# An Open-World Time-Series Sensing Framework for Embedded Edge Devices

Abdulrahman Bukhari, Seyedmehdi Hosseinimotlagh, and Hyoseung Kim
*University of California, Riverside*
abukh001@ucr.edu, shoss007@ucr.edu, hyoseung@ucr.edu

***Abstract*—The rapid advancement of IoT technologies has generated much interest in the development of learning-based sensing applications on embedded edge devices. However, these efforts are being challenged by the need to adapt to unforeseen conditions in an open-world environment. Updating a learning model suffers from the lack of training data as well as the high computational demand beyond that available on edge devices. In this paper, we propose an open-world time-series sensing framework for making inferences from time-series sensor data and achieving incremental learning on an embedded edge device with limited resources. The proposed framework is able to achieve two essential tasks, inference and learning, without requiring access to a powerful cloud server. We discuss the design choices made to ensure satisfactory learning performance and efficient resource usage. Experimental results demonstrate the ability of the system to incrementally adapt to unforeseen conditions and to effectively run on a resource-constrained device.**

***Index Terms*—IoT, embedded edge devices, time-series sensing, open-world learning**

## I. Introduction

Smart sensors have been widely used in environmental, industrial and personal level applications. With the advancement of computational power and machine learning algorithms, IoT systems are able to achieve smart sensing, from data acquisition to data inference, and report the outcomes to users in nearly a real-time manner. In health monitoring applications, user activities, e.g., number of steps, running, or standing, are reported using accelerometer and gyrometer sensors on smart watches or smartphones. Although such systems can provide accurate results for known classes, i.e., input data belonging to one of the classes of training data, they require access to a server to update the learning model due to the lack of ability to distinguish unseen data.

In conventional smart sensing applications, the raw sensor data acquired from the environment is transferred to an edge device for additional processing. Some applications require learning algorithms to grasp a higher level of interaction between the sensor data and the application requirements. For example, Synthetic Sensors [1] collect and process raw data on an embedded device before sending it to the server for SVM-based learning and inference; DeepSense [2] can be applied to accelerometer and gyroscope data to learn and recognize human activities through a deep neural network (DNN). If these sensing frameworks encounter unlabeled data that belong to a new, unseen class by the learning model, it will be incorrectly classified as one of the existing classes. Finding new classes in the environment and incorporating them into the model has traditionally been performed by manually labeling new data and retraining the whole model using both old and new labeled data.

Updating a model over time is an active research problem, called incremental learning, and it has been widely studied especially for vision applications. Although incremental learning algorithms such as iCaRL [3] can add new classes and update network parameters for them, those new classes must be identified by the user and unseen data must be labeled accordingly before each model update. Thus, in this paper, we specifically call them *supervised incremental learning*. These limitations introduce an open-world learning problem [4], where the system has to differentiate by itself whether new data belong to an unknown class or a known class and update the model accordingly without sacrificing the inference performance. Since open-world learning methods do not require any human intervention, we call them *unsupervised incremental learning*. The open-world algorithms also have received a huge interest within the image classification community.

Recent works in both supervised and unsupervised incremental learning have shown significant performance advancement in image classification. However, most of these approaches are computationally hungry and require powerful machines for training. Additionally, they critically depend on the base machine-learning model, e.g., a DNN, in order to run effectively on the given application. To extend the sensing framework to incrementally learn new classes in both supervised and an unsupervised fashions, this paper proposes OpenSense, an open-world sensing framework for resource-constrained embedded edge devices. OpenSense can identify (thus reject from inference) unknown samples, assign new classes to these rejected samples, incorporate the new classes and update the model incrementally on an edge device. The contributions of this work are as follows:

- We present the OpenSense framework that can run different incremental learning algorithms for both supervised and unsupervised time-series sensing data problems.
- We propose an efficient DNN architecture called sDNN, which outperforms the state-of-art architecture in both inference performance and resource efficiency for time-series activity classification.
- We demonstrate the implementation of OpenSense on a resource-constrained edge device and its effectiveness in open-world incremental learning of time-series data.

## II. Related Work

### A. Supervised Incremental Learning

Incremental learning techniques have to overcome two challenges to maintain their performance. The first challenge is to learn new classes from a stream of data without "catastrophic forgetting" [5], in which the new data causes the neural network to forget what has been learned from the previous data. Early attempts made by [6], [7] tackle this challenge by expanding the neural network model as more classes are learned. However, these approaches introduce a new challenge of ever-increasing computation and memory demands as the model will keep growing incrementally, which is particularly problematic for embedded systems.

A number of approaches have been proposed to overcome the aforementioned problems of incremental learning. Few-Shots Class-Incremental Learning (FSCIL) [8] aims to learn both new and old classes with a limited number of training data, which limits the capacity of the model while avoiding forgetting observed classes in a stream of data. iCaRL [3] can learn new classes with a fixed feature representation based on the nearest mean classifier algorithm and the distillation loss to maintain the performance without forgetting. Due to the nature of these approaches, they perform well only under the assumption that the data set used for training is labeled by the user for both old and new classes.

### B. Unsupervised Incremental Learning

Contrary to the supervised approaches, classifiers for unsupervised incremental learning (a.k.a. open-world learning) must be able to distinguish unknown (unlabeled) samples in the input data stream and classify them as new classes, while maintaining the performance of classifying old classes. Early works [9], [10] have established a stepping stone toward this. In [9], the authors introduce the open-set problem for DNNs and propose a layer called OpenMax that extends the softmax layer to find the likelihood that an input sample is of an unknown class. A more sophisticated scheme called the Extreme Value Machine (EVM) is proposed by Ethan et al. [11] based on the Extreme Value Theorem [12]. EVM can fit an initial set of data into extreme vectors (EVs) using the Weibull distribution. Once EVM is trained, it can be updated incrementally by obtaining new EVs. In addition to its inference performance in open-world settings, the major advantage of EVM is the ability to limit the size of the model by setting the number of EVs in the model.

In [4], the authors extend the concept of EVM and re-define the evaluation protocol for open-world learning. They claim that an open-world learner must recognize both known and unknown classes and classify the unknown classes into new classes without forgetting classes previously learned. They also proposed a new metric, the Open-World Metric (OWM), since the testing set should include both known and unknown classes, thereby requiring a novel measurement that captures both the accuracy of the known classes and the classification of unknown classes. The authors in [13] introduce the notion of self-supervised features for open-world learning in order to avoid the overlap between known and unknown classes in the supervised feature space. Another work by Joseph et al. [14] extends the open-world learning to the problem of identifying an unknown object in images, by using an energy-based model [15] to create a separation between known and unknown objects in the energy space.

