Zygarde: Time-Sensitive On-Device Deep Inference and Adaptation on Intermittently-Powered Systems

BASHIMA ISLAM, University of North Carolina at Chapel Hill, USA
SHAHRIAR NIRJON, University of North Carolina at Chapel Hill, USA

We propose *Zygarde* — which is an energy- and accuracy-aware soft real-time task scheduling framework for batteryless systems that flexibly execute deep learning tasks[1] that are suitable for running on microcontrollers. 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. We empirically study the problem and model the energy harvesting pattern as well as the trade-off between the accuracy and execution of a DNN. We develop an *imprecise computing*-based scheduling algorithm that improves the timeliness of DNN tasks on intermittently powered systems. We evaluate Zygarde using four standard datasets as well as by deploying it in six real-life applications involving audio and camera sensor systems. Results show that Zygarde decreases the execution time by up to 26% and schedules 9% – 34% more tasks with up to 21% higher inference accuracy, compared to traditional schedulers such as the earliest deadline first (EDF).

CCS Concepts: • **Computing methodologies → Machine learning**; • **Computer systems organization → Embedded systems**; **Real-time system specification**; • **Hardware → Power and energy**.

Additional Key Words and Phrases: Intermittent Computing, Deep Neural Network, Semi-Supervised Learning, On-Device Computation, Real-Time Systems, Batteryless System, Energy harvesting System, On-Device Learning

ACM Reference Format:
Bashima Islam and Shahriar Nirjon. 2020. Zygarde: Time-Sensitive On-Device Deep Inference and Adaptation on Intermittently-Powered Systems. *Proc. ACM Interact. Mob. Wearable Ubiquitous Technol.* 4, 3, Article 82 (September 2020), 29 pages. https://doi.org/10.1145/3411808

1 INTRODUCTION

Batteryless IoT devices are powered by harvesting energy from ambient sources, such as solar, thermal, kinetic, and radio-frequency signals (RF). These devices, in principle, last forever—as long as the energy harvesting conditions are in favor. They have applications in long-term sensing and inference scenarios, such as wildlife monitoring, remote surveillance, environment and infrastructure monitoring, and health monitoring using wearables and implantables. 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 [64, 70], car

[1]The DNN, by definition, refers to neural networks having more than one hidden layers [41, 57, 63, 91]. 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 [54, 84, 85]

Authors' addresses: Bashima Islam, University of North Carolina at Chapel Hill, 201 S. Columbia St, Chapel Hill, USA, bashima@cs.unc.edu; Shahriar Nirjon, University of North Carolina at Chapel Hill, 201 S. Columbia St, Chapel Hill, USA, nirjon@cs.unc.edu.

2474-9567/2020/9-ART82 $15.00
https://doi.org/10.1145/3411808

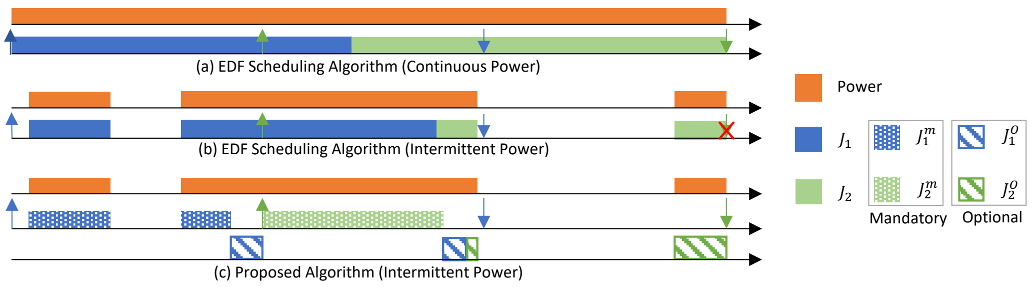
(a) EDF Scheduling Algorithm (Continuous Power)
(b) EDF Scheduling Algorithm (Intermittent Power)
(c) Proposed Algorithm (Intermittent Power)
Power
J_1
J_1^m
J_1^O
J_2
J_2^m
J_2^O
Mandatory
Optional

Fig. 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.

detectors [42], and machine monitors [120], events need to be detected and reported on time in order to ensure prompt responses, safety, and timely maintenance. Though a batteryless system is desirable, the unpredictability of the harvested energy, combined with the complexity of on-device event recognition tasks, complicates timely execution of machine learning-based event detection tasks on such systems.

Prior works on time-aware batteryless computing systems can be broadly categorized into two types. The first category focuses on maintaining a reliable system clock when the power is out [46, 60, 109]. The second category includes runtime systems that consider the temporal aspect of data across power failures [30, 59, 139, 144] by discarding data after a predefined interval or considering energy to increase the chances of task completion. Recent works [53, 54, 85] on batteryless systems have proposed frameworks and runtime to execute *non-real-time* DNN[1] tasks on intermittently-powered systems. The real-time community have proposed techniques [24, 69, 86, 134, 143] for deadline-aware execution of DNNs, primarily on GPU and server-grade machines. 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, variable utility of DNN inference, and real-time schedulability; and merely combining existing solutions does not entirely solve the problem.

We illustrate this using an example in Figure 1. We 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. The intermittency of energy generally has no consistent pattern in the duration of or in the gaps between ON/OFF phases. Figure 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 1(b)), task J_2 misses its deadline. Figure 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.

In this paper, we 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, energy generated by a harvester is *bursty*, i.e., the state of energy generation is maintained over a short period. This property enables us to obtain a probabilistic model of the energy harvesting pattern – which can be characterized by a single parameter for a given harvester used in a particular application scenario. Second, since deeper layers of a DNN extract more detailed and fine-grained features of the input data, for a given accuracy, the required amount of DNN computation depends on the quality of data itself. Zygarde exploits these observations and proposes an *imprecise computing*-based [90, 118] online scheduling algorithm, which considers both the intermittence of energy and the accuracy-execution trade-off of a DNN.

The main contribution of this paper 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 [54, 56, 104, 135–138]. 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) 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 [60, 109] and execution of non-real-time tasks [18, 19, 36, 53, 54, 95, 96, 100, 111]. 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 [54, 95] that handle the lower-level aspects of an intermittent system.

We implement Zygarde on a TI MSP430FR5994 microcontroller and evaluate its performance using four standard datasets (MNIST [82], ESC-10 [108], CIFAR-100 [81], and Visual Wake Works [35]) 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 [142] and contrastive loss [71]. 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.

2 ZYGARDE SYSTEM DESIGN

In this section, we describe the system architecture of Zygarde and provide an example to illustrate its execution.

2.1 System Architecture

Zygarde is a framework and a runtime system that enables time-aware execution of a DNN on intermittently-powered systems. It consists of five major components: a job generator, an energy manager, an agile DNN model, semi-supervised k-means classifiers, and a scheduler. These components are shown in Figure 2.

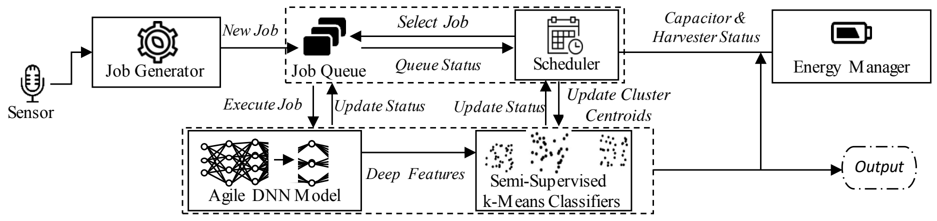

Fig. 2. Zygarde System Architecture.

Job Generator. Zygarde reads data from sensors (e.g., a microphone) and writes them to the non-volatile memory (FRAM) of the microcontroller using direct memory access (DMA). It considers the processing of the sensor data stream for each classification task (e.g., user identification and event detection) as separate *tasks*. The end-to-end processing of one data sample of a sensor stream (i.e., from feature extraction to classification and model adaptation) is called a *job*. For example, an audio sensing system that performs both speaker recognition and hotword detection has two tasks, and the processing of each audio frame to recognize the speaker and to detect the hotword are two separate jobs. Thus, if this system (having 2 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 a queue. A job leaves the queue when it gets scheduled for execution or its deadline has passed.

Energy Manager. The *Energy Manager* monitors the state of the energy storage (i.e., the supercapacitor) and the energy harvesting rate. The scheduler uses this information for scheduling decisions (described in Section 5). The energy manager implements an open-source intermittent computing runtime [54, 95] to manage the execution of jobs across power failures. At the implementation level, each job of Zygarde consists of multiple small atomic fragments that maintain a strict precedence order. These fragments execute atomically and the runtime ensures that repeated attempts to execute a fragment is idempotent.

Agile DNN Model. The *Agile DNN Model* is a pre-trained deep neural network that converts data samples into feature representations [25]. The network is trained offline, on a high-end server, using labeled training data. We use rank-decomposition [26] and separation [56] 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, we call it an *agile* DNN Model. We note that an agile DNN is a special type of anytime DNN [65, 76–78, 125, 140] — 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 [4, 13]. 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 [16]. The clustering algorithm uses the L1 distance between two feature vectors [28]. 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 4.3. 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.

