CIM: A Novel Clustering-based Energy-Efficient Data Imputation Method for Human Activity Recognition

DINA HUSSEIN and GANAPATI BHAT, Washington State University, USA

Human activity recognition (HAR) is an important component in a number of health applications, including rehabilitation, Parkinson's disease, daily activity monitoring, and fitness monitoring. State-of-the-art HAR approaches use multiple sensors on the body to accurately identify activities at runtime. These approaches typically assume that data from all sensors are available for runtime activity recognition. However, data from one or more sensors may be unavailable due to malfunction, energy constraints, or communication challenges between the sensors. Missing data can lead to significant degradation in the accuracy, thus affecting quality of service to users. A common approach for handling missing data is to train classifiers or sensor data recovery algorithms for each combination of missing sensors. However, this results in significant memory and energy overhead on resource-constrained wearable devices. In strong contrast to prior approaches, this paper presents a clustering-based approach (CIM) to impute missing data at runtime. We first define a set of possible clusters and representative data patterns for each sensor in HAR. Then, we create and store a mapping between clusters across sensors. At runtime, when data from a sensor are missing, we utilize the stored mapping table to obtain most likely cluster for the missing sensor. The representative window for the identified cluster is then used as imputation to perform activity classification. We also provide a method to obtain imputation-aware activity prediction sets to handle uncertainty in data when using imputation. Experiments on three HAR datasets show that CIM achieves accuracy within 10% of a baseline without missing data for one missing sensor when providing single activity labels. The accuracy gap drops to less than 1% with imputation-aware classification. Measurements on a low-power processor show that CIM achieves close to 100% energy savings compared to state-of-the-art generative approaches.

CCS Concepts: • **Computer systems organization** → **Embedded systems**; • **Computing methodologies** → *Machine learning*; • **Human-centered computing** → *Mobile devices;*

Additional Key Words and Phrases: Human activity recognition, wearable electronics, missing data detection, data imputation, clustering, health monitoring

ACM Reference format:
Dina Hussein and Ganapati Bhat. 2023. CIM: A Novel Clustering-based Energy-Efficient Data Imputation Method for Human Activity Recognition. *ACM Trans. Embedd. Comput. Syst.* 22, 5s, Article 116 (September 2023), 26 pages.
https://doi.org/10.1145/3609111

This article appears as part of the ESWEEK-TECS special issue and was presented in the International Conference on Compilers, Architectures, and Synthesis for Embedded Systems (CASES), 2023.
This work was supported in part by NSF CAREER award CNS-2238257.
Authors' address: D. Hussein and G. Bhat, Washington State University, P.O. Box 642752, Pullman, Washington, USA, 99164-2752; emails: {dina.hussein, ganapati.bhat}@wsu.edu.

1539-9087/2023/09-ART116 $15.00
https://doi.org/10.1145/3609111

1 INTRODUCTION

Wearable devices are transforming several applications, including health monitoring, rehabilitation, and fitness [8, 13, 24, 27, 31, 38]. Human activity recognition (HAR) forms an important component of wearable health applications as it is crucial to identify what a person is doing before performing a more detailed analysis. For example, in Parkinson's disease monitoring, knowing activity patterns of the patient is critical to prescribing personalized therapy [8, 27, 35]. Similarly, in fitness monitoring, duration and intensity of various activities aid in determining appropriate fitness schedules [43, 45]. In addition to these applications, HAR is also useful in gait analysis and smart homes [16, 23].

Recent HAR algorithms leverage data from multiple sensors mounted on the body to either improve activity classification accuracy or complexity of activities [3, 36, 40]. For instance, the Shoaib et al. [40] dataset uses five accelerometers to perform HAR. Multiple sensors offer complementary information when placed at appropriate locations. For example, the w-HAR dataset [3] uses stretch sensor on the knees and accelerometer on the ankles. Stretch sensor provides movement of knees as users perform activities while the accelerometer provides acceleration along the three directions of motion. Combining the two sensors allows us to achieve accuracy of 98% while accuracy with individual sensors are 90% and 89%, respectively. Similarly, authors of the Shoaib et al. [40] show that multiple sensors are required for activities such as walking upstairs and downstairs. In general, several recent studies have performed HAR with multiple sensors and concluded that multiple sensors offer benefits in accuracy and robustness [1, 5, 30]. One of the key assumptions made by multi-sensor HAR approaches is that data from all sensors are available at runtime for activity classification [14]. However, sensor data may be unavailable due to energy constraints, sensor malfunction, or user error [14, 21, 22, 25]. For example, batteries on one or more sensors may be exhausted due to limited battery capacity in wearable devices. Users may also forget to place sensors at the appropriate location for accurate HAR [17]. Missing data from sensors can significantly degrade the classification accuracy [18]. Indeed, our experiments show that classification accuracy can degrade by more than 30% when two or more sensors are missing, as shown in Figure 1. Therefore, there is a strong need to develop approaches that are able to recover accuracy in HAR applications in presence of missing data.

Generative networks have been recently used to impute missing data in wearable applications [15, 26, 41, 44]. These approaches aim to use generative adversarial networks (GANs) to impute missing data. GANs first learn the joint distribution of sensor observations during training. Then, at runtime the generator is used to obtain imputation conditioned upon observed data [44]. While GANs have shown good accuracy in imputing data, they incur high memory and execution time overhead, which is not feasible for low-power wearable devices. Training sensor-suite specific models is also not feasible since we need to store a model for each possible missing data scenario. That is, in a system with M sensors, we need up to $2^M - 2$ models to handle all potential missing data scenarios. Storing multiple models and switching between them at runtime also leads to memory and execution time overhead. Training classifiers with missing data is also challenging because it requires prior knowledge of missing data patterns for all scenarios, which may not be feasible for all datasets or users. Finally, prior imputation approaches for HAR do not include any detection mechanism for missing data [15, 44]. This is an important component since accurate detection of missing sensors is the first step towards imputation. Therefore, there is a strong need for energy-efficient and low overhead methods for data imputation.