Although these approaches have shown acceptable performance in image classification, to the extent of our knowledge, none has been applied to time-series multisensor IoT sensing applications which this paper focuses on.

### C. Time-Series Sensing Frameworks

Extending supervised or unsupervised incremental learning to sensing frameworks requires a state-of-the-art classifier that on one hand has a high inference performance across different applications, and on the other hand is deployable on embedded edge devices such as Raspberry Pi.

There are mainly two approaches for sensing applications: special-purpose [16]–[20] and general-purpose sensing [1], [21]–[24]. The special-purpose sensing approach uses a single sensor to measure one aspect of the environment, e.g., temperature. Setting a threshold in these sensors is usually sufficient for controlling or monitoring applications and does not require complex learning-based techniques. The general-purpose sensing approach uses a combination of two or more sensors to recognize multiple events or activities in the scene, often through machine learning. Laput et al. [1] propose a custom sensor tag including 9 sensors to detect 38 different events, such as kettle on/off, door open/close, phone ringing, etc., using a simple SVM trained for each event. DeepSense [2] is a DNN-based time-series mobile sensing framework that can be applied to regression and classification problems by extracting both spatial and temporal features/relationships using CNN and RNN. DeepSense outperforms other traditional learning methods in different classification problems such as human activity recognition and user identification. However, it requires longer training time due to the complexity of the architecture and does not support incremental learning in open-world settings. We address these limitations in this paper.

## III. OpenSense Framework

The main objectives of the proposed OpenSense framework are to characterize raw time-series sensor data into informative labels to the user and to incrementally learn new classes that have not been seen by the classifier in both supervised and unsupervised manners. The framework consists of two units: a sensor board attached to the point of interest to acquire data from the application environment, and a resource-constraint edge device that is responsible for performing inference and incrementally learning novel classes as shown in Fig. 1. Unlike traditional time-series sensing frameworks, this framework does not require a cloud server to learn new classes and update the model. Except for the initial DNN training, all procedures are running on the edge device.

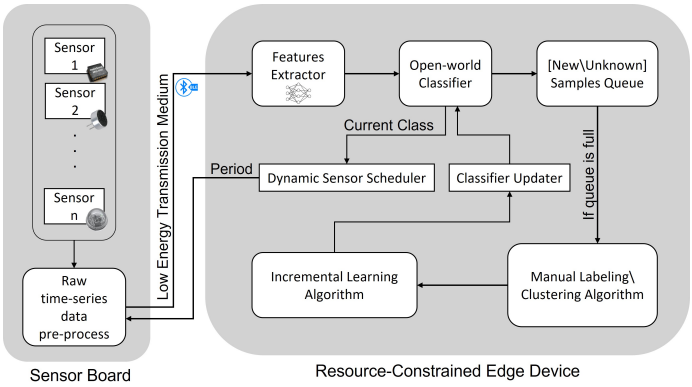



Fig. 1: Block diagram of the proposed OpenSense framework

### A. Sensor Board and Edge Device

**Sensor Board:** The sensor board collects and preprocesses raw sensor data samples, and then transmits them periodically in a specific size of window to the edge device. There can be one or more sensor boards in the scene. Each sensor board consists of a single sensor or an array of sensors and a low energy transmission medium with a processing unit to buffer and prepare the raw time-series data. The number and type of sensors in the board is decided based on application requirements. Multiple different sensors are used in most recent applications; for example, a combination of accelerometer, microphone, and illumination sensors enable capturing events in the kitchen, such as whether the faucet, kettle, microwave, air vent are running or turned off.

The processing unit prepares raw data based on the frequency of the corresponding sensor to reduce noise and overhead on data transmission without impacting the overall framework performance. A simple moving average window has shown reliable performance for sensors with high sampling frequencies, e.g., accelerometer. After prepossessing the sensors data, they are stacked together and transmitted periodically to the edge device through a low-energy data transmission medium, such as Bluetooth Low Energy (BLE). The transmission period between the sensor board and the edge device is set by the dynamic scheduler (Sec. III-E).

**Edge Device:** The resource-constraint edge device is the heart of the framework. When sensor data samples are received, the edge device processes them for *feature extraction* using a DNN trained network (Sec. III-B). The *open-world classifier* (Sec. III-C) then uses these features to distinguish whether a sample belongs to an existing class provided by the pretrained dataset or it is a new or an unknown class that has not been seen the classifier. Such samples are collected in a queue to be either manually labeled (supervised) or clustered into different classes (unsupervided) for the classifier to incrementally learn them through an incremental learning algorithm (Sec.III-D). In addition, the dynamic sensor scheduler (Sec. III-E) and the classifier updater (Sec. III-F) are included to change the sensor transmission period and trigger model updates, respectively. The rest of this section describes each component in detail.

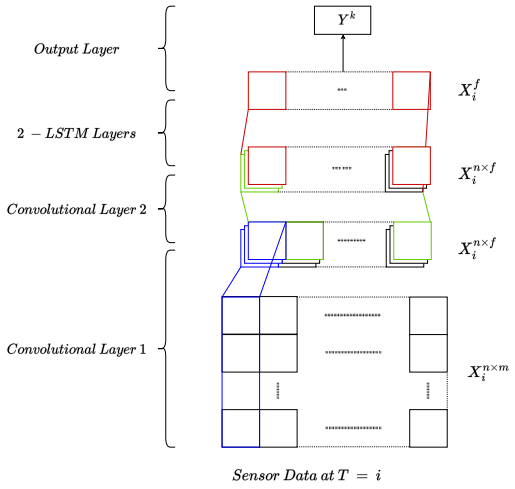



Fig. 2: sDNN architecture

### B. Feature Extraction

The data received from the sensor board requires further preprocessing before extracting features for inference and novel class detection. Time-series data from different sensors are processed based on the sampling rate of the corresponding sensor. Fast-Fourier Transform (FFT) is applied to data acquired from high-sampling sensors to capture the frequency patterns regardless of time dependency [2]. Statistical information, such as mean, variance, and peaks, is used for low-frequency sensors data like heart-rate and light-intensity sensors.