Scheduler. 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 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

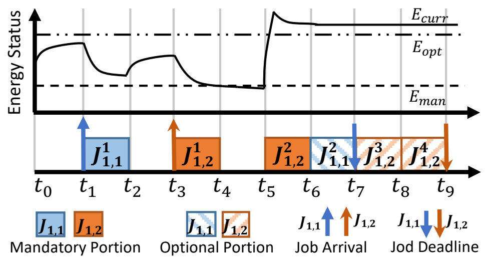
Energy Status
E_{curr}
E_{opt}
E_{man}
$J^1_{1,1}$
$J^1_{1,2}$
$J^2_{1,2}$
$J^2_{1,1}$
$J^3_{1,2}$
$J^4_{1,2}$
t_0 t_1 t_2 t_3 t_4 t_5 t_6 t_7 t_8 t_9
$J_{1,1}$ $J_{1,2}$ Mandatory Portion
$J_{1,1}$ $J_{1,2}$ Optional Portion
$J_{1,1}$ $J_{1,2}$ Job Arrival
$J_{1,1}$ $J_{1,2}$ Jod Deadline

Fig. 3. Execution schedule of the workload.

Table 2. Explanation of the schedule in Figure 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).
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.

2.2 Example Execution

Table 1 defines two jobs $J_{1,1}$ and $J_{1,2}$ from the same task, τ_1. Figure 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 job $J_{i,j}$. Table 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, we assume 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) 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 6) estimates the maximum energy required by any atomic fragment by running EnergyTrace++ [1] 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, we minimize this wastage, and increase the performance of the system. However, a developer can override these values using the APIs provided (Section 6.2). 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.

3 MODELING INTERMITTENT ENERGY

This section derives a single metric, namely the η-factor, which characterizes an energy harvester used in a particular application. The metric is later used in the real-time scheduler in Zygarde.

3.1 Energy Event

The unavailability of harvestable energy and the required time to buffer sufficient energy before it can be used cause intermittence in batteryless systems. The pattern of intermittence 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, we 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. In our empirical model, we set ΔK to E_{man}.

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, we conduct a two-month long study on solar and RF harvesters (using empirically collected data) and human footsteps (using the dataset [98]) to analyze the pattern of energy events. 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 next 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 is going to be high over the next 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. We denote an energy event at time t using the random variable $H_t \in \{0, 1\}$.

3.2 Conditional Energy Event

We 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 $N < 0$):

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

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

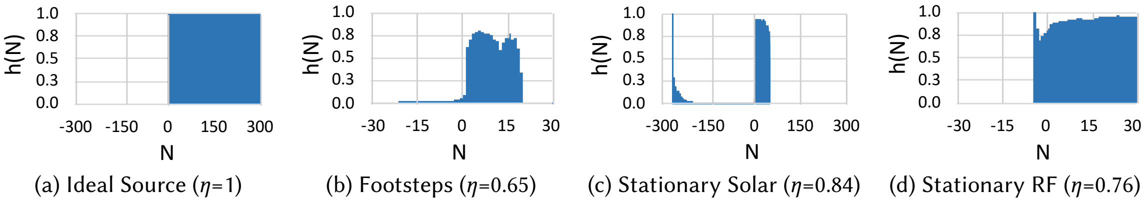

Fig. 4. Conditional energy event for (a) persistent power source, (b) piezo-electric harvester, (c) stationary solar harvester, and (d) stationary RF harvester. We use $\Delta T = 5$ minutes.

Figures 4(a)-(d) show the distribution of $h(N)$ for a persistently-powered and three energy harvested systems. To characterize an energy harvester, we measure the Kantorovich-Wasserstein (KW) distance [75] between its distribution, $\mathcal{H}^{(i)}$ from an ideal (persistent power) source, $\mathcal{P}$ to obtain:

$$KW(\mathcal{H}^{(i)}, \mathcal{P}) = \int_{-\infty}^{+\infty} |CDF(\mathcal{H}^{(i)}) - CDF(\mathcal{P})| \tag{2}$$

In Figure 4, we observe that $h(N)$ drops when $|N|$ increases. For example, in Figure 4(b), $h(N)$ drops after $N = 20$ since the person we studied never walked for more than 100 minutes. Similarly, in Figure 4(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. We also notice that after about 19 hours of absence of any energy event (i.e., light intensity < 2730 lux), the probability of the next energy event is high as the sun shows up again at the window.

3.3 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, we normalize the KW score against a purely random harvesting pattern, $\mathcal{R}$ to obtain a revised metric, called the η-factor:

$$\eta = 1 - \frac{KW(\mathcal{H}^{(i)}, \mathcal{P})}{KW(\mathcal{R}, \mathcal{P})} \tag{3}$$

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 thus 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. A critical discussion of η-factor is presented in Section 11.

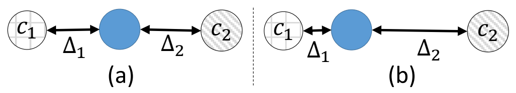

Fig. 5. Two nearest clusters c_1 and c_2 of an input (in the middle) is shown: (a) early exit does not happen since $|\Delta_2 - \Delta_1|$ is small; (b) early exit happens since $|\Delta_2 - \Delta_1|$ is large.

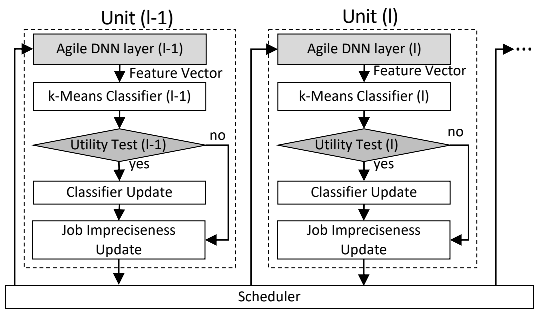

Fig. 6. Sequential execution of units of an agile DNN.

4 MODELING DNN TASKS

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

4.1 Task Model

Tasks, Jobs, Units, and Fragments. Zygarde considers the processing of a sensor data stream for each classification task as an *imprecise sporadic task* [90], $\tau_i = (T_i, D_i, C_i)$, where T_i denotes the period (i.e., the minimum separation between two consecutive jobs), D_i is the relative deadline, and C_i is the worst-case execution time.

An instance of a task, aka a *job*, comprises of an ordered sequence of modules (henceforth these modules are called *units*). 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 [90, 101, 141]. 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.

Although a *unit* represents a logical grouping of related modules at each layer of the DNN, the size of a typical unit is generally too large to execute without an intermittence. Hence, at the implementation level, to avoid corrupted results and to ensure forward progress of code execution, these units are further divided into atomically executable *fragments*—which guarantees correct intermittent execution using SONIC API [54].

Dynamic Partitioning and Utility. Unlike traditional imprecise computing models [90, 101, 141] where the partition of a job into mandatory and optional parts is known, the number of mandatory units in Zygarde is determined at runtime. We 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 we use a k-means classifier, we 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, we 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, we 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.

The utility function described above runs in linear time with the number of clusters, i.e., $O(k)$. 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 4.3 describes how the utility threshold is computed using an empirical dataset. Section 11 further discusses alternative utility functions that are suitable for other types of classifiers.

Preemption and Task Switching Zygarde allows limited preemption [20] 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. By prohibiting preemption of a unit, Zygarde reduces context switching and read-write overheads, and minimizes the memory requirements to $O(N)$ for N jobs by using double-buffering [17]. Figure 6 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.

4.2 Agile DNN Construction

Unlike previous works where inference happens only at the last layer [133] or where a second classifier is used at hidden layers to decide early exit [125], 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 [71] at each layer is used as the loss function of an Agile DNN – which we call the *layer-aware* loss function:

$$LA = \sum_{i=1}^{L} a_i \times LC\Big(W^i, X_1^i, X_2^i, \cdots, X_N^i\Big) \tag{4}$$

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:

$$LC\Big(W^i, X_1^i, X_2^i\Big) = \frac{1}{2}\Big(1 - Y\Big)\Big(G_{W^i}(X_1^i) - G_{W^i}(X_2^i)\Big) + \frac{1}{2} Y \max\Big(0, \Delta - G_{W^i}(X_1^i) - G_{W^l}(X_2^i)\Big) \tag{5}$$

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, we use a siamese network architecture [71] as shown in Figure 7. In a siamese network, there are two identical neural networks, called the sister networks, that share the same weights. From the labeled dataset, we 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 [71], which only uses the contrastive loss at the last layer (LC_3), we use the layer-aware loss function (Equation 4) at every layer to train these networks. We perform an exhaustive search for hyper-parameter tuning and to determine the weights of each layer. After the training, we use only one of the sister networks for inference. During inference, we 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, we 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.

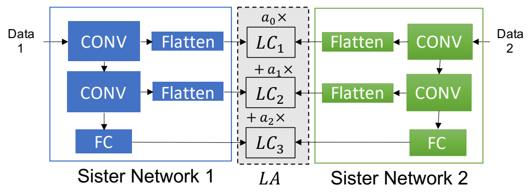

Fig. 7. Training agile DNN with Siamese network

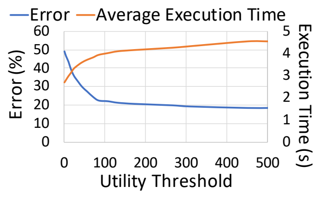

Fig. 8. Effect of utility threshold on performance

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.

4.3 Semi-Supervised k-Means Classifiers Construction

In Zygarde, we maintain a semi-supervised k-means classifier at each unit. In this section, we describe how we construct and update these classifiers.

Computing Cluster Centroids. Using the trained agile DNN, we obtain a feature representation from each layer for each data point in the training dataset. Using these representations, we train a semi-supervised k-means classifier corresponding to each layer of the agile DNN (see Figure 6). Using the labeled training data, we select the top N features using SelectKBest [52] and χ^2 tests, so that the features are computable on the resource-constrained target device. We utilize the labeled training data to determine the value of k (for k-means) and assign a class label to each cluster. Finally, we 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 8 for the first layer of the DNN on the CIFAR-100 dataset. For different layers, we see similar trade offs. To determine a suitable utility threshold for each layer, we 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. We 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 [23, 33, 47]. Referring to Figure 6, 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 [114].