This paper presents an energy-efficient clustering-based imputation (CIM) for missing data detection and imputation in wearable HAR applications. Our approach is guided by two key insights as follows. First, for a given set of activities, each sensor has a repetitive pattern with potential

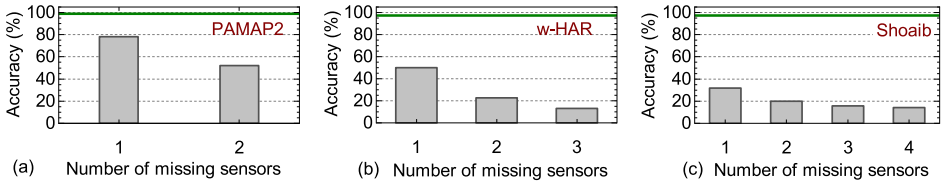

Fig. 1. Comparison of accuracy with missing sensors for (a) PAMAP2, (b) w-HAR and (c) Shoaib et al. datasets. We assume that missing data are observed as Gaussian noise with zero mean and unit variance. Baseline accuracy with no missing data is shown with a green reference line.

variations across different instances of the same activity. For instance, the set of sensor data patterns for sitting is different and separable from walking. Using this insight, we can learn a set of clusters for each sensor and store a single pattern that is representative of all activity windows in a cluster. Secondly, we can learn the inter-relationship and mapping between clusters of all sensors through offline analytics. By learning the mapping between sensors, we are able to predict the cluster information of missing sensors through clusters of available sensors. The representative window for the predicted cluster is used at runtime to efficiently impute missing data while maintaining accuracy. We also propose to utilize sensor data clusters to detect missing data at runtime. Specifically, for each cluster of a sensor, we define a validity region for real data. If the data observed at runtime falls outside the valid region, we flag it as missing. Overall, the proposed CIM approach enables energy-efficient and low-overhead imputation at runtime.

One of the key challenges with data imputation is that uncertainty in classification results increases with the number of missing sensors. As such, providing a single activity label may lead to lower classification accuracy. To overcome this limitation, we augment HAR training data with examples that include representative sensor data for each cluster so that the classifier provides high accuracy with imputed data. In addition to augmentation, we provide a set of possible activity labels when multiple sensors are missing to account for uncertainty in imputation. This is similar to the concept of conformal prediction where classifiers provide the set of most likely labels [39, 42].

We validate the proposed CIM approach on three publicly available HAR datasets. For each dataset, we simulate all possible missing data scenarios and use CIM for imputation. Our results show that the accuracy achieved by CIM for single-label prediction is within 10% of accuracy with no missing data for one missing sensor. At the same time, our imputation-aware prediction achieves accuracy within 1% of the baseline accuracy without missing data. We also implement a generative approach using generative adversarial imputation networks (GAIN) since they have shown high accuracy in prior work [10, 44]. Accuracy analysis with GAIN shows that they provide accuracy that is comparable or lower than CIM with significantly higher overhead. For instance, CIM incurs up to 219 KB of memory overhead, while GANs consume up to 102 MB of memory. Finally, we implement CIM on the Odroid-XU3 board [12] to measure execution time and energy overhead. Our real-world measurements on the Odroid-XU3 board show that CIM takes up to 1 ms per imputation for the largest dataset with about 0.5 mJ energy consumption.

In summary, this paper makes the following contributions:

- A low overhead and energy-efficient approach for sensor data imputation at runtime to handle multiple missing data scenarios. Specifically, CIM handles all possible missing data scenarios with a single set of clusters and mapping function.
- Novel algorithm to detect missing data using pre-defined clusters for sensor data.
- Imputation-aware HAR to account for uncertainty in classification with imputed data.
- Extensive experimental evaluation with three publicly available datasets to evaluate the efficacy of CIM.

The rest of the paper is organized as follows: Section 2 reviews the related work, while Section 3 introduces preliminaries for HAR and sets up the data imputation problem. Section 4 introduces the proposed CIM approach for missing data detection and imputation at runtime. We provide the experimental results with three datasets in Section 5 and finally conclude the paper with some future research directions in Section 6.

2 RELATED WORK

HAR forms an important component of several health applications such as rehabilitation, movement disorders, and fitness monitoring [8, 24, 27]. Recent HAR approaches have multiple sensors mounted on the body to enable recognition of complex activities and improve HAR accuracy [36, 40]. Addition of multiple sensors for HAR makes the system susceptible to missing data from one or more sensors due to energy constraints, user error, malfunction, or communication challenges [14, 21, 22, 25]. Therefore, there is a need to develop approaches that handle missing data at runtime in an energy-efficient manner.

Several statistical and machine learning (ML) approaches have been proposed to handle missing data in HAR and other time series applications [15, 33, 34, 37]. Popular statistical methods include mean, median, and regression-based imputation [6, 10, 34, 37]. While these methods are useful, they are able to handle only isolated instances of missing data where real sensor data are available to perform statistical analysis. For instance, mean imputation methods substitute the missing data with mean of data around the missing time instance. Due to this limitation, statistical methods are generally not suitable for long sequences of missing data.