We perform feature extraction based on our simple DNN architecture, called sDNN, which can extract reliable features from time-series sensor data while providing runtime efficiency in inference and training. Fig. 2 shows the sDNN architecture. The two convolutional layers first capture spatial relationships between preprocessed sensors data, and the following LSTM layers capture temporal relationships between time-series data slices. The input size of sDNN depends on the preprocessed sensor data, where $n$ is the number of time-series slices and $m$ is the length of each slice. Compared to other state-of-the-art architectures, such as DeepSense [2], our proposed DNN ensures a fixed-size model and a feature vector with length $f$, where $f$ is the filter size of convolution and LSTM layers. On the other hand, the number of layers in DeepSense and the length of extracted features depend on the number of sensors on the device, making it more complex and longer as more sensors are used. Despite its simplicity, sDNN can provide competitive and often better inference accuracy than DeepSense as we will show in the evaluation. Note that such properties are important for incremental learning, especially on embedded edge devices.

sDNN is trained using available training datasets before deployment. The trained model is then used at runtime to extract features from preprocessed sensor data for the open-world classifier, by taking the output of the last layer in the DNN before the dense layer. The features extractor is also used for the initial training dataset to train the open-world classifier.

### C. Open-World Classifier

The purpose of an open-world classifier is not only to recognize data samples that belong to known classes, but also to recognize unknown samples and reject them from inference. The open-world classifier is initially trained on the features extracted from the training datasets and deployed to the edge device.

The built-in classifier of sDNN might be sufficient if there is no unknown sample. However, in the presence of unknown samples, it can misclassify them into one of the known classes, making it unable to use for open-world classification. To address this issue, we adopt EVM [11] as the open-world classifier of our framework. The reasons why we chose EVM here are that (i) it can be used for both supervised and unsupervised incremental learning scenarios, (ii) it avoids the need to update the entire DNN model, mitigating the catastrophic forgetting phenomena, and (iii) it has been shown to be effective in rejecting the samples that belong to unknown classes. We take the last layer of sDNN before dense layers as the input features for EVM. Then the open-world classifier notifies classification results to the user if given samples belong to known classes, and gives feedback to the sensor dynamic scheduler. The samples rejected by the classifier (unknown samples) are placed into a queue for manual labeling under supervised learning and for clustering under unsupervised learning.

### D. Incremental Learning Process

There are three criteria that an incremental learning framework must meet for practical applications:

1) the model can be trained from a stream of data introducing new classes to the classifier over time.
2) the inference performance must not be significantly lowered, especially due to the catastrophic forgetting phenomena [5].
3) updating the model must meet the resource requirements of the system since the framework should be able to run on embedded devices

For supervised learning, the samples in the queue are manually labeled and then we can update the classifier. For unsupervised learning, we need to identify and assign new classes to the unknown samples in the queue via a clustering algorithm before updating the classifier. Since this is a clustering problem where the number of groups is unknown, our framework uses the Finch algorithm [25] which is a parameter-free clustering method. The Finch algorithm provides different partitioning results for the given samples. Then our framework chooses the partition with the minimum number of clusters and only select clusters with at least a certain number of samples as new classes and label them accordingly [4]. The open-world model updater will incorporate these new classes into our classifier based on EVM.

One advantage of using the EVM-based classifier is that it only needs the extracted features of the new samples to update the classifier without the need to retrain the entire DNN, enabling more effective and efficient incremental learning.

**Algorithm 1:** Sensor Dynamic Scheduler

**Input :** $T_e$: All time intervals for class $C$
$CL$: User allowable classification latency
**Output:** $T_{sp}$: Sensor idle period for class $C$

$T_e \leftarrow AscendingSort(T_e)$
$T_{sp} \leftarrow FindMinimum(T_e)$
$L1 \leftarrow T_{sp}$
$L2 \leftarrow Length(T_{sp})$
$i \leftarrow 0$; $j \leftarrow 0$
**for** $i \leq L1$ **do**
  **for** $j \leq L2$ **do**
    $n \leftarrow Ceil(T_e[j]/T_{sp})$
    $Thresholds \leftarrow T_e[j] + CL$
    **if** $n \times T_{sp} \geq Threshold$ **then**
      $T_{sp} \leftarrow T_{sp} - 1$
      break
    **end**
    $j++$
  **end**
  $i++$
**end**
**if** $T_{sp} > 1$ **then**
  return $T_{sp}$
**else**
  return fail
**end**

Another advantage is controlling the model size by selecting the minimum number of points needed to cover a known class, which is known as the set cover problem [26] and important for incremental learning on resource-constrained devices.

### E. Sensor Dynamic Scheduler

The sensor dynamic scheduler is responsible for assigning the transmission period between the sensor board and the edge device such that the sensor board can be idle for the longest possible period to minimize the energy consumption of the sensor board while freeing resources on the edge device for open-world learning tasks. We propose a scheduler based on the classification such that the idle period does not exceed a given classification latency constraint (CL) that is the maximum time for the current event class to change to a different class while the sensor is idling. CL imposes the following condition to be satisfied:

$$n \times T_{sp} - T_e \leq CL \tag{1}$$

where $T_{sp}$ is the idle sensor period assigned by the scheduler for a current event, $T_e$ is the time when the current event actually ends, and $n$ is the number of times the idling period has repeated until the classifier recognizes a new event. In order to find $T_{sp}$ that does not exceed CL, a naïve approach would be using the greatest common factor of all occurrence intervals of an event, which would result in only one second of period for most cases. Therefore, we propose an algorithm that approximates the largest factor of all periods and guarantees the $CL$ condition to be met.

Alg. 1 gives the pseudo code of our algorithm. It selects the minimum time interval among all time intervals for a class $C$ as a base idle period, $T_{sp}$. The base period is compared with all other time intervals such that the difference after $n$ cycles does not exceed $CL$. If the base period $T_{sp}$ does not meet this

**Algorithm 2:** Model Update Scheduler

**Input:** $T_{sp}$: The sensor period for a given class
$N_u$: # of samples of the new discovered class
$S_u$: Samples from the new discovered class

$N_{old} \leftarrow 0$
**while** $N_u \neq 0$ **do**
  **if** $T_{sp} \geq T_{min}$ **then**
    $N_{ST} \leftarrow ComputeSamplesToTrain(T_{sp})$
    $UpdateTheModel(S_u[N_{old} : N_{ST}])$
    **if** $N_u \geq N_{ST}$ **then**
      $N_{old} \leftarrow N_{ST}$
    **else**
      return success
    **end**
  **else**
    return fail /* wait for next $T_{sp}$ */
  **end**
**end**

condition, the algorithm decrements $T_{sp}$ by one and continues the search. If $T_{sp}$ becomes one or smaller, it means there is no feasible $T_{sp}$ that satisfies the user's latency requirement and thus the algorithm returns fail. Then, the user can increase $CL$ to increase the search space and rerun the algorithm. In the worst case where no feasible $T_{sp}$ is found, the user may decide to set $CL$ to the minimum value of one, which ensures $T_{sp}$ to be at least two, i.e., $CL = 1$ and $T_{sp} = 2$.[1] This still ensures the sensor board to be idle for at least half the total operation time, reducing energy consumption by up to half.[2] The user has the freedom to choose a single $CL$ for all events or specify different values for individual events based on the importance and tolerated latency.