Since this technique estimates the cluster centroids of a deeper layer instead of actually running the data samples through those layers, it saves $O(r)$ multiplication operations and performs the operation in $O(1)$; 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 REAL-TIME SCHEDULER

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

5.1 Scheduler for Persistent Systems

Despite being an optimal online scheduling algorithm for sporadic tasks, the earliest deadline first (EDF) algorithm [80] is not directly applicable to Zygarde as EDF does not consider the accuracy of a DNN. Furthermore, traditional scheduling algorithms for imprecise tasks [90, 117] 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, we 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 4.1) and the dynamically determined impreciseness status (i.e., mandatory vs. optional) of a unit:

$$\zeta_{i,j}^l = \Big(1 - \alpha(d_{i,j} - t_c)\Big) + \Big(1 - \beta\Psi_{i,j}^l\Big) + \gamma_{i,j}^l \tag{6}$$

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 $\gamma_{i,j}^l = 1$ or optional $\gamma_{i,j}^l = 0$, 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.

5.2 Scheduling for Intermittent System

For an intermittently-powered system, we utilize the η-factor introduced in Section 3 to extend ζ as follows:

$$\zeta_{I_{i,j}}^l = \begin{cases} \big(1 - \alpha(d_{i,j} - t_c)\big) + \big(1 - \beta\Psi_{i,j}^l\big) + \gamma_{i,j}^l, & \eta E_{curr} \geq E_{opt} \\ \gamma_{i,j}^l\Big(\big(1 - \alpha(d_{i,j} - t_c)\big) + \big(1 - \beta\Psi_{i,j}^l\big)\Big), & \eta E_{curr} < E_{opt} \end{cases} \tag{7}$$

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. We 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. We omit the explanation of the three terms 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 [30]. 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 $\eta E_{curr} \geq E_{opt}$.

5.3 Schedulability Condition

A set of N sporadic tasks is schedulable by an imprecise scheduler if the total utilization, $\sum_{i=1}^{N} \frac{C_i}{T_i} \leq 1$ [90], 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, we model power outages event as a very high-priority job of a sporadic *Energy 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 $T_E \geq \frac{\eta/(1-\eta)}{1-\sum_{i=1}^{N}(C_i/T_i)}$.

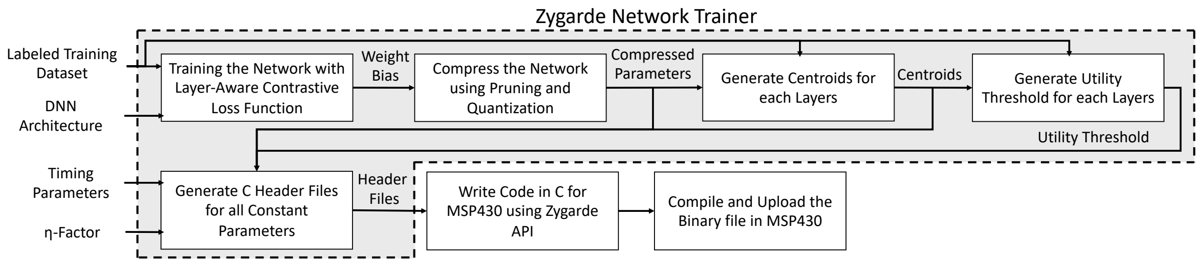
Zygarde Network Trainer
Labeled Training Dataset
DNN Architecture
Timing Parameters
η-Factor
Training the Network with Layer-Aware Contrastive Loss Function
Weight Bias
Compress the Network using Pruning and Quantization
Compressed Parameters
Generate Centroids for each Layers
Centroids
Generate Utility Threshold for each Layers
Utility Threshold
Generate C Header Files for all Constant Parameters
Header Files
Write Code in C for MSP430 using Zygarde API
Compile and Upload the Binary file in MSP430

Fig. 9. Zygarde Programming Framework

6 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, we obtain an executable binary file for the MSP430 MCU.

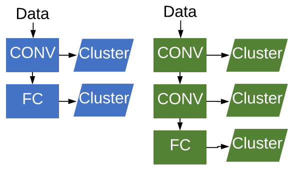

Fig. 10. Two sample DNNs.

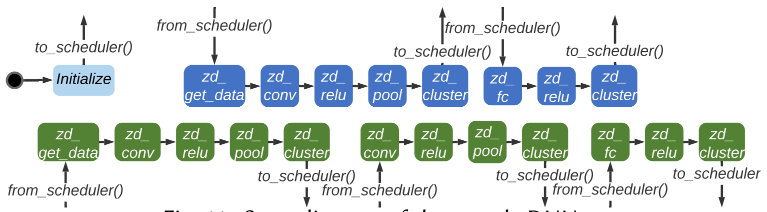

Fig. 11. State diagram of the sample DNNs.

6.1 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 9 shows the intermediate steps inside the Zygarde network trainer.

At first, the network trainer trains the agile DNN model using the labeled dataset using the layer-aware loss function described in Section 4.2. 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 [26, 34, 43, 44, 56, 104, 127]. 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 4.3.

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).

6.2 Zygarde APIs

The Zygarde APIs extend open source SONIC [54] APIs for intermittent DNN computing by incorporating Zygarde-specific capabilities, such as early termination, cluster-based inference, and scheduling. We divide the APIs 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 10 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 11.

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

7 IMPLEMENTATION

Computing Device. We use TI-MSP430FR5994 [10] MCU (shown in Figure 12) 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, we 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.

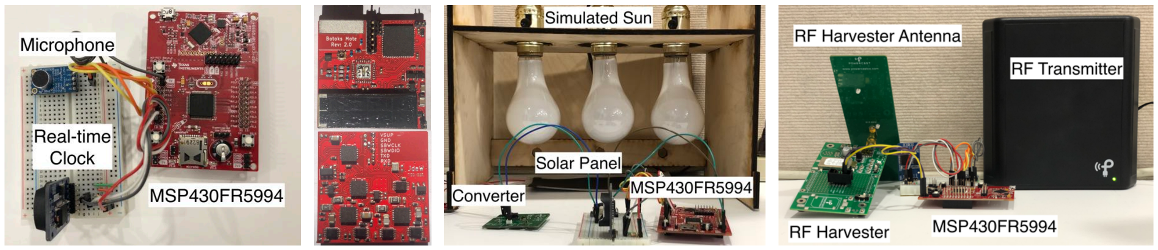

Fig. 12. Zygarde experimental setup.

Energy Harvester. Figure 12 shows our solar and RF energy harvester setup. The solar harvester includes an Ethylene Tetrafluoroethylene (ETFE) based solar panel [12] and a step-up regulator [3]. We use the Powercast harvester-transmitter pair [6, 7] to harvest RF energy. Like previous works on intermittent systems [54, 95], both harvesters use a 50mF capacitor.

Sensor Peripheral. We use an electret microphone [8] and the built-in ADC in MSP430 for acoustic sensing. For visual sensing, we use an OV2640 CMOS camera module [2] connected via I2C and SPI. Using LEA and DMA, we perform FFT on audio data and write the audio data to the FRAM without involving the CPU.

Time Keeping. Like [59, 139], we use a real-time clock, DS3231 [5] for timekeeping in most of the experiments. We 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 [60, 109]. In order to quantify the effect of such a batteryless timekeeper on Zygarde, we implement and use an open-source remanence clock, called CHRT [46], 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 [54] and related APIs (e.g., ALPACA [95]). We use Tensorflow [14] for training the DNN models.

8 MICROBENCHMARKS

In this section, we evaluate each component of Zygarde using datasets and compare Zygarde with baseline algorithms. We also observe the effect of capacitor size and remanence clock on Zygarde.

8.1 Datasets and Environments

Datasets and DNNs. To evaluate the performance of different components of Zygarde, we use four datasets: MNIST [82], ESC-10 [107], CIFAR-100 [81], and Visual Wake Word [35]. 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 [54]. ESC-10 also has ten classes and 44.1 kHz five seconds-long audio clips. We use 1s audio downsampled to 8KHz. We 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, we 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 [89]. To fit these images into the MCU's memory, we first crop an image to move the target object (human) in the center and then downsample the cropped image to 32×32 pixels. Our dataset can be accessed at [11]. Note that if we only downsample the image to 32×32 pixels without cropping it

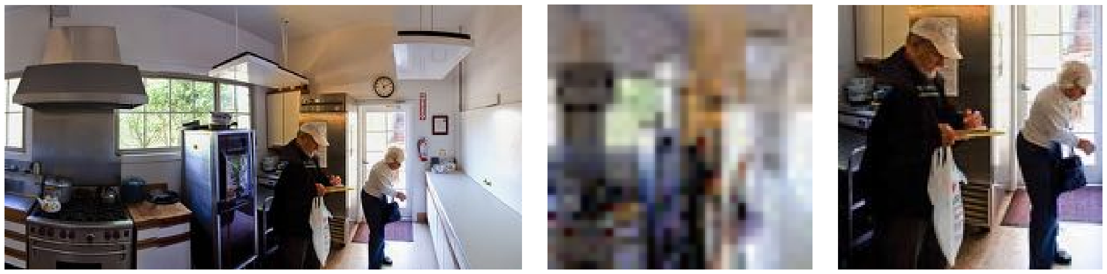
(a) Original Image (b) Downsampled Only (c) Targeted Crop

Fig. 13. Visual Wake Word dataset: (a) Original image (640×320), (b) Only downsampled (32×32), (c) After targeted cropping and downsampling (32×32).

first, the resultant image scales down the target object (human) so much that they are not recognizable anymore. Figure 13 shows an example image from the VWW dataset, followed by two downsampled versions of it – with and without cropping.