Recent approaches have used machine and deep learning approaches to recover longer sequences of missing data [28, 33, 37, 44]. The method in [33] uses the k-nearest neighbor (KNN) algorithm to impute missing data. However, KNN algorithms must store entire training data on the device to obtain imputation, thus making them unsuitable for low-power wearable devices. Generative methods are also becoming popular for data imputation at runtime. The CNNAE [41] approach aims to employ single variational autoencoder architecture to recover missing data in brain recording data from multiple users. The key limitation of variational autoencoders is their large size and resulting overhead. For instance, our initial experiments show that variational autoencoders require memory in the order of hundreds of megabytes, which is not feasible for implementation on wearable devices. GAIN [44] is another generative training approach for data imputation. The key idea behind GAIN is that we can learn relationships between different sensors during training. The goal of the generator in GAIN is to accurately reproduce data while the discriminator aims to identify if each data point is missing or not. During runtime, GAIN defines a mask for missing sensor values and generates data for both available and missing sensors. The mask is then used to extract imputation for missing sensors. While the GAIN is useful, it suffers from accuracy drop when more than one sensor is missing. Overall, prior approaches for imputation either suffer from low accuracy or high overhead, making them unsuitable for our use case.

In strong contrast to prior approaches, this paper proposes CIM, a clustering-based imputation approach for HAR. CIM obtains clusters for sensor data along with representative windows for each cluster to enable energy-efficient missing data detection and imputation at runtime. Our experiments with three HAR datasets show that CIM achieves, on average, 55% higher accuracy with two missing sensors while having close to 100% lower energy when compared to GAIN.

3 BACKGROUND AND PROBLEM SETUP

3.1 HAR Background

HAR classifier design typically involves four major steps. The first step involves data collection from a suite of sensors mounted on a user's body while they perform activities of interest, as shown

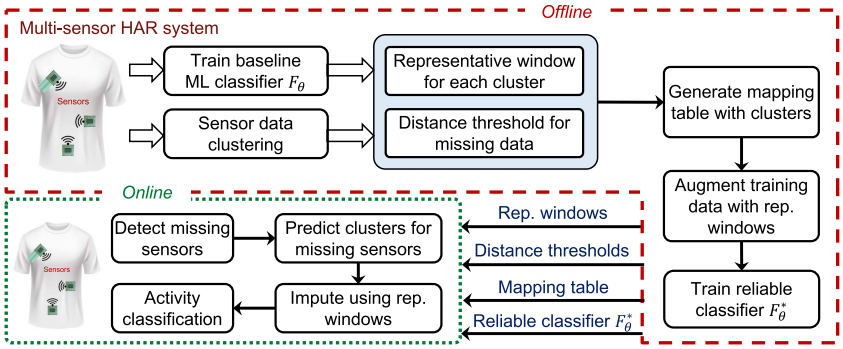

Fig. 2. Overview of offline and online steps in the proposed CIM approach.

in Figure 2. Commonly used sensors for HAR include accelerometers, gyroscopes, bend, heart rate, and stretch sensors. The data from all sensors are aggregated at a host device or one of the wearable nodes to perform activity classification. Aggregation of data at a host device is required since each sensor might be located on different parts of the body with no connection between them. The data goes through the following steps once it is aggregated at a host or wearable device.

Segmentation: Streaming sensor data must be divided into distinct activity segments to enable seamless classification at runtime. Prior HAR approaches use either fixed or variable-length segments [3, 40]. Fixed-length windows in HAR ensure that the input length to k-means is constant. Variable length windows offer fine-grained information on activities, however, classification and clustering algorithms expect fixed length time series data. As such, the variable length windows are passed through a filter to obtain equal length windows. For example, walking activity has an average length of about 100 samples in the w-HAR dataset, while sitting activities have 183 samples. These are passed through a resampling filter to obtain windows with 64 samples for both walking and sitting activities. Ensuring fixed-length windows is critical because classification algorithms expect inputs to be of fixed length for learning. In this work, we re-use fixed-length (PAMAP2 and Shoaib et al.) or resampled windows (w-HAR) provided by respective datasets to perform k-means clustering and classification. We refer to each activity segment as a window.

Feature Generation and Classification: After segmentation, the sensor data are passed through a feature generation block to obtain a set of features for use in ML models. The feature set and activity labels are used to train supervised learning models to enable classification at runtime. Common classifiers for HAR include multilayer perceptrons (MLP), random forest, and convolutional neural networks (CNN) [2, 3, 24]. In this paper, we use 1-dimensional CNNs for activity classification while noting that any supervised learning classifier can be used with CIM.

3.2 Problem Setup

Let us consider a system with M sensors on the user's body for HAR, as shown in Figure 2. Without loss of generality, we can assume that each activity window consists of T samples from each sensor. In some scenarios, we may have unequal samples from each sensor due to differences in sampling rates. However, we assume T to be same across all sensors for ease of notation and exposition. Once sensor data are aggregated, we can represent the data for each window t using a $M \times T$ matrix $\mathcal{X}_t = \{X_1, X_2, \ldots, X_M\} \in \mathbb{R}^{M \times T}$, where X_i $(1 \leq i \leq M)$ is the data vector from the i^{th} sensor. Using the above notation, we can represent a training dataset $\mathcal{D}_{train}$ with multiple windows as $\mathbb{X} = \{\mathcal{X}_1, \mathcal{X}_2, \ldots, \mathcal{X}_N\}$, where N is the number of training windows. The data matrix is passed through a feature generation function to obtain labeled feature and activity a^* pairs. The feature

and activity pairs are used in a supervised learning algorithm to train an activity classifier F_θ, where θ denotes the classifier parameters. At runtime, we use the trained classifier F_θ to predict current activities as $\hat{a}_t$.