### F. Model and Classifier Updater

Once the unknown samples are clustered into new classes, given that the number of samples of the new class meets the minimum requirement, the model will be updated. However, due to the fact that the framework is designed for resource-constraint edge devices, we propose a model updating scheduler, Alg. 2, such that the model will be only updated partially with a certain number of samples from the new class based on the sensor transmission period set by the sensor dynamic scheduler. We fit a polynomial to the average execution time of updating the model for different number of samples to approximate the number of samples to update the model for the current sensor period, given that it meets the minimum average time to train at least one sample ($T_{min}$). Note that $T_{sp}$ may not satisfy the condition ($T_{sp} \geq T_{min}$). Therefore, different approaches to choose $T_{sp}$ can be used to extend the length of the sensor idling time without significantly compromising the latency performance of the scheduler, as can be seen in the last experiment in Sec. IV.

After the model is updated, the classifier updater will deploy the new open-world classifier to enable the framework to recognize the newly learned classes and discover new classes from the upcoming time-series data from sensor boards.

[1]With $T_{sp} = 2$, the difference between $n \times T_{sp}$ and any other number is always 0 or 1, thereby satisfying Eq. (1).

[2]The actual amount of energy savings in such a worst-case condition $T_{sp} = 2$ may vary depending on the idle energy consumption of the board.

## IV. Evaluation

There are two essential performance metrics to evaluate the overall performance of our proposed framework on resource-constrained edge devices: classification and incremental learning, and latency and energy consumption. For classification and incremental learning, we first compare the inference and efficiency performances of our proposed sDNN to DeepSense. After that we investigate the effectiveness of OpenSense in both supervised and unsupervised settings and compare it to other methods. We analyze the efficiency of each algorithm as an open-world learner by comparing the execution time of different tasks and evaluate our design choices on an embedded device. Lastly, we show the latency and energy consumption performances for different sensor dynamic scheduler approaches. In this section, we also provide more details on the evaluation platforms and the dataset.

### A. Evaluation Platforms

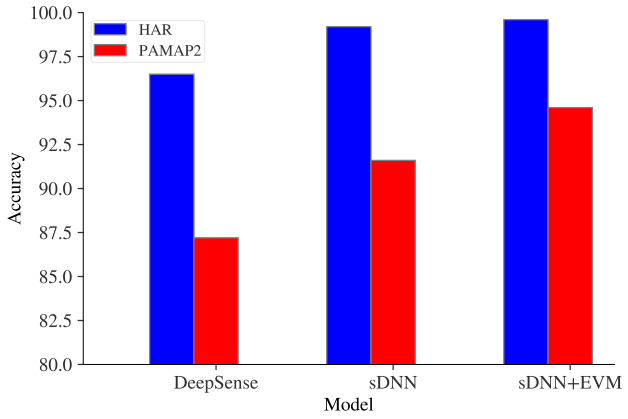
OpenSense was implemented and evaluated on a Raspberry Pi 4 Model B for the framework's runtime performance on a realistic embedded edge device. Experiments on model accuracy were conducted on a Linux-based machine equipped with Intel i7-8650U, 16GB memory and a dedicated NVIDIA GeForce GTX 1060 GPU with 6GB VRAM.

### B. Datasets

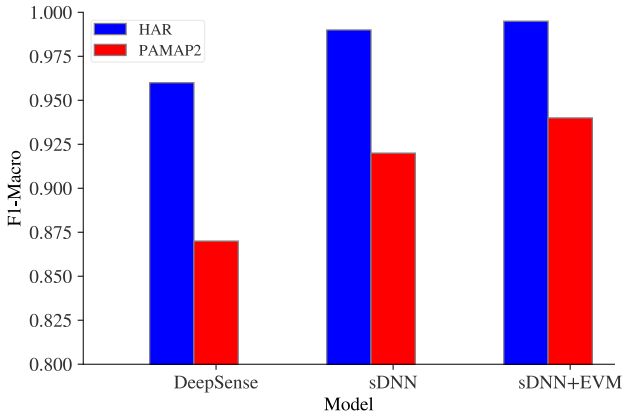
We used two datasets,HHAR and PAMAP2, that have been widely used in the literature of time-series human activity recognition. These datasets consist of time-series human activities data captured by different sensors and classified into distinct classes based on the activities of the wearer. The details of each dataset and preprocessing are as follow.

**HHAR:** The heterogeneous human activity recognition (HHAR) [27] dataset records 6 activities performed by 9 different users. The data is collected using the accelerometer and gyroscope embedded on smartphones and smartwatches. We followed the preprocessing suggested by DeepSense [2] to reproduce similar results for evaluation: time-series data is divided into 5-seconds non-overlapping windows, each window is divided into 0.25 second segments, and FFT is applied to each segment to compute the frequency response. The number of samples after preprocessing is around 120K and we divided them into training (70%), validation (10%) and testing (20%) sets.

**PAMAP2:** The physical activity monitoring data set (PAMAP2) [28] is collected using 3 inertial measurement units (IMU) sampled at 100Hz, each capturing temperature, acceleration, gyroscope and manometer, attached at wrist, chest and ankle of the user, and a heart sensor at 9Hz (total of 13 sensors data). The data is recorded by monitoring 18 activities performed by 9 users. The time-series data is divided into 1-second windows, each window is divided into ten segments each is 0.1 second. FTT is applied to each segment




(a) Accuracy




(b) F1-Macro Score

Fig. 3: Inference performance of DeepSense, sDNN, sDNN+EVM

TABLE I: DNN Model Efficiency

| Dataset | HHAR | | PAMAP2 | |
|---|---|---|---|---|
| DNN Model | DeepSense | sDNN | DeepSense | sDNN |
| #epochs to converge | 100 | 10 | 150 | 50 |
| average execution time \epoch | 37 sec | 16 sec | 18 sec | 3 sec |
| speedup \epoch | x2.3 | | x6 | |
| total training time | 61 min 40 sec | 2 min 31 sec | 45 min | 2 min 30 sec |

to calculate the frequency response. The number of samples after preprocessing is around 27K divided into training (60%), validation (20%) and testing (20%) sets since the number of samples is much smaller per class compared to HHAR.

Since both datasets are imbalanced, we perform under-sampling for each class in each dataset for the testing set to ensure that each class weighs equally to the performance, especially in the incremental learning experiments.