We implement four compressed networks summarized in Table 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), we perform controlled experiments. To determine the value of ΔK, we run the system for multiple iterations and take the highest observed energy consumption. We vary the distance between the transmitter and the receiver between 1-5 feet for RF. We simulate solar power with three dimmable bulbs with varying intensity (5.6 Klx - 35 Klx) as shown in Figure 12. The seven scenarios considered for the evaluation is described in Table 4. Note that, we use outdoor scenarios and windowed rooms to get the sunlight for the real-life experiments in Section 9.

Table 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

Table 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

8.2 System Overhead

Figure 14 shows the overhead of different components of Zygarde described in Section 2. To measure the execution time and energy consumption, we use the TI eZ-FET debug probe with EnergyTrace++ [1], which provides milliseconds and μJ resolution data. We isolate each component of Zygarde and report the average overhead from five repeated measurements. To measure smaller overheads we repeat the experiment for multiple iterations (e.g., 2000 iterations for energy manager) and report the mean overhead of a single execution. We use a persistent power source during overhead measurements.

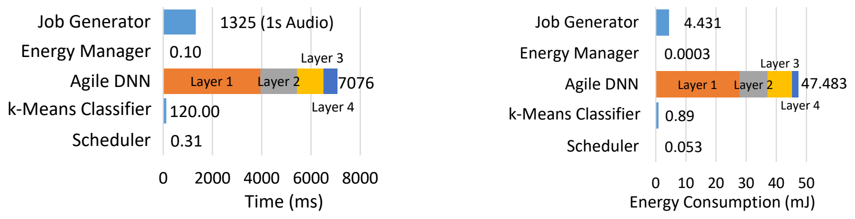

Fig. 14. Overhead of Zygarde.

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.

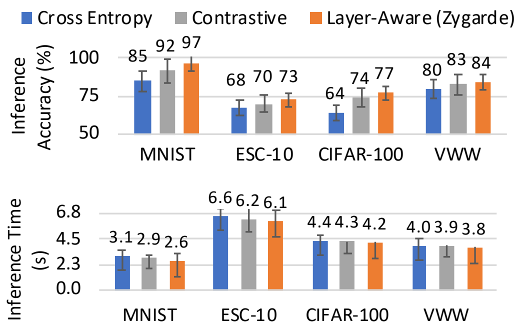

Fig. 15. Comparison of Loss Functions with Early Exit.

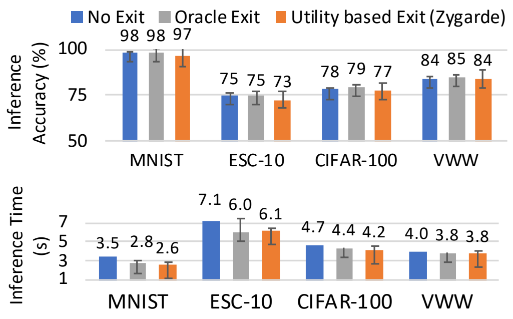

Fig. 16. Comparison of the Termination Policies.

8.3 Effect of Layer-Aware Loss Function

In Figure 15, we compare our proposed layer-aware loss function with cross-entropy loss [142] and contrastive loss [71] functions when early termination is in action. Since loss functions are equally applicable to both persistently-powered and energy-harvested systems, we conduct this experiment in a persistently-powered setting. We 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 15, the layer-aware loss function demonstrates 4.13%-13.40% higher accuracy than cross-entropy loss by forcing the layers to learn distinguishable features [132]. 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,

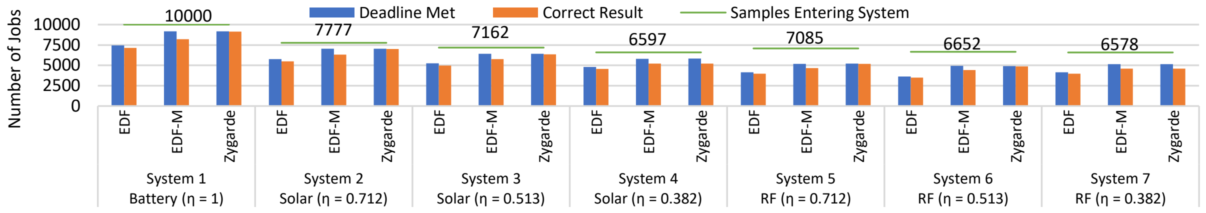

Fig. 17. Real-time Scheduling for different Systems on MNIST test dataset.

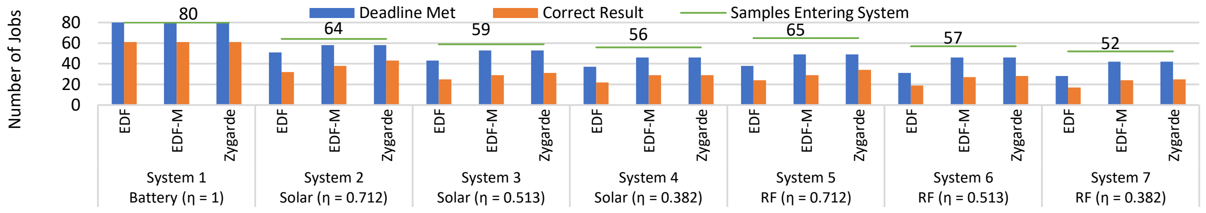

Fig. 18. Real-time Scheduling for different Systems on ESC-10 test dataset.

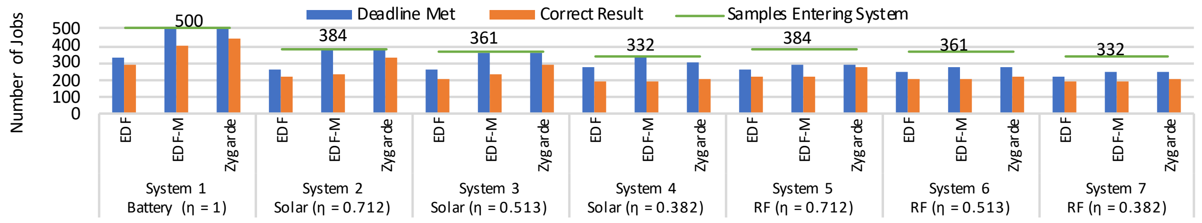

Fig. 19. Real-time Scheduling for different Systems on CIFAR-100 test dataset.

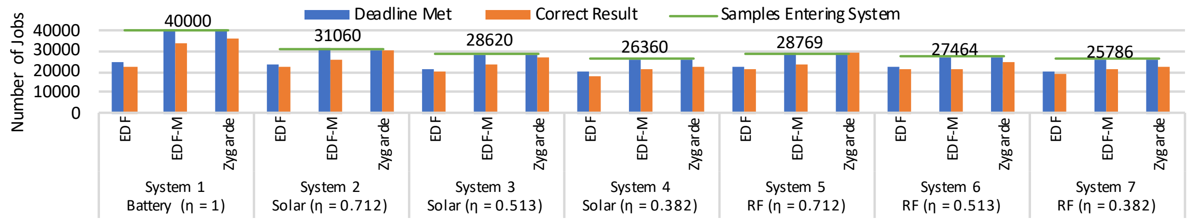

Fig. 20. Real-time Scheduling for different Systems on Visual Wake Word test dataset.

layer-aware loss function achieves higher accuracy and lower inference time than other loss functions when early termination is active.

8.4 Effect of Early Termination

In Figure 16, we 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. We use the same persistently-powered system and dataset as in Section 8.3. 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%.

8.5 Performance of the Real-Time Scheduler

We 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 4. To compare our proposed algorithm, we 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. We 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, we discard a job after its deadline to avoid domino effect [116].

Persistently Powered System. Figure 17 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 18, we 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 19 and 20, we 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. We 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.

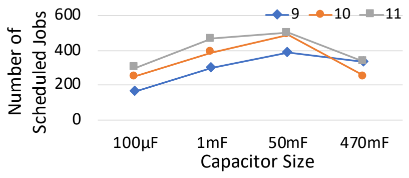

Fig. 21. Effect of Capacitor

Table 5. Effect of Cascaded Hierarchical Remanence Timekeeper

System	Number of 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

8.6 Effect of Capacitor Size

The goal of this experiment is to quantify the effect of the capacitor's size on scheduling. We use the CIFAR-100 dataset and its corresponding DNN (Table 3), and power the system from an intermittent RF energy source (η = 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. We 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 21 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, we choose to use a 50mF capacitor for the rest of the experiments. Note that although we 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: $C = \sqrt{\frac{2P\delta T}{V^2}}$.

8.7 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) [46] 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. We implement this clock following the open source hardware design (see Figure 12) and use it to power Zygarde– to implement a completely batteryless system. We 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 shows the number of tasks meeting deadlines for both types of clocks for the systems 2–4 (see Table 4 for definitions). We 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. We observe that the number of missed jobs increases with the number of reboots due to intermittent energy. Upon investigating the cause we 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%.

9 REAL-WORLD APPLICATION EVALUATION

In the previous section, we compared the performance of Zygarde with different baseline algorithms. In this section, we observe Zygarde in two real-world applications. In the first application, we perform acoustic sensing and show how different scenarios affect the system performance. In the second application, we compare the performance of Zygarde with a state-of-the-art intermittent DNN inference system [54] for visual sensing tasks in a real-world setting.