Now consider that k ($1 \leq k \leq M-1$) of the M sensors are unavailable at runtime for an activity window t. The number of missing sensors goes from 1 to $M-1$ since we can ignore the case of no sensors missing and all sensors missing. Since the HAR classifier expects data from all sensors for classification, the missing sensor values will get substituted with zeros, a constant value, or noise. We denote the data matrix with missing sensor data by $\mathcal{X}_m$. The missing data causes a mismatch in the feature values, leading to inaccurate classification by the HAR classifier F_θ. Recovering missing data at runtime to obtain accurate classification is a challenging problem because there are $2^M - 2$ missing data scenarios. Two scenarios corresponding to $k = 0$ and $k = M$ are excluded since the case of no missing data does not need imputation and it is not possible to impute data if all sensors are missing. To overcome the drop in accuracy due to missing data, our goal is to obtain a missing data detection function f_d^* and recovery function f_r^* to efficiently detect and recover missing data. We describe key requirements and challenges for each of these functions below.

Missing data detection function f_d^*: The goal of f_d^* is to take observed sensor data in each window $\mathcal{X}_t$ and specify if any of the sensors are missing. Specifically, the output of f_d^* is a $M \times 1$ vector with each element denoting whether the corresponding sensor is missing. We set an element to one if the sensor is missing, otherwise it is set to zero. The missing data detection function f_d^* is challenging to design because it must balance accuracy of detection while avoiding false positives. Specifically, f_d^* must avoid flagging actual observed data as missing data while maintaining high accuracy for cases when data are missing. False positives for missing data detection may hamper accuracy since the observed data gets substituted with imputed data. Therefore, our goal is to design a detection function that accurately identifies missing sensors while avoiding false positives.

Missing data recovery function f_r^*: The objective of f_r^* is to leverage data from available sensors to recover data that are unavailable. Specifically, $f_r^* : \mathbb{R}^{(M-k)\times T} \rightarrow \mathbb{R}^{k\times T}$ maps the data from $M-k$ available sensors to data for k missing sensors. The recovery function f_r^* must ensure that the imputed data recovers accuracy of classification with minimal overhead. Design of the recovery function is challenging since it must work for all missing data scenarios since storing multiple functions leads to additional memory overhead for wearable devices.

4 PROPOSED CLUSTERING-BASED IMPUTATION APPROACH

4.1 High-level Overview of CIM

We provide a high-level overview of CIM in Figure 2. CIM starts by obtaining clusters for each sensor used in HAR and trains a baseline activity classifier F_θ. Next, we obtain a representative window for each cluster and distance thresholds for missing data detection. The cluster information and representative windows are then used to obtain an *offline* mapping table that learns the inter-relationship between sensors. The mapping table is used at runtime to predict clusters for missing sensors. Since our goal is to obtain high classification accuracy at runtime, we augment the training data with representative windows and re-train the activity classifier to obtain F_θ^*. At the end of the offline design stage CIM provides the set of representative windows, cluster centroids, distance thresholds, mapping table, and reliable classifier F_θ^* as outputs.

At runtime, we obtain sensor data from user activities with the potential for missing sensors. Therefore, as a first step, CIM performs missing data detection to obtain the set of missing sensors. Then, it uses the mapping table to predict clusters for missing sensors and imputes missing data with representative windows. Finally, CIM performs activity classification with imputed data and provides it to the user. We provide details of each step in the following sections.

4.2 Sensor Data Clustering

The first step in the proposed CIM approach is partitioning data from each sensor into clusters. The goal of clustering is to ensure that distinct activity patterns are separated into their own clusters. The key insight behind clustering is that human activities are generally repeatable in nature, with variations across users and time. For instance, the general walking pattern for a majority of healthy users is similar, with differences due to a person's height, limb length, or weight. The repeatability of activities, in turn, leads to repeatability of sensor data patterns. Secondly, different classes of activities have distinct sensor data patterns. For example, accelerometer sensor data for sitting and walking are distinct and easily separable. Sitting shows a relatively constant acceleration, while walking has variable acceleration as the user moves. Following this insight, we obtain clusters of sensor data where each cluster contains sensor data patterns that are close to each other. The data from clustering are used at runtime to both detect and impute missing sensors.

We employ the commonly used k-means clustering algorithm [19] to obtain distinct clusters of sensor data. The input to the k-means algorithm is data from each sensor $\mathbb{X}^j (1 \leq j \leq M)$ and the number of desired clusters c_j for the j^{th} sensor. The output of clustering for each sensor is a set of centroids for c_j clusters and mapping of each window to its respective cluster. The centroids are used at runtime to infer the clusters for new, unseen data. At the end of clustering, we obtain a set of clusters $C = \{c_1, c_2, \ldots, c_M\}$ for all M sensors.

The number of clusters must be found through a design space exploration since k-means is an unsupervised learning approach. The number of clusters in CIM is driven by the following factors:

- Similarity of sensor data patterns in the cluster. We would like the data patterns to be similar in each cluster so that the representative window could be used for imputation.
- The clusters should not be sparse with a small number of windows. Small number of windows in a cluster could indicate too many clusters for a dataset. Therefore, we aim to avoid sparse clusters during the k-means clustering step.
- Label purity, i.e., number of unique activity labels in a cluster is another consideration when choosing the number of clusters for a dataset. Label purity is helpful since different activities may have distinct patterns and must be in different clusters. At the same time, label purity is not a primary consideration since multiple activities may have similar patterns. For instance, stand and lie down have similar patterns for stretch sensor in the w-HAR dataset, leading to windows of both activities being in the same cluster.