### C. Learning Algorithms to Compare

We evaluate OpenSense and compare it to 2 learning algorithms in both supervised and unsupervised settings.

**Naïve approach (NA):** The based DNN model is updated using only the data coming in the data stream without accessing previously trained samples.

**Fixed-representation Class Incremental Learning (FRCI) [3]:** In this algorithm the DNN model with the new samples in the data-stream but with the addition of exemplars learned based on the Nearest Mean Class to avoid Catastrophic Forgetting while incrementally learn without imposing an overhead on the system.

### D. Results: Classification and Learning Performance

**DNN Base Models and EVM:** The base DNN is very important to the proposed framework for building a sustainable features extractor, EVM model and a classifier for other incremental learning approaches. Therefore, we compare sDNN with a state-of-the-art architecture, DeepSense. The experiment is conducted on 2 different datasets. For DeepSense, we implemented it in Keras and successfully reproduced the results in the original work. We then compare it with sDNN, implemented on Keras as well, to produce consistent evaluation. We used fixed parameters in our comparison and used accuracy and F1-macro scores for evaluation, similar to DeepSense [2]. We also built an EVM model based on sDNN to evaluate it as a classifier. We used fixed parameters as well for EVM training in all experiments. We performed cross-validation in the early stage of development and found out that the following EVM parameters: tail-size = 100, cover threshold = 0.7 and distance multiplier = 0.4 produce acceptable results across the two time-series dataset we used. In this set of experiments, all models are trained with all classes as known classes and the training is performed on batches with all samples in the training dataset.

We can see that the proposed DNN, outperforms DeepSense on both HHAR and PAMAP2 datasets in Fig. 3. We can also see that sDNN can produce robust features, when freezing the last layer, for the EVM model which slightly provide better accuracy and F1-Macro score than the model on the same testing sets. In addition to that, sDNN is faster than DeepSense by up to x2.3 and x6 per epoch for HHAR and PAMAP2, respectively as shown in table.I. We can see that sDNN converges much faster than DeepSense on both datasets. Faster DNN training can allow for more efficient representation update in case we consider updating the base DNN in our framework, and in other supervised incremental learning approaches as well. When implementing DeepSense, each sensor in the sensor board will add three individual convolutional layers to the framework leading to a total of 12 and 45 layers for HHAR and PAMAP2, respectively compared to a fixed 5 layers in sDNN, two convolutional layers, two LSTM layers and an output layer. We assumed that due to the increased complexity of DeepSense, the training data is over-fitting the model, especially with PAMAP2 dataset that has limited number of samples and requires longer training time to converge.

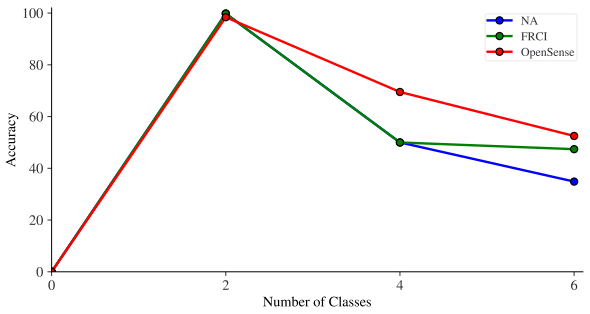
**Supervised Incremental Learning:** In the supervised incremental learning setup, the data-stream is divided into equally

batched samples with a specific number of known classes. The known classes are incoming as an increment of 2 classes per data-stream for HHAR and an increment of 3 classes for PAMAP2. For OpenSense, the base DNN is trained with the first batch of data, 2 and 3 classes from HHAR and PAMAP2 respectively. The purpose of this experiment is to compare the performance of OpenSense to NA and FRCI algorithms, in case we manually label new classes and incrementally update the model. We used accuracy as a metric to compare each algorithm since all samples are labeled.

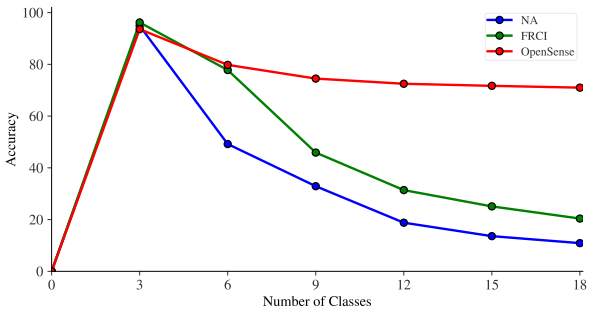
We can see in Fig 4 that all algorithms achieve around 99% and 95% accuracy for HHAR and PAMAP2, respectively, with both the naïve approach, noted as NA, and FRCI slightly better accuracy than OpenSense in the initial training phase. As more data coming in the stream, we can clearly see that OpenSense outperforms NA and FRCI on both datasets since OpenSense uses the features extracted by the base DNN model, in which the features representation is fixed for all class. The EVM model of OpenSense then use these features with labels and incrementally assign new EVM vectors to represent each class.

For the naïve approach and FRCI, their accuracy significantly drops to 50% in HHAR dataset with much better performance for FRCI in PAMAP2 dataset closely to OpenSense. NA accuracy keeps falling as it incrementally learning to 34.8% and 10.9% for HHAR and PAMAP2, respectively. We see that during testing after each increment in NA that it almost completely forgets any classes that are learned in the previous increment due the catastrophic forgetting problem. Although FRCI performance is similar to OpenSense on HHAR dataset, we see that, similar to NA, it reaches to a low 20.4% accuracy by the end of the incremental learning of PAMAP2 dataset. Updating the DNN model with a no or limited number of samples from previously learned classes leads to a degrade in its inference performance, making OpenSense more suitable for incremental learning.