9.1 Acoustic Sensing

Experimental Setup. In this section, we evaluate Zygarde in real-world uncontrolled experiments using six audio event detection applications. Due to the presence of background noise and multiple audio 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 [9, 79].

Table 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. We 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

Table 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

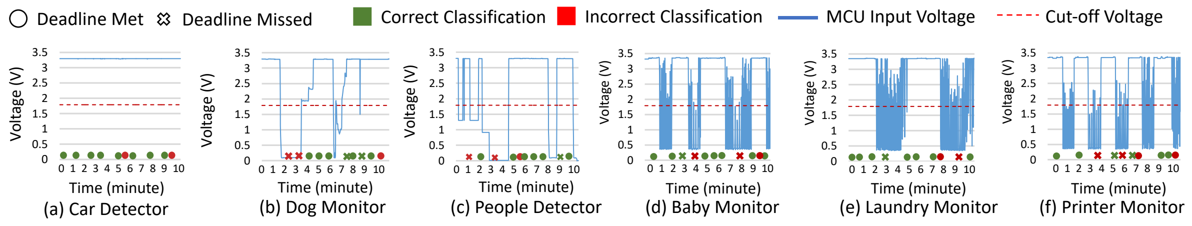

Fig. 22. Real-life evaluation of Zygarde for acoustic event detection.

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, we combine the outputs of two consecutive jobs by taking their logical OR.

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

Results. In Figures 22(a)-(f) we plot 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 22(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 22(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 22(c) fails to sense two events and misclassifies one.

The baby monitor in Figure 22(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 22(e) misclassifies one event and misses the deadline for two. The printer monitor in Figure 22(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.

9.2 Visual Sensing

Experimental Setup. We 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 23(a) shows the setup for this experiment. We use a 2MP OV2640 camera sensor and capture the test images from the GTSB [121] dataset displayed on the screen of a laptop. We use 80% of the dataset for training and the remaining 20% for testing. We annotate the dataset to label the shape of the sign. We 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.

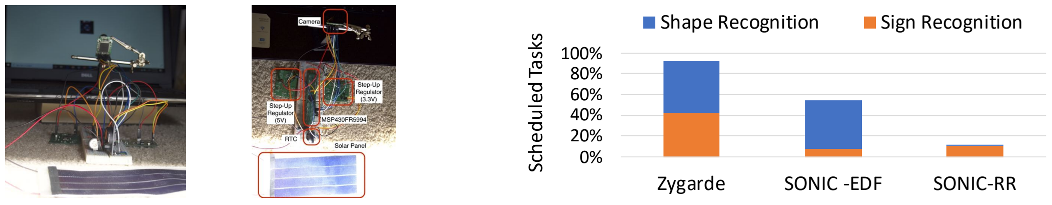

Fig. 23. (a) Experimental setup. (b) Percentage of captured events that meet the deadline.

Results. In Figure 23(b), we compare the performance of Zygarde against SONIC's [54], which does not implement early termination and uses either an EDF or a round-robin (RR) scheduler. We 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.

10 RELATED WORK

This section describes the existing literature on intermittent computing, timeliness in batteryless systems, DNN compression and partial execution (Anytime Neural Networks), energy harvesting system model, and real-time scheduling for imprecise and mixed critical task set.

Intermittent Computing. Intermittently powered systems experience frequent power failures that reset the software execution and result 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 [27, 61, 72, 92, 96, 100, 111, 128], hardware interruption [18, 19, 99], atomic task-based model [36, 37, 95] and, non-volatile processors(NVP) [93, 94]. Recently SONIC [53, 54] proposes a unique software system for intermittent execution of deep neural inference combining atomic task-based model with loop continuation. Despite being used for intermittent execution by Zygarde, SONIC does not handle time-aware, partial execution of DNN with model adaptation like Zygarde.

Timeliness of Batteryless Systems. Prior works on intermittent computing propose runtime systems to increase the likelihood of task completion by finding optimum voltage [30], adapting execution rate [49, 119], and discarding stale data [59]. However, none of these works consider the accuracy/utility of the target application or deadline-aware execution of tasks. Some works in wireless sensors [102, 103, 144] have addressed scheduling, but none of them consider the higher computation load of inference tasks. INK [139] proposes a reactive kernel that enables energy-aware dynamic execution of multiple threads. 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, Zygarde schedules multiple data samples present in the job queue. Recent work on real-time in intermittent systems [69] explores this problem for only periodic tasks and unlike Zygarde does not considers the impreciseness of the taskset. Other works focus on maintaining time through power loss [46, 60, 109] by exploiting discharge rate of SRAM and capacitors. Our work is complementary to these works and relies on these techniques for timekeeping.

Compression and Partial execution of DNN. Recent works reduce the cost of DNN inference by pruning and splitting models [56, 104, 135–138], reduction of floating-point and weight precision [45, 55, 74], and factorization of computation [26, 34, 105, 122–124, 131]. Though these works are crucial for enabling fast DNN execution, they alone are not sufficient for batteryless systems. Binary networks [38, 39, 68, 112] are not suitable for batteryless systems due to the higher number of required parameters [54].

Recent works propose early exit during DNN inference [29, 51, 87, 125]. However, they are not sufficient for highly constrained batteryless systems due to the large termination overhead. In most cases, the terminations require execution of a neural layer requiring 45× more execution cycles than performing utility test and classification with Zygarde . Moreover, these approaches lack model adaptation capability which is required for life-long sensing. Some works on anytime neural networks depends on module selection [66, 106, 126], dynamic layer pruning during inference [29, 58, 66, 67, 129] and depth and width adjusting techniques [65, 76–78, 140] for anytime prediction. However, none of these works have considered the the effect of energy intermittence and they are yet to be modeled as imprecise task.

Modeling Energy Harvesting Systems. Previous works on energy harvesting (EH) modeling of a specific energy source have achieved promising results in predicting available energy [40, 73, 115]. Some other works have focused on analytically model the trade-off associated with length of history to maximize forward propagation [113]. However, none of the prior works are generalizable and fails to model energy harvesting systems irrespective of energy source. In Zygarde, we provide a single metric, η factor that models the predictability of an energy harvesting system irrespective of the source.

Real-time Scheduling for Imprecise and Mixed Critical Task Set. Previous works on imprecise computing only considered the task models where mandatory-optional partitions are fixed and known a priori [90, 117]. Existing works on Quality of Service (QoS) based resource management [83, 110] do not handle the data-dependant dynamic relationship between quality and required unit for each job in Zygarde. Though, Zygarde can also be formulated as a mixed-critical system, the changing utility of jobs at runtime and intermittent energy makes it more suitable as a imprecise-computing system. Previous works on mixed-critical systems [21, 22, 31, 48, 88, 130] propose scheduling schemes for predetermines critical levels, which is a characteristic of a task. On the other hand, utility in Zygarde varies for each job.

11 DISCUSSION

11.1 Importance of DNNs

For batteryless sensing systems, the inference accuracy dictates the response time and the energy-efficiency of the system [54]. 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 ?? shows that DNNs are 1%–15% more accurate than the traditional classifiers.

Table 7. Classification Accuracy for Different Models.

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

11.2 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 [15], 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 [15], e.g., neural networks, naïve bayes, and logistic regression, we recommend using the entropy [32] 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.

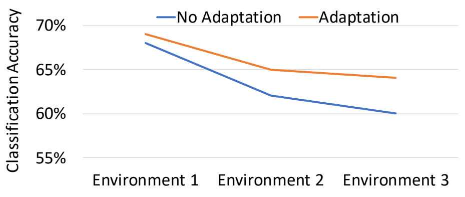

Fig. 24. Performance gain due to adaptation.

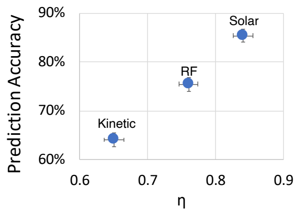

Fig. 25. Validation of η-Factor.

11.3 Adapting the k-Means Classifiers

In Zygarde, we adapt 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, we implement a simple strategy where the cluster centroids are updated by taking the weighted average of the current centroid and the new data point. By assigning more weights to the current centroid, we ensure 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. We conduct an experiment to quantify this. We first divide the ESC-10 audio dataset in to 80% training and 20% testing subsets. Then we 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. We 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). We repeat this experiment with and without the cluster adaptation step of Zygarde. Figure 24 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 [62, 97] has to be employed by adding an extra layer of computation prior to the clustering step.

11.4 Limitation of η Factor

In Zygarde, we estimate the η-factor offline, using energy harvesting traces of the systems that we experiment with. The modeling accuracy of η largely depends on the length of this empirical study – which must be long enough to capture the variability in harvested energy. Note that this variability depends not only on the energy source, but also on how the system is used by a user or how/where it is deployed. Hence, prior knowledge about the system's energy usage pattern and/or the experience of the system designer are crucial to determining a reasonable study duration to obtain an accurate estimate of the η-factor.

A limitation of the offline estimator for η in Zygarde is that there is always a possibility that the estimate is off when the system is deployed in the wild. To deal with this scenario, the accuracy of η could be assessed at runtime, and then η could be updated via an online or an offline re-estimation process. Such an assessment is possible since η is used by the system to predict the energy state of the next time slot, for which, the system observes the ground truth immediately after the current time slot, and thus, the system can precisely compute the prediction error at runtime. Depending on the amount of this error, η can be adapted to $\eta \pm \delta\eta$, where $\delta\eta$ is proportional to the prediction error.