Overall, we use similarity of sensor data patterns in a cluster as the primary factor in deciding the number of clusters, followed by sparsity of clusters and label purity.

Representative window for each cluster: The k-means clustering divides sensor data into c_j clusters for the j^{th} sensor. Each cluster consists of multiple windows from the training data. As such, it is infeasible to store every window from a cluster for data imputation at runtime. Therefore, we propose to identify a single window that is representative of all windows in the cluster. The representative window must fulfill two goals: (1) it must provide high accuracy when it is used to impute missing data, (2) it must be representative of all data for a sensor in the cluster. That is, it must be an 'average' case of data in the cluster.

Algorithm 1 shows the procedure of identifying the representative window for each cluster. Inputs to the algorithm include data for all windows in the cluster, number of windows, and base activity classifier F_θ. Given these inputs, the algorithm first calculates the mean pairwise Euclidean distance from each window to all other windows and stores it in an array $\mathcal{D}$. The pairwise distance from a window w to others is given by the vector $\boldsymbol{d}_w$ as:

$$\boldsymbol{d}_w = \{||X_{j,w} - X_{j,k}|| \Big| k \in (1, 2, \ldots, W) \;\&\; k \neq w\},\ 1 \leq w \leq W \tag{1}$$

ALGORITHM 1: Representative window for each cluster

1 **Input:** Windows in current cluster of a sensor j (c_j), Number of windows W, Activity classifier F_θ
2 Initialize a $W \times 1$ empty array to store pairwise distances $\mathcal{D}$
3 Initialize a $W \times 1$ empty array to accuracies with representative windows $\mathcal{A}$
4 **for** $w = 1, 2, \ldots, W$ **do**
5 $\quad \boldsymbol{d}_w \leftarrow$ Pairwise from window w to all other windows in c_j
6 $\quad \mathcal{D}_w \leftarrow \text{mean}(\boldsymbol{d}_w)$
7 **end**
8 $\mathcal{D}_{sorted} \leftarrow$ Sort $\mathcal{D}$ in ascending order
9 **for** k = *sorted order in* $\mathcal{D}_{sorted}$ **do**
10 $\quad$ Replace sensor j data with candidate representative window k
11 $\quad \mathcal{A}_k \leftarrow$ Classification accuracy with candidate representative window k using F_θ
12 **end**
13 Representative window for $c_j \leftarrow \arg\max(\mathcal{A}_k)$
14 **Return:** Window with the highest accuracy as the representative window for cluster c_j

where $X_{j,w}$ are the data from sensor j in window w. k is a dummy variable that iterates over all possible windows in the cluster except for w. A large pairwise distance means that the window is likely to be an outlier, whereas a small distance means it is close to all other windows. For instance, if we consider that all windows in a cluster form a T-dimensional ball, the windows near surface of the ball are further away from other windows compared to windows near center of the ball. As such, our goal is to find windows that are close to center of a cluster. After obtaining mean distances, we sort them from smallest to largest for selection of the representative window that offers highest classification accuracy. Specifically, our goal is to maintain high classification accuracy when a candidate representative window is used instead of observed data for feature generation and classification. This emulates a real-world situation where missing data are substituted with the corresponding representative window. To this end, we obtain the classification accuracy for activities in the cluster using F_θ while substituting each candidate representative window for the target sensor. That is, if the current candidate representative window belongs to sensor j, we substitute the j^{th} sensor data in $\mathcal{X}_t$. We note that the baseline classifier training does not use representative windows.

The classification accuracies in the cluster are given by the vector $\mathcal{A}$ as:

$$\mathcal{A} = \text{Accuracy}\Big(F_\theta([X_{1,k}, X_{2,k}, \ldots, \tilde{X}_{i,k}, X_{i+1,k}, \ldots, X_{M,k}])\Big)\ 1 \leq k \leq W \tag{2}$$

where we use the classifier function F_θ to perform classification on imputed data. Specifically, sensor j data are substituted with the candidate window $\tilde{X}_{j,k}$. The accuracy vector contains classification accuracy for all W windows in a given cluster. We note that the accuracy analysis can be optimized by considering a small number of windows with the lowest mean distances. This is achieved by modifying line 9 in the algorithm to consider windows that have the smallest distances. The algorithm description goes over all windows in sorted order to maintain generality. Finally, the window that offers the highest classification accuracy is chosen as the representative window:

$$X_r^{c_j} = \arg\max \mathcal{A} \tag{3}$$

where $X_r^{c_j}$ is the chosen representative window for cluster c_j. $X_r^{c_j}$ is returned by the algorithm and appended to list of representative windows of a sensor. Set of representative windows are used at runtime for data imputation, as we describe in the following sections.

Cluster centroids as representative windows: We note that cluster centroids can also be used as representative windows. We do not use them as representative windows since our goal is to substitute missing data with real sensor data from observations during training. Moreover, centroids may not always achieve higher accuracy when compared to using real, observed data as representative windows. Consequently, we do not use cluster centroids as representative windows.

4.3 Runtime Missing Data Detection

Missing data detection is an important component of enabling reliable HAR algorithms. In general, we can divide missing data patterns into two classes depending on the data unavailability. We provide brief descriptions of the two classes and our problem setting below.