**Open-World (Unsupervised) Incremental Learning:** In the unsupervised incremental learning setup, only the first batch of the training dataset consists of known classes. Unlike the supervised setup, we assume that initially we have a large number of known classes and incrementally discover new classes from the data-stream. Since HHAR dataset has a limited number of classes, namely six, it is not sufficient for the evaluation of this experiment. Therefore, we only performed this experiment on PAMAP2 where the initial DNN model is trained on the dataset of 9 known classes and the remaining class are incoming as 3 unknown classes in each data-stream. Since OpenSense is designed for open-world problem, we added a novel class detector and unsupervised clustering algorithm for the naïve approach and FRCI by setting a threshold on the output of the DNN, such that any sample with a probability less than the threshold will be rejected as an unknown sample. The threshold is chosen based on the probability history of previously discovered classes by taking the mean of the maximum probability of each prior sample. The rejected samples are collected in a queue to cluster them into novel classes and assign a label for each new




(a) HHAR




(b) PAMAP2

Fig. 4: Comparing the accuracy of the Naïve approach (NA), FRCI and OpenSense for each step in a supervised incremental learning set-up

class. The metric used in the unsupervised setup is the Open-World Metric (OWM) proposed in [4] defined as follows:

$$OWM = \frac{N_{KK} \cdot Acc(X_{KK}) \; + \; N_{UU} \cdot B3(X_{UU})}{N_{KK} + N_{KU} + N_{UK} + N_{UU}} \quad (2)$$

where $X$ is the targeted dataset, $N$ refers to the number of samples in the corresponding category, the subscripts $K$ and $U$ are abbreviations for known and unknown, respectively, e.g., $N_{KU}$ stands for known samples that are misclassified as unknown, $Acc()$ is the accuracy for the known data, and $B3()$ is the B3 score [29] for the unknown data.

Fig 5 shows that all algorithms are very well trained on the initial set with around 95% accuracy, since all classes are known at this stage. We can see that OpenSense outperforms the other approaches with large margins (0.48 and 0.3) compared to NA and FRCI. The OWM score slowly decreases, with a rate of 0.13 scores, as more unknown samples are discovered by OpenSense. The key to this performance is the ability of EVM-based classifier to reject unknown samples. This allows the clustering algorithm to produce clusters that actually represent unknown classes and update the model accordingly.

Since the naïve approach suffers from catastrophic forgetting, it is expected that its performance will degrade as more unknown samples are introduced. The NA open-world score was 0.11 after it incrementally exposed to all samples. Although it performs better than NA, FRCI still struggle to incrementally adapt new classes with an average decrease of

TABLE II: The average execution time for different tasks in the Open-World (Unsupervised) Incremental Learning experiment

| Task | Inference | | | Learning | | Total Session Time |
|---|---|---|---|---|---|---|
| | Feature Extraction | Classification | Queuing | Clustering | Model Updating | |
| Naïve Approach | 0.5 Sec | 12 mSec | 16 $\mu$Sec | 0.46 Sec | 17.4 Sec | 67.8 Sec |
| FRCI | 0.47 Sec | 35 mSec | 22 $\mu$Sec | 0.27 Sec | 16.5 Sec | 64.5 Sec |
| OpenSense | 0.49 Sec | 31 mSec | 18 $\mu$Sec | 0.13 Sec | 0.92 Sec | 6.1 Sec |

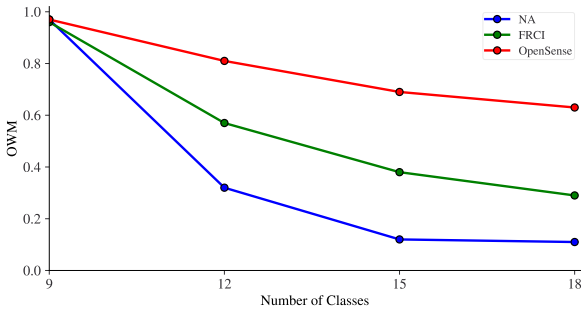



Fig. 5: Comparing the performance of NA, FRCI and OpenSense in an Open-world set

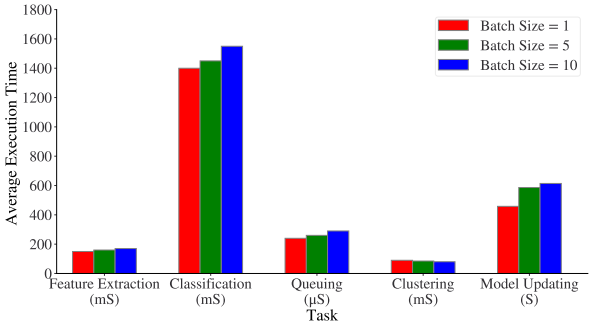



Fig. 6: The average execution time of different tasks on the Raspberry Pi for different batch sizes

0.23 scores as more unknown classes are coming. Moreover, the number of discovered classes are only 3 and 5 out of 9 classes for NA and FRCI, respectively, wherein OpenSense discovered 9 new classes.

There are many factors, in addition to the fact that most incremental learning works are developed in a supervised approach, that contribute to the expected low performance. One factor is due to the fact that the current unsupervised clustering algorithm, Finch, is unable to cluster the preprocessed time-series data from different classes, while using the features extracted from the base DNN provides better representation to the time-series data. Another important factor is the ability of classifier to correctly reject unknown samples. Even if the features representation is used in the clustering algorithm, it will fail to produce the right number of clusters, or clusters that corresponds to the right label if the rejected samples actually belong to known classes which will lead to confusion to the DNN when updating with new classes.

### *E. Results: Latency and Energy Consumption Performance*

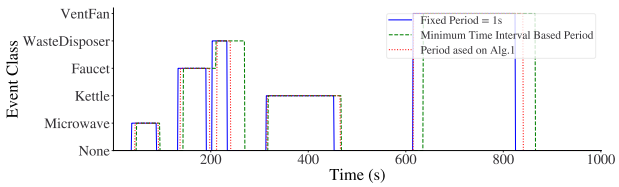
**Execution Time of Incremental Learning:** We compare the execution time of different tasks in the framework for the 3 incremental algorithms in the open-world learning experiment conducted above, as shown in Table II. The purpose of this experiment is to demonstrate that our design choices does not only outperform the naïve approach and FRCI in unsupervised incremental learning but also in latency. We compare the execution time of the essential tasks in the framework including preparing the samples for inference, novel class detection, queuing unknown samples, clustering and incrementally updating the model. We compare these latencies for each algorithm running on a Linux-based machine. We also compare the overall execution time needed to incrementally discover and adapt new classes for a complete dataset for each algorithm. We measure the execution time of each task in Seconds (s).

All experiments are conducted on the same dataset and with 100 samples per batch for the inference task. Note that the number of samples may differ in subsequent tasks depending on classification results. For example, the number of samples in the queue depends on how many samples are classified as unknown by the classifier in the prediction task. The last column in the table represents the total time consumed during a complete training session. It takes on average 6 seconds for OpenSense to finish updating the model with all discovered classes from the training dataset, faster by at least x10.5 from NA and FRCI. We see that the features extraction latencies are similar across all algorithms since they are based on the same DNN architecture, and lower classification and queuing latencies for the naïve approach compare to FRCI and OpenSense.