In our experiments, however, we do not have to perform such adaptations as the empirically derived values for η have been fairly accurate. Here, the accuracy of estimation refers to how closely we are able to characterize the randomness in intermittent energy, as opposed to accurately predicting the harvested energy. In Figure 25, we show that the estimated value of η for three harvesters (used in Section 3) 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 fairly accurate.

11.5 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, we only provide a necessary condition for schedulability analysis (Section 5.3), 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. We leave it as one of our future works. Third, the queue size has a significant effect on the scheduler. Due to memory limitations, we cannot implement a longer queue (in Section 8, the queue size is 3). If the queue size is smaller (e.g., 1), the scheduler will only schedules the mandatory portions.

12 CONCLUSION

In this paper, we introduce a deadline-aware DNN runtime framework for intermittent systems. First, We devise a generic metric, the η factor, that models the predictability of an energy harvesting system. Second, we propose a DNN construction and execution technique which adapts the DNN inference process at runtime, and decreases the execution time by 5%-26%. Finally, we utilize 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. We also derive a necessary condition for scheduling real-time imprecise DNN tasks on intermittently-powered systems.

ACKNOWLEDGMENTS

This paper was supported, in part, by NSF grants CNS-1816213 and CNS-1704469.

REFERENCES

[1] 2014. EnergyTrace++. http://www.ti.com/tool/ENERGYTRACE
[2] 2015. Arducam Mini. https://www.arducam.com/product/arducam-2mp-spi-camera-b0067-arduino/
[3] 2015. LTC3105 step up DC/DC converter. https://www.analog.com/media/en/technical-documentation/data-sheets/3105fb.pdf
[4] 2015. MSP MCU IQmathLib Users Guide. http://www.ti.com/tool/MSP-IQMATHLIB
[5] 2015. Timer Module. https://datasheets.maximintegrated.com/en/ds/DS3231.pdf
[6] 2016. Powercast P2210B. https://www.powercastco.com/wp-content/uploads/2016/12/P2110B-Datasheet-Rev-3.pdf
[7] 2016. Powercaster Transmitter. http://www.powercastco.com/wp-content/uploads/2016/11/User-Manual-TX-915-01-Rev-A-4.pdf
[8] 2018. Adafruit Electret Microphone. shorturl.at/yIX03
[9] 2018. Evaluation of ESC dataset with different algorithms. https://github.com/karoldvl/ESC-50
[10] 2018. MSP430FR5994. http://www.ti.com/lit/ds/symlink/msp430fr5994.pdf
[11] 2020. Downsampled VWW. https://bitbucket.org/Bashima/vww_dataset/
[12] 2020. Flexible Ethylene Tetrafluoroethylene (ETFE) based solar panel. shorturl.at/almrC
[13] 2020. GCC Instruction Timing. http://mspgcc.sourceforge.net/manual/x528.html.
[14] Martin Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, Manjunath Kudlur, Josh Levenberg, Rajat Monga, Sherry Moore, Derek G. Murray, Benoit Steiner, Paul Tucker, Vijay Vasudevan, Pete Warden, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. 2016. TensorFlow: A system for large-scale machine learning. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16).*
[15] Ethem Alpaydin. 2020. *Introduction to machine learning.* MIT press.
[16] David Arthur and Sergei Vassilvitskii. 2006. *k-means++: The advantages of careful seeding.* Technical Report. Stanford.
[17] Antonio Asaro, Indra Laksono, James Doyle, and Gordon F Grigor. 2000. Method and apparatus for improved double buffering. US Patent 6,100,906.
[18] Domenico Balsamo, Alex S Weddell, Anup Das, Alberto Rodriguez Arreola, Davide Brunelli, Bashir M Al-Hashimi, Geoff V Merrett, and Luca Benini. 2016. Hibernus++: a self-calibrating and adaptive system for transiently-powered embedded devices. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 35, 12 (2016), 1968–1980.
[19] Domenico Balsamo, Alex S Weddell, Geoff V Merrett, Bashir M Al-Hashimi, Davide Brunelli, and Luca Benini. 2015. Hibernus: Sustaining computation during intermittent supply for energy-harvesting systems. *IEEE Embedded Systems Letters* 7, 1 (2015), 15–18.

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

[21] Sanjoy Baruah, Haohan Li, and Leen Stougie. 2010. Towards the design of certifiable mixed-criticality systems. In *2010 16th IEEE Real-Time and Embedded Technology and Applications Symposium*. IEEE, 13–22.

[22] Sanjoy K Baruah, Alan Burns, and Robert I Davis. 2011. Response-time analysis for mixed criticality systems. In *2011 IEEE 32nd Real-Time Systems Symposium*. IEEE, 34–43.

[23] Sugato Basu, Arindam Banerjee, and Raymond Mooney. 2002. Semi-supervised clustering by seeding. In *In Proceedings of 19th International Conference on Machine Learning (ICML-2002*. Citeseer.

[24] Soroush Bateni and Cong Liu. 2018. ApNet: Approximation-Aware Real-Time Neural Network. In *2018 IEEE Real-Time Systems Symposium (RTSS)*. IEEE, 67–79.

[25] Yoshua Bengio, Aaron Courville, and Pascal Vincent. 2013. Representation learning: A review and new perspectives. *IEEE transactions on pattern analysis and machine intelligence* 35, 8 (2013), 1798–1828.

[26] Sourav Bhattacharya and Nicholas D Lane. 2016. 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*. ACM, 176–189.

[27] Naveed Bhatti and Luca Mottola. 2016. Efficient state retention for transiently-powered embedded sensing. In *International Conference on Embedded Wireless Systems and Networks*. 137–148.

[28] Paul E Black. 1998. *Dictionary of algorithms and data structures*. Technical Report.

[29] Tolga Bolukbasi, Joseph Wang, Ofer Dekel, and Venkatesh Saligrama. 2017. Adaptive neural networks for efficient inference. *arXiv preprint arXiv:1702.07811* (2017).

[30] Michael Buettner, Ben Greenstein, and David Wetherall. 2011. Dewdrop: an energy-aware runtime for computational RFID. In *Proc. USENIX NSDI*.

[31] Giorgio Buttazzo, Marco Spuri, and Fabrizio Sensini. 1995. Value vs. deadline scheduling in overload conditions. In *Proceedings 16th IEEE Real-Time Systems Symposium*. IEEE, 90–99.

[32] Chun-Hung Cheng, Ada Waichee Fu, and Yi Zhang. 1999. Entropy-based subspace clustering for mining numerical data. In *Proceedings of the fifth ACM SIGKDD international conference on Knowledge discovery and data mining*. 84–93.

[33] Hao Cheng, Kien A Hua, and Khanh Vu. 2008. Constrained locally weighted clustering. *Proceedings of the VLDB Endowment* (2008).

[34] François Chollet. 2017. Xception: Deep learning with depthwise separable convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 1251–1258.

[35] Aakanksha Chowdhery, Pete Warden, Jonathon Shlens, Andrew Howard, and Rocky Rhodes. 2019. Visual Wake Words Dataset. arXiv:cs.CV/1906.05721

[36] Alexei Colin and Brandon Lucia. 2016. Chain: tasks and channels for reliable intermittent programs. *ACM SIGPLAN Notices* (2016).

[37] Alexei Colin and Brandon Lucia. 2018. Termination checking and task decomposition for task-based intermittent programs. In *Proceedings of the 27th International Conference on Compiler Construction*. ACM, 116–127.

[38] M Courbariaux and Y Bengio. 2017. BinaryNet: Training deep neural networks with weights and activations constrained to+ 1 or- 1. arXiv: 1602.02830, 2016.

[39] Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David. 2015. Binaryconnect: Training deep neural networks with binary weights during propagations. In *Advances in neural information processing systems*. 3123–3131.

[40] Andrea Crovetto, Fei Wang, and Ole Hansen. 2014. Modeling and optimization of an electrostatic energy harvesting device. *journal of microelectromechanical systems* 23, 5 (2014), 1141–1155.

[41] George Cybenko. 1989. Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems* (1989).

[42] Daniel de Godoy, Bashima Islam, Stephen Xia, Md Tamzeed Islam, Rishikanth Chandrasekaran, Yen-Chun Chen, Shahriar Nirjon, Peter R Kinget, and Xiaofan Jiang. 2018. Paws: A wearable acoustic system for pedestrian safety. In *2018 IEEE/ACM Third International Conference on Internet-of-Things Design and Implementation (IoTDI)*. IEEE, 237–248.

[43] Lieven De Lathauwer, Bart De Moor, and Joos Vandewalle. 2000. A multilinear singular value decomposition. *SIAM journal on Matrix Analysis and Applications* 21, 4 (2000), 1253–1278.

[44] Lieven De Lathauwer, Bart De Moor, and Joos Vandewalle. 2000. On the best rank-1 and rank-(r 1, r 2,..., rn) approximation of higher-order tensors. *SIAM journal on Matrix Analysis and Applications* 21, 4 (2000), 1324–1342.

[45] Christopher De Sa, Matthew Feldman, Christopher Ré, and Kunle Olukotun. 2017. Understanding and optimizing asynchronous low-precision stochastic gradient descent. In *ACM SIGARCH Computer Architecture News*, Vol. 45. ACM, 561–574.

[46] Jasper de Winkel, Carlo Delle Donne, Kasim Sinan Yildirim, Przemysław Pawełczak, and Josiah Hester. 2020. Reliable Timekeeping for Intermittent Computing. In *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*. 53–67.

[47] Inderjit S Dhillon, Yuqiang Guan, and Jacob Kogan. 2002. Iterative clustering of high dimensional text data augmented by local search. In *2002 IEEE International Conference on Data Mining, 2002. Proceedings*. IEEE, 131–138.