Random missing data: Random missing data occurs when the device encounters isolated missing samples. The missing samples are not clustered around a time instance. Prior work has proposed methods to impute isolated missing data using statistical methods, such as mean or median [6, 33]. Random missing data are easier to handle since we have observed data around the missing samples that can be used as a reference point for imputation.

Block missing data: Block missing data occurs when sensor data are unavailable for longer intervals of time, such as multiple activity windows. Block missing data typically occurs due to energy limitations, sensor malfunction, or communication channel challenges. It is challenging to recover block missing data since there are no reference observations for the missing sensors. In this paper, our goal is to detect and impute sensor data in the block missing scenario.

One of the key considerations in missing data detection is observed data when a sensor is unavailable. The type of missing data filling is typically device dependent and each wearable device may choose one of the following mechanisms for filling missing data.

Zero-filling: In this case if a sensor is unavailable, corresponding observations for the sensor are filled with zeros. This is a reasonable assumption since sensor data are typically read into processors through memory buffers that may be initialized with zeros.

Constant data or average filling: Another common scenario for observed missing data is that the sensor value remains stuck at the previously observed value. A corollary to filling with previously observed value is average filling where the data are filled with an 'average' case value whenever a sensor is available. In both cases, observed sensor data remains at a constant value.

Random noise filling: The last case we consider is when missing sensor data are filled with random noise from either a Gaussian or uniform distribution. This can be attributed to noise from the power supply or communication channels. In this paper, we primarily consider that random noise is drawn from a Gaussian distribution with zero mean and variable variance.

Next, we describe our approach to missing data detection for each of the above scenarios. The missing data detection algorithm runs once a window of data have been accumulated from the sensors. That is, we run the missing data detection algorithm once a window has been identified for classification. As such, the missing data detection algorithm will be able to handle windows with missing data block at any point in the window. Indeed, our experiments in later sections show that the proposed approach handles both block and random missing scenarios with a single algorithm.

4.3.1 Detecting Zero-filled or Constant Missing Data. First two cases of missing data filling contain constant data: zeros or constant non-zero data. It is typically easier to detect missing data with constant values. Specifically, we propose to calculate the finite difference and variance in the window to determine if the data are constant. If the algorithm detects that data in a window for a

given sensor are constant, it is flagged as missing data. This approach is ideal for zero or average data filling since it is highly unlikely that real, observed data are constant for an entire window.

4.3.2 Detecting Missing Data with Random Noise. It is more challenging to detect missing data filled with random values since we are unable to employ finite differences to check for a constant value. Indeed, random noise may appear to be real-world sensor data on visual inspection. Therefore, we need a principled approach that accurately classifies noise as missing data.

We propose to leverage sensor data clusters to detect random missing data at runtime. During the data clustering step, we analyze the distance of all windows in a cluster to the cluster centroid. That is, for each cluster, we obtain the set of distances from the centroid for each window as:

$$\mathcal{S} = [|y_1 - y^{c_j*}|^2, |y_2 - y^{c_j*}|^2, \ldots, |y_j - y^{c_j*}|^2, \ldots, |y_W - y^{c_j*}|^2] \tag{4}$$

where y_i represents the data for a given sensor in a window i, while y^{c_j*} is the centroid for the cluster c_j. The range of distances represents valid region of windows for a given cluster. Any window that falls outside the valid region is an outlier for the cluster. Moreover, given a set of centroids obtained from k-means classification, inferred cluster label for a new window is the centroid closest to it. Consequently, we can store the validity region bounds for each cluster and classify any window that falls outside of the valid region as missing data. Our approach is based on the key insight that offline clustering of training data allows us to learn valid regions of real-world sensor observations and classify windows outside the region as missing data.

The missing data detection algorithm can be tuned by varying the threshold for distance that is classified as missing data. Specifically, the missing data threshold is a hyperparameter that can be tuned to balance false positives and accuracy. The algorithm is more sensitive if we choose 50^{th} percentile of the distance as the threshold, while it has higher flexibility if we choose 95^{th} percentile as the threshold. The threshold must be chosen through a design space exploration with detection and classification accuracies as objectives. In our experiments, we vary the threshold from 95^{th} percentile of the distances in Equation (4) to 1.05 times the maximum distance. The design space exploration with these threshold values shows that w-HAR and Shoaib et al. achieve higher accuracy with 95^{th} percentile while PAMAP2 has higher accuracy with 1.05 times the maximum distance. Therefore, we choose missing detection threshold of 95^{th} percentile for w-HAR and Shoaib et al. and 1.05 for PAMAP2. Choosing 95^{th} percentile marks 5% of windows missing when actual data are available. This is acceptable because k-means algorithms are heuristic in nature and windows at the boundary of a cluster are generally outliers for the respective cluster.

