to the lack of programmable RF harvester or proper setup to harvest from WiFi, we change the distance between RF harvester and transmitter. We use the same setup mentioned in Section 7.1 for the computation and sensing unit. Note that other sound sources were present in all scenarios and we included silence as an event in the classifier for training.
**Performance.** In Figure 18 shows the voltage across the capacitor along with the operating voltage of the computational unit of the three scenarios where the system is powered by a solar harvester. Note that the voltage across the capacitor and operating voltage is used to measure energy generation rate. The system runs a DNN with a single convolution layer and two fully connected layers. The execution time varies between 1.7 s and 3 s depending on the early exit. Among the 30 events we experienced we have four incorrect detection. We experience three deadline misses even though the detection is accurate. We miss four samples due to intermittence as the system did not have enough energy to be turned on.

Similar to Figure 18, Figure 19 shows the result for scenarios with RF harvester. We notice two interesting facts with this experiment. The first one is that the MCU experience higher intermittence rate for RF harvester than solar harvester. However, the duration of power off is much higher in solar harvester specially when we do not have the direct sunlight.

8 RELATED WORK

**Intermittent Computing.** Intermittently powered systems experience frequent power failure that resets the software execution and results in repeated execution of the same code and inconsistency in non-volatile memory. Previous works address the progress and memory consistency using software check-pointing [23, 54, 58, 64, 68, 70, 82, 96, 96], hardware interruption [18, 19, 71], atomic task based model [31, 32, 67] and, non-volatile processors(NVP) [65, 66]. Recently [43, 44] proposes a special software system for intermittent execution of deep neural inference combining task atomic task based model with loop continuation. Zygarde relies on [43, 44] for intermittent computation of deep neural network.

**Timeliness of Batteryless Systems.** Prior works on intermittent computing proposes runtime systems that increases the likelihood of a task completion by finding optimum voltage for task execution [26], adapting execution rate [41, 90] and discarding stale data [52]. However, none of these works consider the accuracy/utility of the running application or the real-time deadline-aware execution of tasks. Some works have addressed scheduling in wireless sensors [72, 73, 109] but none of them consider the higher computation load of intermittent computing systems. [105] proposes a reactive kernel that enables energy aware dynamic execution of multiple threads. Note that they do not consider deep neural tasks or utilize early termination of tasks for increasing schedulability. Unlike Zygarde which schedules multiple incoming data samples, [105] schedules kernel threads and can only have one data sample in the system at a time point. Other works focuses on maintaining time-keeping through power loss [53, 80]. Our work is a complementary of these works and relies on these techniques for time keeping.

**Compression and Partial execution of DNN.** Recent works have focused on reducing the cost of DNN inference by pruning and splitting models without compromising accuracy [46, 75, 101-104]. Other works have focused on reduction of floating point and weight precision [40, 45, 60], and factorization of computation [22, 29, 76, 92-94, 97] to reduce storage and computation cost. The proposed binary networks [34, 35, 55, 83] are not suitable for energy harvesting systems due to the higher number parameters needed by such systems [44]. Even though these works are crucial for enabling DNN execution in batteryless system these can be enhanced by exploiting the fact that in real-life data are usually a combination of easy examples and difficult examples and the easy examples do not need the full DNN inference [25, 42, 62]. Unlike prior works that require an additional classifier after each layer, Zygarde depends on semi-supervised models to reduce computational overhead. [21, 108] proposes scheduling algorithm for deep neural network in GPU and does not consider the constraints introduced by embedded system and intermittent power supply.

**Modeling Energy Harvesting System.** [85] analytically model the trade-off associated with backing up data to maximize forward propagation. Even though energy harvesting system for a specific energy source has been analyzed and modeled before [36, 59, 87], none of the prior works focus on modeling energy harvesting systems irrespective of energy source.

9 DISCUSSION

After execution an DNN layer, Zygarde calculates the Manhattan distance of the data sample from all n centroids using top k features, where n is the number of clusters. This requires n×k subtractions, n×k comparison (to determine absolute value) and n×k addition. Zygarde needs n+1 comparisons to find the minimum distance and k additions, k multiplications and k divisions to update a centroid. In our implementation, the highest number of n and k are 10 and 50, respectively. The total computation needed after each layer is 10% smaller than the required multiplication and addition needed for the smallest network layer mentioned in Table 1.

10 CONCLUSION

In this paper we propose a generic metric η that expresses the stability on an energy harvesting system. We also utilize the fact that real-life data samples are a combination of easy and hard samples and all samples do not require the same amount of computation to achieve similar performance. We propose early termination of deep neural network without compromising much accuracy. To decrease the accuracy loss due to early termination, we propose a layer-aware loss function and achieve 9.45% - 3.1% increase in accuracy and 14% decrease in execution time. We then model such DNN tasks as imprecise tasks and propose a scheduling algorithm that considers time, energy condition, η and performance of the system. We evaluate our system with state of the art scheduling algorithms and our algorithm schedules 33%-12% more tasks successfully.

REFERENCES

[1] [n. d.]. Flexible Ethylene Tetrafluoroethylene (ETFE) based solar panel. https://www.amazon.com/gp/product/B01EY5PFGW/ref=oh_aui_search_asin_title?ie=UTF8&psc=1.

[2] [n. d.]. LTC3105 step up DC/DC converter. https://www.analog.com/media/en/technical-documentation/data-sheets/3105fb.pdf.

[3] [n. d.]. MSP430FR5994. http://www.ti.com/lit/ds/symlink/msp430fr5994.pdf.

[4] [n. d.]. Network Traffic Dataset. https://www.kaggle.com/jbriggs/ip-network-traffic-flows-labeled-with-87-apps.

[5] [n. d.]. Powercast P2210B. http://www.powercastco.com/wp-content/uploads/2016/12/P2110B-Datasheet-Rev-3.pdf.

[6] [n. d.]. Powercaster Transmitter. http://www.powercastco.com/wp-content/uploads/2016/11/User-Manual-TX-915-01-Rev-A-4.pdf.
a compressor-critic framework. In *Proceedings of the 15th ACM Conference on Embedded Network Sensor Systems*. ACM, 4.

[105] Kasım Sinan Yıldırım, Amjad Yousef Majid, Dimitris Patoukas, Koen Schaper, Przemysław Paweleczak, and Josiah Hester. 2018. Ink: Reactive kernel for tiny batteryless sensors. In *Proceedings of the 16th ACM Conference on Embedded Networked Sensor Systems*. ACM, 41–53.

[106] Wanghong Yuan and Klara Nahrstedt. 2003. Energy-efficient soft real-time CPU scheduling for mobile multimedia systems. In *ACM SIGOPS Operating Systems Review*, Vol. 37. ACM, 149–163.

[107] Zhila Zhang and Mert Sabuncu. 2018. Generalized cross entropy loss for training deep neural networks with noisy labels. In *Advances in Neural Information Processing Systems*. 8778–8788.

[108] Husheng Zhou, Soroush Bateni, and Cong Liu. 2018. S’3DNN: Supervised Streaming and Scheduling for GPU-Accelerated Real-Time DNN Workloads. In *2018 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*. IEEE, 190–201.

[109] Ting Zhu, Abdelaziz Mohaisen, Yi Ping, and Don Towsley. 2012. DEOS: Dynamic energy-oriented scheduling for sustainable wireless sensor networks. In *INFOCOM, 2012 Proceedings IEEE*. IEEE, 2363–2371.