[48] François Dorin, Pascal Richard, Michaël Richard, and Joël Goossens. 2010. Schedulability and sensitivity analysis of multiple criticality tasks with fixed-priorities. *Real-Time Systems* 46, 3 (2010), 305–331.

[49] Ajay Dudani, Frank Mueller, and Yifan Zhu. 2002. Energy-conserving feedback EDF scheduling for embedded systems with real-time constraints. In *ACM SIGPLAN Notices*, Vol. 37. ACM, 213–222.

[50] Reza Ebrahimzadeh and Mahdi Jampour. 2014. Efficient handwritten digit recognition based on histogram of oriented gradients and SVM. *International Journal of Computer Applications* 104, 9 (2014).

[51] Michael Figurnov, Maxwell D Collins, Yukun Zhu, Li Zhang, Jonathan Huang, Dmitry P Vetrov, and Ruslan Salakhutdinov. 2017. Spatially Adaptive Computation Time for Residual Networks.. In *CVPR*, Vol. 2. 7.

[52] Blesson George. 2017. A study of the effect of random projection and other dimensionality reduction techniques on different classification methods. *Baselius Researcher* (2017), 201769.

[53] Graham Gobieski, Nathan Beckmann, and Brandon Lucia. 2018. Intermittent Deep Neural Network Inference. *SysML* (2018).

[54] Graham Gobieski, Brandon Lucia, and Nathan Beckmann. 2019. Intelligence Beyond the Edge: Inference on Intermittent Embedded Systems. In *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*. ACM, 199–213.

[55] Song Han, Xingyu Liu, Huizi Mao, Jing Pu, Ardavan Pedram, Mark A Horowitz, and William J Dally. 2016. EIE: efficient inference engine on compressed deep neural network. In *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*.

[56] Song Han, Huizi Mao, and William J Dally. 2015. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149* (2015).

[57] Boris Hanin. 2017. Universal function approximation by deep neural nets with bounded width and relu activations. *arXiv preprint arXiv:1708.02691* (2017).

[58] Nicholas JA Harvey, John Dunagan, Mike Jones, Stefan Saroiu, Marvin Theimer, and Alec Wolman. 2002. Skipnet: A scalable overlay network with practical locality properties. (2002).

[59] Josiah Hester, Kevin Storer, and Jacob Sorber. 2017. Timely Execution on Intermittently Powered Batteryless Sensors. In *Proceedings of the 15th ACM Conference on Embedded Network Sensor Systems*. ACM, 17.

[60] Josiah Hester, Nicole Tobias, Amir Rahmati, Lanny Sitanayah, Daniel Holcomb, Kevin Fu, Wayne P Burleson, and Jacob Sorber. 2016. Persistent clocks for batteryless sensing devices. *ACM Transactions on Embedded Computing Systems (TECS)* 15, 4 (2016), 77.

[61] Matthew Hicks. 2017. Clank: Architectural support for intermittent computation. In *2017 ACM/IEEE 44th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 228–240.

[62] Judy Hoffman, Eric Tzeng, Taesung Park, Jun-Yan Zhu, Phillip Isola, Kate Saenko, Alexei A Efros, and Trevor Darrell. 2017. Cycada: Cycle-consistent adversarial domain adaptation. *arXiv preprint arXiv:1711.03213* (2017).

[63] Kurt Hornik. 1991. Approximation capabilities of multilayer feedforward networks. *Neural networks* 4, 2 (1991), 251–257.

[64] Matthew B Hoy. 2018. Alexa, Siri, Cortana, and more: an introduction to voice assistants. *Medical reference services quarterly* (2018).

[65] Hanzhang Hu, Debadeepta Dey, Martial Hebert, and J Andrew Bagnell. 2019. Learning anytime predictions in neural networks via adaptive loss balancing. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 33. 3812–3821.

[66] Gao Huang, Danlu Chen, Tianhong Li, Felix Wu, Laurens van der Maaten, and Kilian Q Weinberger. 2017. Multi-scale dense networks for resource efficient image classification. *arXiv preprint arXiv:1703.09844* (2017).

[67] Gao Huang, Yu Sun, Zhuang Liu, Daniel Sedra, and Kilian Q Weinberger. 2016. Deep networks with stochastic depth. In *European conference on computer vision*. Springer, 646–661.

[68] Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. 2016. Binarized neural networks. In *Advances in neural information processing systems*. 4107–4115.

[69] Bashima Islam and Nirjon Shahriar. 2020. 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.

[70] Md Tamzeed Islam, Bashima Islam, and Shahriar Nirjon. 2017. Soundsifter: Mitigating overhearing of continuous listening devices. In *Proceedings of the 15th Annual International Conference on Mobile Systems, Applications, and Services*. ACM, 29–41.

[71] Md Tamzeed Islam and Shahriar Nirjon. 2019. SoundSemantics: exploiting semantic knowledge in text for embedded acoustic event classification. In *Proceedings of the 18th International Conference on Information Processing in Sensor Networks*. ACM, 217–228.

[72] Hrishikesh Jayakumar, Arnab Raha, and Vijay Raghunathan. 2014. QuickRecall: A low overhead HW/SW approach for enabling computations across power cycles in transiently powered computers. In *VLSI Design and 2014 13th International Conference on Embedded Systems, 2014 27th International Conference on*. IEEE, 330–335.

[73] Jinda Jia, Xiaobiao Shan, Deepesh Upadrashta, Tao Xie, Yaowen Yang, and Rujun Song. 2018. Modeling and Analysis of Upright Piezoelectric Energy Harvester under Aerodynamic Vortex-induced Vibration. *Micromachines* 9, 12 (2018), 667.

[74] Jeff Johnson. 2018. Rethinking floating point for deep learning. *arXiv preprint arXiv:1811.01721* (2018).

[75] Leonid Kantorovich. 1942. On translation of mass. In *Dokl. AN SSSR*, Vol. 37. 20.

[76] Sergey Karayev, Tobias Baumgartner, Mario Fritz, and Trevor Darrell. 2012. Timely object recognition. In *Advances in Neural Information Processing Systems*. 890–898.

[77] Sergey Karayev, Mario Fritz, and Trevor Darrell. 2014. Anytime recognition of objects and scenes. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 572–579.

[78] Eunwoo Kim, Chanho Ahn, and Songhwai Oh. 2018. Nestednet: Learning nested sparse structures in deep neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 8669–8678.

[79] Zvi Kons, Orith Toledo-Ronen, and M Carmel. 2013. Audio event classification using deep neural networks.. In *Interspeech*. 1482–1486.

[80] C Mani Krishna. 2001. Real-Time Systems. *Wiley Encyclopedia of Electrical and Electronics Engineering* (2001).

[81] Alex Krizhevsky, Geoffrey Hinton, et al. 2009. Learning multiple layers of features from tiny images. (2009).

[82] Yann LeCun, Corinna Cortes, and Christopher JC Burges. 1998. The MNIST database of handwritten digits, 1998. 10 (1998), 34. http://yann.lecun.com/exdb/mnist

[83] Chen Lee, John Lehoczky, Dan Siewiorek, Ragunathan Rajkumar, and Jeff Hansen. 1999. A scalable solution to the multi-resource QoS problem. In *Proceedings 20th IEEE Real-Time Systems Symposium (Cat. No. 99CB37054)*. IEEE, 315–326.

[84] Seulki Lee, Islam Bashima, Luo Yubo, and Shahriar Nirjon. 2020. Intermittent Learning: On-Device Machine Learning on Intermittently Powered System. In *Proceedings of ACM on Interactive, Mobile, Wearable, and Ubiquitous Computing, 2020*. ACM.

[85] Seulki Lee and Shahriar Nirjon. 2019. 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.

[86] Seulki Lee and Nirjon Shahriar. 2020. 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.

[87] Sam Leroux, Steven Bohez, Elias De Coninck, Tim Verbelen, Bert Vankeirsbilck, Pieter Simoens, and Bart Dhoedt. 2017. The cascading neural network: building the Internet of Smart Things. *Knowledge and Information Systems* 52, 3 (2017), 791–814.

[88] Haohan Li and Sanjoy Baruah. 2010. An algorithm for scheduling certifiable mixed-criticality sporadic task systems. In *2010 31st IEEE Real-Time Systems Symposium*. IEEE, 183–192.

[89] Tsung-Yi Lin, Michael Maire, Serge Belongie, Lubomir Bourdev, Ross Girshick, James Hays, Pietro Perona, Deva Ramanan, C. Lawrence Zitnick, and Piotr Dollár. 2014. Microsoft COCO: Common Objects in Context. arXiv:cs.CV/1405.0312

[90] Jane W.-S. Liu, Kwei-Jay Lin, Wei Kuan Shih, Albert Chuang-shi Yu, Jen-Yao Chung, and Wei Zhao. 1991. Algorithms for scheduling imprecise computations. In *Foundations of Real-Time Computing: Scheduling and Resource Management*. Springer, 203–249.

[91] Zhou Lu, Hongming Pu, Feicheng Wang, Zhiqiang Hu, and Liwei Wang. 2017. The expressive power of neural networks: A view from the width. In *Advances in Neural Information Processing Systems*. 6231–6239.

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

[93] Kaisheng Ma, Xueqing Li, Jinyang Li, Yongpan Liu, Yuan Xie, Jack Sampson, Mahmut Taylan Kandemir, and Vijaykrishnan Narayanan. 2017. Incidental computing on IoT nonvolatile processors. In *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture*. ACM, 204–218.