Missing data detection algorithm summary: Algorithm 2 summarizes the runtime missing data detection procedure used in this paper. The inputs to the algorithm are data from the sensor that is potentially filled with zeros, constant value, or noise. It also includes a set of cluster centroids for the sensor and distance thresholds for each cluster. Using these inputs, we first check if the data in the input window are constant by taking finite difference of the elements. The finite difference of the window is given by:

$$\Delta(X_j) = X_j[k+1] - X_j[k] \; 1 \leq k \leq T-1 \tag{5}$$

where $\Delta(X_j)$ is a $(T-1)\times 1$ vector containing the finite difference of samples in X_j. If all elements in $\Delta(X_j)$ are zero, it means that the sensor data are constant. Therefore, we flag it as missing. Next, if the data are not constant, the algorithm uses the set of centroids to assign a cluster to the window and calculate distance from the closest centroid. If the distance d is greater than the pre-defined cluster threshold c_j^{thresh} from Equation (4), we flag the data as missing. Specifically, we obtain a

missing data flat f_m as:

$$f_m = \begin{cases} 1 & \text{if} \quad d \geq c_j^{thresh} \\ 0 & \text{if} \quad d < c_j^{thresh} \end{cases} \tag{6}$$

The intuition behind the algorithm is that the inferred cluster for sensor data is the cluster that is closest to X_j. Therefore, if the distance from the inferred cluster centroid is higher than the threshold, we can classify it as missing data. The algorithm is run for each sensor used for HAR and its output is a vector of zeros and ones indicating if data in each sensor are missing or not.

4.4 Data Imputation with Sensor Clusters

The next step after missing data detection is imputation of sensors that are missing. We propose to use stored representative windows and cluster mapping across sensors to impute missing data. Our approach is based on the following key insights:

- Given a set of sensors and their respective clusters, we can learn the mapping between clusters across sensors. The learned mapping can then be used to predict the cluster of missing sensors at runtime.
- After predicting cluster for missing sensors, we can use representative window of the predicted cluster to impute missing data. Since the representative window is an 'average' case of sensor observations for a cluster, we can use it as the most likely data pattern for missing sensors.
- The imputed data does not have to exactly match real-world observed data as long as it preserves the general data pattern and classification accuracy. Preserving classifier accuracy improves user satisfaction while providing the most likely pattern for missing data allows for a deeper analysis by health experts or classifier developers.

Based on the above insights, we use the following steps to design an imputation algorithm.

4.4.1 Mapping Table Construction. Using the first insight above, we obtain a table that contains the cluster information of each sensor for all windows in the training set. Each row in the table records cluster for the corresponding sensor. Since the data has repetitions of all activities in the training data, initial table of clusters will have repeated rows. Storing all repetitions of rows for runtime usage leads to higher overhead. Therefore, we reduce the table by obtaining all unique

ALGORITHM 2: Missing data detection

```
Input: Current sensor j data observations X_j, Cluster centroids for sensor j, Missing data distance
  thresholds for all clusters in sensor j
Obtain finite difference of data in X_j
if X_j is constant then
    Flag missing data
end
else
    Use centroids to obtain cluster for X_j
    Calculate distance d from X_j to classified cluster centroid
    if d > Distance threshold for current cluster then
        Flag missing data
    end
end
Return: Missing data flag for X_j
```

Table 1. Data Format for Each Row in the Mapping Table

Sensor 1 cluster	Sensor 2 cluster	...	Sensor M cluster	Count

combinations of clusters and recording the number of occurrences of each combination. Each row of the reduced table contains the elements shown in Table 1 to capture the cluster mapping across sensors and count of occurrences for each unique mapping. Intuitively, the mapping table captures the inter-relationship between observations in each sensor. For instance, if data of sensor 2 is generally flat when sensor 1 has a repetitive pattern, respective cluster mappings of the sensors will reflect the relationship in the table. This is similar to capturing the joint probability of sensors without utilizing expensive generative networks.

4.4.2 Runtime Imputation using Mapping Table. At runtime one or more sensors may be missing and we must find the appropriate imputation for each missing sensor. We use the mapping table to predict corresponding clusters for each missing sensor. The algorithm starts by obtaining clusters for sensors that are available. The clusters for available sensors are then used as keys to find corresponding rows in the mapping table obtained in the previous section. The matching row is used to predict the cluster for missing sensors based on observations made during design time. For example, in a system with three sensors, consider that sensor 1 is missing and other sensors map to clusters 1 and 2, respectively. We use (_, 1, 2) as a key to find a potential table row (3, 1, 2). Given this, we predict the cluster for the missing sensor as cluster 3. Finally, the stored representative window for the predicted cluster is used as imputation for the missing sensor.

Resolution for multiple mapping table entries: Table lookup with clusters of available sensors may find more than one matching row when data from multiple sensors are unavailable. In such cases, we must choose one of the matching rows for predicting the clusters of missing sensors. This is a challenging problem because we do not have any information about missing sensor to obtain the imputation in an energy-efficient manner. Since the goal of CIM is to find the most likely data pattern missing sensors, we choose the row with highest count for our prediction.

Choosing the row with highest count is reasonable since the mapping is observed with highest frequency in the training data. If multiple rows have the same count, we use the representative windows from all rows for imputation. Then, we choose the row that provides classification with the highest logit value as the imputation row. This ensures that the row which has higher confidence for classification is chosen as the representative window for missing data recovery. After predicting clusters for missing sensors, we use the respective representative windows as imputation and perform activity classification.

Choosing the row with highest count can result in incorrect clusters for some windows. This can result in one of the following two conditions:

- Wrong cluster is chosen for imputation, however, the classification result is correct. This happens when two clusters are close to each other and imputation with either cluster results in correct classification.
- Wrong cluster is chosen for imputation, which also results in incorrect classification. These instances could possibly be avoided by choosing rows that do not have the highest count.

In our experiments, we observe that there are some instances of the second scenario that result in incorrect classification due to choosing the row with highest count. Resolving these errors would require using more complicated algorithms for the mapping function, such as generative

neural networks. The neural network can take data of available sensors as input and provide the appropriate output cluster as the output. However, this results in higher overhead of running neural networks at runtime. Since our goal is to provide low overhead imputation and classification we use the rows with highest count for imputation. Choosing the row with highest count is also reasonable since the mapping is observed with highest frequency in the training data.