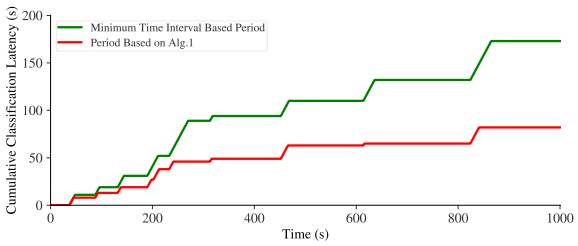
In the learning task, NA requires more time for clustering than FRCI and OpenSense due to the fact that the naïve approach forget the classes previously learned leading to a greater rejection rate of unknown samples than other algorithms, wherein OpenSense has lower clustering latency because it has a higher accuracy in predicting unknown samples. As for the updating the model with novel classes, we see that NA and FRCI require in average around 17 and 16 seconds, respectively, 18x slower than OpenSense. Since other algorithms do not need additional tasks before updating the model, we included the task of finding exemplars in the model updating task for FRCI.

Interestingly, we can update the EVM model regardless of how many samples we have in one epoch making this approach even more suitable for resource-constrained systems. This allow us to process samples periodically as they are coming from the data-stream and choose to update the model when there is enough time given by the dynamic scheduler. We see that in average, it takes less than a second to update the EVM model on a machine with GPU support.

**Execution Time on Embedded Devices:** We only imple-




(a) Sensor data at different periods




(b) Cumulative latency

Fig. 7: Classification latency of the sensor dynamic scheduler

mented OpenSense on the Raspberry Pi 4 Model B since it outperforms the other algorithms in inference, supervised and unsupervised incremental learning, as well as in efficiency. We run the open-world experiment three times for different batch sizes (1,5,10) to simulate periodic data acquisition where each sample in the batch is processed as 1 second-window. In Fig 6, each task execution time is measured by {Seconds, mSeconds, μSeconds} as indicated in the x-axis labels.

For inference tasks, the latency increases as there are more samples in the batch, but the increase is not significant making the choice of the batch size more flexible depending on system requirement. However, it is important to consider that less samples per batch leads to more inference instances before filling the queue and trigger learning. Although, the execution time of the learning tasks is lower when there is only a sample per batch, it does not imply that choosing lower sample rate is better for the system. The average execution time when clustering the rejected samples is dependent on the queue size regardless of the batch size, while updating the model latency depends on the size of the clusters assigned with a new label. The framework is able to inference and learn new classes from a stream of time-series sensor data.

**Classification Latency of Sensor Dynamic Scheduler:** In this experiment, we evaluate the classification latency of the proposed sensor dynamic scheduler (Alg. 1) in simulation, based on the actual dataset of kitchen events collected using the sensor board in Sec. III-A. We compare the proposed scheduler with two different approaches: (i) Fixed Period: using a fixed period of 1s at all time, and (ii) Minimum Class Interval: choosing the minimum running time of each class as the idle sensor period for that class. For the sensor dynamic scheduler, the idling sensor period for each class is found by Alg. 1 for different $CL$ constraints. The largest periods are 25 seconds for the kettle and the vent-fan classes with $CL$ of 13 and 14 seconds, respectively. For microwave, waste disposer, faucet and none, the periods selected are 16, 14, 20 and 15 seconds with $CL$ of 9, 7, 13 and 11 seconds, respectively.

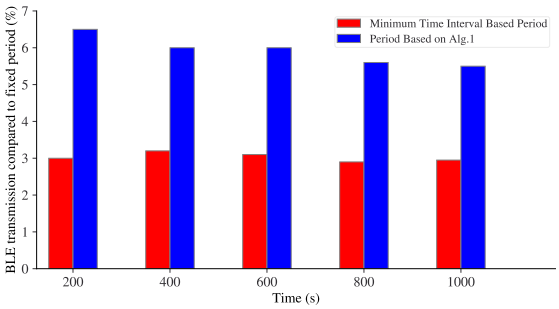



Fig. 8: Percentage of BLE transmissions normalized to the fixed period of 1s

Note that these periods are relatively smaller than the ones selected by the minimum class interval approach.

Simulation results are shown in Fig. 7. The pattern generated by the proposed sensor dynamic scheduler closely matches the fixed 1-sec period pattern with only small delays at detecting the first three events, as shown in Fig. 7(a). The minimum class interval method not only has a higher classification latency, but also misses the none event between the faucet and waste disposer events. These results indicate that the proposed dynamic scheduler significantly outperforms the minimum class interval method.

In Fig. 7(b), the dynamic scheduler with $T_{sp}$ based on Alg. 1 has an overall cumulative latency of around 80 seconds over one thousand seconds of simulation time, while $T_{sp}$ selected based on the minimum class interval method has over 170 seconds of cumulative latency during the same time span. These idle periods can be used to partially update the open-world learning model since EVM allows that.

**Energy Consumption of Sensor Dynamic Scheduler:** By reducing the number of BLE packet transmission while the sensor board is idling, the energy consumption of the sensor board will be significantly reduced. This experiment is based on the simulation results from the previous experiment.

Fig. 8 compares the percentage of BLE transmissions under the proposed dynamic scheduler and the minimum class interval, each normalized to the case when the fixed period of 1s is used. The transmitted BLE packets using Alg. 1 is approximately 6% of the total number of transmissions made by the fixed period approach, and 3% for the minimum class interval method. Considering the latency improvement achieved by Alg. 1 as discussed previously, we conclude that only 3% more of polling requests is an acceptable trade-off.

Subsequently, a dynamic scheduler based on Alg. 1 improves the energy performance of the overall sensing framework while maintain the classification latency within the application requirement.

**The Model Updater Scheduler Performance:** This experiment is based on the previous one but with the assumption that we have 200 samples of an unknown class that has been discovered by the clustering algorithm. To evaluate the model updater scheduler, we choose $T_{sp}$ based on the minimum time interval instead of using Alg. 1 to satisfy the least time to

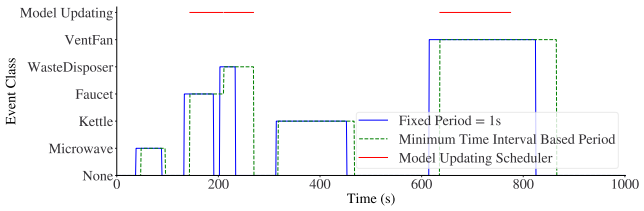



Fig. 9: The model updater is triggered when it meets the conditions in Alg.2

update the model with 1 sample (31s). In Fig 9, we see that the model updater is triggered 3 times to adapt the 200 samples into the model, during the faucet, waste disposal and air vent events with minimum interval periods of 67s, 59s and 46s, respectively. We can see that the model is successfully updated with all the samples even before the last event finishes.