[94] Kaisheng Ma, Yang Zheng, Shuangchen Li, Karthik Swaminathan, Xueqing Li, Yongpan Liu, Jack Sampson, Yuan Xie, and Vijaykrishnan Narayanan. 2015. Architecture exploration for ambient energy harvesting nonvolatile processors. In *2015 IEEE 21st International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 526–537.

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

[96] Kiwan Maeng and Brandon Lucia. 2018. Adaptive Dynamic Checkpointing for Safe Efficient Intermittent Computing. In *13th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 18)*.

[97] Akhil Mathur, Anton Isopoussu, Fahim Kawsar, Nadia Berthouze, and Nicholas D Lane. 2019. 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*. 169–180.

[98] Elijah Meyer, Mark C Greenwood, and Tan Tran. 2016. Daily Step Count Profile Data for 61 Days [dataset]. (2016).

[99] Azalia Mirhoseini, Ebrahim M Songhori, and Farinaz Koushanfar. [n.d.]. Idetic: A high-level synthesis approach for enabling long computations on transiently-powered ASICs. In *2013 IEEE International Conference on Pervasive Computing and Communications*.

[100] Azalia Mirhoseini, Ebrahim M Songhori, and Farinaz Koushanfar. 2013. Automated checkpointing for enabling intensive applications on energy harvesting devices. In *Low Power Electronics and Design (ISLPED), 2013 IEEE International Symposium on*. IEEE, 27–32.

[101] Mohammad Moallemi and Gabriel Wainer. 2011. 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.

[102] Clemens Moser, Davide Brunelli, Lothar Thiele, and Luca Benini. 2006. Lazy scheduling for energy harvesting sensor nodes. In *IFIP Working Conference on Distributed and Parallel Embedded Systems*. Springer.

[103] Clemens Moser, Davide Brunelli, Lothar Thiele, and Luca Benini. 2007. Real-time scheduling for energy harvesting sensor nodes. *Real-Time Systems* (2007).

[104] Tarek M Nabhan and Albert Y Zomaya. 1994. Toward generating neural network structures for function approximation. *Neural Networks* 7, 1 (1994), 89–99.

[105] Preetum Nakkiran, Raziel Alvarez, Rohit Prabhavalkar, and Carolina Parada. 2015. Compressing deep neural networks using a rank-constrained topology. (2015).

[106] Augustus Odena, Dieterich Lawson, and Christopher Olah. 2017. Changing model behavior at test-time using reinforcement learning. *arXiv preprint arXiv:1702.07780* (2017).

[107] Karol J. Piczak. [n.d.]. ESC: Dataset for Environmental Sound Classification. In *Proceedings of the 23rd Annual ACM Conference on Multimedia* (2015-10-13). ACM Press, 1015–1018. https://doi.org/10.1145/2733373.2806390

[108] Karol J Piczak. 2015. Environmental sound classification with convolutional neural networks. In *2015 IEEE 25th International Workshop on Machine Learning for Signal Processing (MLSP)*. IEEE, 1–6.

[109] Amir Rahmati, Mastooreh Salajegheh, Dan Holcomb, Jacob Sorber, Wayne P Burleson, and Kevin Fu. 2012. 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*. USENIX Association, 36–36.

[110] Ragunathan Rajkumar, Chen Lee, John Lehoczky, and Dan Siewiorek. 1997. A resource allocation model for QoS management. In *Proceedings Real-Time Systems Symposium*. IEEE, 298–307.

[111] Benjamin Ransford, Jacob Sorber, and Kevin Fu. 2012. Mementos: System support for long-running computation on RFID-scale devices. *Acm Sigplan Notices* 47, 4 (2012), 159–170.

[112] Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. 2016. Xnor-net: Imagenet classification using binary convolutional neural networks. In *European Conference on Computer Vision*. Springer, 525–542.

[113] Joshua San Miguel, Karthik Ganesan, Mario Badr, and Natalie Enright Jerger. 2018. The EH model: Analytical exploration of energy-harvesting architectures. *IEEE Computer Architecture Letters* 17, 1 (2018), 76–79.

[114] Jürgen Schmidhuber. 2015. Deep learning in neural networks: An overview. *Neural networks* 61 (2015), 85–117.

[115] Himanshu Sharma, Ahteshamul Haque, and Zainul Jaffery. 2018. Modeling and Optimisation of a Solar Energy Harvesting System for Wireless Sensor Network Nodes. *Journal of Sensor and Actuator Networks* 7, 3 (2018), 40.

[116] Rashmi Sharma et al. 2014. Performance evaluation of new joint EDF-RM scheduling algorithm for real time distributed system. *Journal of Engineering* 2014 (2014).

[117] Qinfeng Shi, James Petterson, Gideon Dror, John Langford, Alex Smola, and SVN Vishwanathan. 2009. Hash kernels for structured data. *Journal of Machine Learning Research* 10, Nov (2009), 2615–2637.

[118] W-K Shih and Jane W-S Liu. 1992. On-line scheduling of imprecise computations to minimize error. In *Real-Time Systems Symposium, 1992*. IEEE, 280–289.

[119] Jacob Sorber, Alexander Kostadinov, Matthew Garber, Matthew Brennan, Mark D Corner, and Emery D Berger. 2007. Eon: a language and runtime system for perpetual systems. In *Proceedings of the 5th international conference on Embedded networked sensor systems*.

[120] Ravi Srinivasan, Md Tamzeed Islam, Bashima Islam, Zeyu Wang, Tamim Sookoor, Omprakash Gnawali, and Shahriar Nirjon. 2017. Preventive maintenance of centralized HVAC systems: use of acoustic sensors, feature extraction, and unsupervised learning. In *Proceedings of the 15th IBPSA Conference*. 2518–2524.

[121] J. Stallkamp, M. Schlipsing, J. Salmen, and C. Igel. 2012. Man vs. computer: Benchmarking machine learning algorithms for traffic sign recognition. *Neural Networks* 0 (2012), –. https://doi.org/10.1016/j.neunet.2012.02.016

[122] Christian Szegedy, Sergey Ioffe, Vincent Vanhoucke, and Alexander A Alemi. 2017. Inception-v4, inception-resnet and the impact of residual connections on learning. In *Thirty-First AAAI Conference on Artificial Intelligence*.

[123] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. 2015. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 1–9.

[124] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. 2016. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2818–2826.

[125] Surat Teerapittayanon, Bradley McDanel, and Hsiang-Tsung Kung. 2016. Branchynet: Fast inference via early exiting from deep neural networks. In *2016 23rd International Conference on Pattern Recognition (ICPR)*. IEEE, 2464–2469.

[126] Ravi Teja Mullapudi, William R Mark, Noam Shazeer, and Kayvon Fatahalian. 2018. Hydranets: Specialized dynamic architectures for efficient inference. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 8080–8089.

[127] Ledyard R Tucker. 1966. Some mathematical notes on three-mode factor analysis. *Psychometrika* 31, 3 (1966), 279–311.

[128] Joel Van Der Woude and Matthew Hicks. 2016. Intermittent computation without hardware support or programmer intervention. In *12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16)*. 17–32.

[129] Andreas Veit and Serge Belongie. 2018. Convolutional networks with adaptive inference graphs. In *Proceedings of the European Conference on Computer Vision (ECCV)*. 3–18.

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

[131] Min Wang, Baoyuan Liu, and Hassan Foroosh. 2017. Factorized Convolutional Neural Networks.. In *ICCV Workshops*. 545–553.

[132] Saining Xie and Zhuowen Tu. 2015. Holistically-nested edge detection. In *Proceedings of the IEEE international conference on computer vision*. 1395–1403.

[133] Bo Yang, Xiao Fu, Nicholas D Sidiropoulos, and Mingyi Hong. 2016. Towards k-means-friendly spaces: Simultaneous deep learning and clustering. *arXiv preprint arXiv:1610.04794* (2016).

[134] Ming Yang, Shige Wang, Joshua Bakita, Thanh Vu, F Donelson Smith, James H Anderson, and Jan-Michael Frahm. 2019. 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)*. IEEE, 305–317.

[135] Shuochao Yao, Yiran Zhao, Huajie Shao, Shengzhong Liu, Dongxin Liu, Lu Su, and Tarek Abdelzaher. [n.d.]. FastDeepIoT: Towards Understanding and Optimizing Neural Network Execution Time on Mobile and Embedded Devices. *arXiv preprint arXiv:1809.06970* ([n. d.]).

[136] Shuochao Yao, Yiran Zhao, Huajie Shao, Aston Zhang, Chao Zhang, Shen Li, and Tarek Abdelzaher. 2018. Rdeepsense: Reliable deep mobile computing models with uncertainty estimations. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies* 1, 4 (2018), 173.

[137] Shuochao Yao, Yiran Zhao, Huajie Shao, Chao Zhang, Aston Zhang, Dongxin Liu, Shengzhong Liu, Lu Su, and Tarek Abdelzaher. 2018. Apdeepsense: Deep learning uncertainty estimation without the pain for iot applications. In *2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 334–343.

[138] Shuochao Yao, Yiran Zhao, Aston Zhang, Lu Su, and Tarek Abdelzaher. 2017. 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*.

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

[140] Jiahui Yu and Thomas S Huang. 2019. Universally slimmable networks and improved training techniques. In *Proceedings of the IEEE International Conference on Computer Vision*. 1803–1811.

[141] Yao-Xue Zhang, Cun-Hao Fang, and Yong Wang. 2004. A feedback-driven online scheduler for processes with imprecise computing. *Journal of Software* (2004).

[142] Zhilu 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.

[143] Husheng Zhou, Soroush Bateni, and Cong Liu. 2018. 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)*. IEEE, 190–201.

[144] Ting Zhu, Abedelaziz 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.