## V. Conclusion

This paper presents the OpenSense framework that has the capability to discover and learn new classes incrementally from a stream of time-series sensing data in both supervised and unsupervised settings. The EVM model adopted in our framework is shown to be effective in rejecting unknown samples correctly while being computationally efficient during incremental learning compared to other algorithms. Our proposed sDNN architecture not only outperforms the state-of-an-art, but also requires fewer epochs to converge during training the network. In addition, our sensor dynamic scheduler finds the longest idle time that does not compromise the required event classification latency, and our model update scheduler enables partially updating the model during such idle time. With these features, our framework can be successfully deployed on a resource-constrained edge device. In the future, we can further consider other resources on the edge device, such as heterogeneous accelerators and GPUs, while minimizing the overhead on the memory. Taking into account the intermittent availability of batteryless energy-harvesting devices would also be an interesting topic. These open a broad range of research directions to improve the practicality of real-time open-world learning on edge devices.

## Acknowledgment

This work was sponsored by the National Science Foundation (NSF) grant 1943265, the National Institute of Justice (NIJ) grant 2019-NE-BX-0006, and the National Institute of Food and Agriculture (NIFA) grant 2020-51181-32198.

## References

[1] G. Laput, Y. Zhang, and C. Harrison, "Synthetic sensors: Towards general-purpose sensing," in *ACM CHI Conference on Human Factors in Computing Systems (CHI)*, 2017.

[2] S. Yao *et al.*, "DeepSense: A unified deep learning framework for time-series mobile sensing data processing," in *International Conference on World Wide Web (WWW)*, 2017.

[3] S.-A. Rebuffi, A. Kolesnikov, G. Sperl, and C. H. Lampert, "iCaRL: Incremental classifier and representation learning," *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[4] M. Jafarzadeh, A. R. Dhamija, S. Cruz, C. Li, T. Ahmad, and T. E. Boult, "Open-world learning without labels," *ArXiv*, vol. abs/2011.12906, 2020.

[5] M. McCloskey and N. J. Cohen, "Catastrophic interference in connectionist networks: The sequential learning problem," in *Psychology of Learning and Motivation*, 1989, vol. 24, pp. 109–165.

[6] T. Xiao, J. Zhang, K. Yang, Y. Peng, and Z. Zhang, "Error-driven incremental learning in deep convolutional neural network for large-scale image classification," in *ACM Multimedia*, 2014.

[7] A. A. Rusu, N. C. Rabinowitz, G. Desjardins, H. Soyer, J. Kirkpatrick, K. Kavukcuoglu, R. Pascanu, and R. Hadsell, "Progressive neural networks," 2016. [Online]. Available: https://arxiv.org/abs/1606.04671

[8] C. Zhang, N. Song, G. Lin, Y. Zheng, P. Pan, and Y. Xu, "Few-shot incremental learning with continually evolved classifiers," in *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2021.

[9] A. Bendale and T. E. Boult, "Towards open set deep networks," *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[10] M. Hassen and P. K. Chan, "Learning a neural-network-based representation for open set recognition," *ArXiv*, vol. abs/1802.04365, 2020.

[11] E. M. Rudd, L. P. Jain, W. J. Scheirer, and T. E. Boult, "The extreme value machine," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 40, no. 3, pp. 762–768, 2018.

[12] S. Kotz and S. Nadarajah, *Extreme value distributions: theory and applications*. World Scientific, 2000.

[13] A. R. Dhamija, T. Ahmad, J. Schwan, M. Jafarzadeh, C. Li, and T. E. Boult, "Self-supervised features improve open-world learning," 2021. [Online]. Available: https://arxiv.org/abs/2102.07848

[14] K. J. Joseph, S. Khan, F. Khan, and V. N. Balasubramanian, "Towards open world object detection," in *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2021.

[15] Y. LeCun, S. Chopra, R. Hadsell, F. J. Huang *et al.*, "A tutorial on energy-based learning," in *Predicting structured data*. MIT Press, 2006.

[16] N. Klingensmith *et al.*, "Hot, cold and in between: Enabling fine-grained environmental control in homes for efficiency and comfort," in *ACM International Conference on Future Energy Systems (e-Energy)*, 2014.

[17] S. Kuznetsov and E. Paulos, "Upstream: Motivating water conservation with low-cost water flow sensing and persuasive displays," in *ACM CHI Conference on Human Factors in Computing Systems (CHI)*, 2010.

[18] J. Scott *et al.*, "PreHeat: Controlling home heating using occupancy prediction," in *ACM International Conference on Ubiquitous Computing (UbiComp)*, 2011.

[19] M. Karimi, H. Choi, Y. Wang, Y. Xiang, and H. Kim, "Real-Time Task Scheduling on Intermittently Powered Batteryless Devices," *IEEE Internet of Things Journal*, vol. 8, no. 17, pp. 13 328–13 342, 2021.

[20] M. Karimi and H. Kim, "Energy Scheduling for Task Execution on Intermittently-Powered Devices," *ACM SIGBED Review*, vol. 17, no. 1, pp. 36–41, 2020.

[21] J. Ward *et al.*, "Activity recognition of assembly tasks using body-worn microphones and accelerometers," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 28, no. 10, pp. 1553–1567, 2006.

[22] A. Akl, C. Feng, and S. Valaee, "A novel accelerometer-based gesture recognition system," *IEEE Transactions on Signal Processing*, vol. 59, no. 12, pp. 6197–6205, 2011.

[23] M. Gadaleta and M. Rossi, "Idnet: Smartphone-based gait recognition with convolutional neural networks," *ArXiv*, vol. abs/1606.03238, 2018.

[24] A. Zahin, L. T. Tan, and R. Q. Hu, "Sensor-based human activity recognition for smart healthcare: A semi-supervised machine learning," *Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering*, 2019.

[25] S. Sarfraz, V. Sharma, and R. Stiefelhagen, "Efficient parameter-free clustering using first neighbor relations," in *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019.

[26] P. Slavık, "A tight analysis of the greedy algorithm for set cover," *Journal of Algorithms*, vol. 25, no. 2, pp. 237–254, 1997.

[27] A. Stisen *et al.*, "Smart devices are different: Assessing and mitigating mobile sensing heterogeneities for activity recognition," in *ACM Conference on Embedded Networked Sensor Systems (SenSys)*, 2015.

[28] A. Reiss and D. Stricker, "Introducing a new benchmarked dataset for activity monitoring," in *International Symposium on Wearable Computers*, 2012.

[29] B. Baldwin *et al.*, "Description of the upenn camp system as used for coreference," in *Message Understanding Conference (MUC)*, 1998.