4.5 Training Data Augmentation for Reliable Classification

Data imputation with representative windows provides high accuracy for most activity windows. However, using representative windows instead of actual data may result in misclassifications for a small number of cases. We can improve the accuracy for these cases by training the classifier to recognize representative windows so that it is able to obtain accurate activity classification with missing sensors. To this end, we propose to augment training data by substituting observed data with representative windows. Specifically, we first obtain the cluster for observed data and then replace it with the respective representative window. The augmentation is done for one sensor at a time so that new training examples contain at most one representative window at a time. The HAR classifier is re-trained with the augmented data to improve reliability of classification. We denote the re-trained classifier with F^*_θ. We note that we can also perform augmentation for specific activities if classification accuracy after imputation is lower for certain activities.

4.6 Imputation-Aware Activity Classification

The uncertainty in classification and sensor data increases when we perform imputation for missing data. However, most prior approaches for data recovery in HAR provide a single activity label after imputing data. The single label does not capture uncertainty in the sensor data. Instead, providing a single label assigns the highest confidence to the label even when the model is not certain of the outcome. To overcome this challenge, we propose to provide imputation-aware prediction sets for activities to capture uncertainty in classification with imputed data.

CIM leverages mapping tables in Section 4.4 to obtain imputation-aware classification sets. Let us consider that a lookup in the mapping table with clusters of available sensors yields R matching rows. For each of these rows, we impute missing data with respective representative windows and use the activity classifier to obtain normalized logits. That is, for each row r_i, we obtain logits as:

$$\mathcal{L}_{r_i} = [l_{1,i}, l_{2,i}, \ldots, l_{A,i}] \quad 1 \leq i \leq R \tag{7}$$

where $\mathcal{L}_{r_i}$ is the set of normalized logits for i^{th} row, A is the number of activities, and $l_{k,i}$ ($1 \leq k \leq A$) is the logit value for k^{th} activity. The logit values for each row add to one since we use normalized values. Repeating classification for all matching rows results in a matrix with R rows and A columns. Next, we add logits for each activity across all the rows and normalize it to obtain the overall logit value of each activity as:

$$\mathcal{L} = \left[\sum_{i=1}^{R} l_{1,i}, \sum_{i=1}^{R} l_{2,i}, \ldots, \sum_{i=1}^{R} l_{A,i} \right] \Big/ R \tag{8}$$

Intuitively, each value in Equation (8) represents probability of each activity across the R matching rows. Given these probabilities, we can define a desired confidence level α and use it to obtain a set of predictions. Specifically, we sort the vector of probabilities in Equation (8) and add activities into the prediction set until desired confidence level α is achieved. The imputation-aware activity classification is similar to conformal prediction, which is being widely used to quantify uncertainty in machine learning models [39, 42].

We note that imputation-aware prediction presents a trade-off in terms of the runtime overhead and classification accuracy. In particular, obtaining prediction sets requires multiple rounds of

classification, which increases overhead when compared to obtaining a single classification. We can lower the overhead by using a subset of R rows for classification. Specifically, CIM can perform classification with R rows with highest counts in the mapping table while using the counts as weights. This avoids performing classification with all possible matching rows in the mapping table, thus lowering the overhead. Using counts as weights in imputation-aware classification also gives higher priority to activity labels observed with higher frequency during training. Overall, prediction sets significantly improve the accuracy with slightly higher overhead.

Choosing single vs imputation-aware classification: The choice between single and imputation-aware classification approaches typically depends on user preferences and application. If a user wishes to obtain a single classification albeit with a lower accuracy, they may choose the single activity label. This is useful when we would like to get exact information about a user's activity, such as the time spent in walking. In these scenarios, it is beneficial to obtain a single activity label with the knowledge that some of the classifications may be incorrect in case of multiple missing sensors. Similarly, imputation-aware classification provides a set of possible activities with a high probability of containing the actual activity. The classification sets allow higher accuracy while noting that the true activity is one of the predicted activities. In general, imputation-aware classification is superior since it provides both activities and an indication of uncertainty in classification. As such, we envision that imputation-aware classification will be used as the default option in CIM. Single activity classification is also provided as an option for users, if needed.

4.7 CIM Summary

In summary, CIM first performs the following offline steps to train clusters and classifiers.

(1) Train a baseline classifier with actual sensor data.
(2) Obtain clusters for each sensor used for HAR and mapping between sensors.
(3) Utilize Algorithm 1 to find representative window for each cluster for all sensors.
(4) Calculate missing data threshold for each cluster.
(5) Train reliable classifier with data augmentation.

Using the trained clusters and classifiers, CIM performs following steps at runtime.

(1) Detect the set of missing sensors using Algorithm 2.
(2) Predict clusters for missing sensors using mapping table and impute with representative windows using the approach in Section 4.4.2.
(3) Obtain either a set of activities using imputation-aware classification in Section 4.6 or a single activity label using reliable activity classifier F^*_θ.

Training complexity: The training complexity consists of the k-means clustering, obtaining representative windows for each cluster, calculation of threshold for missing data detection, and mapping across clusters. The first step during training is k-means clustering. In general, k-means clustering is a NP-hard problem to obtain the optimal solution. However, most real-world heuristics are able to converge with significantly lower iterations. In our implementation, we utilize the Lloyd's algorithm for clustering, which has an average case complexity of $O(kNI)$ where N is the number of windows, k is the number of clusters, and I is the number of iterations [11]. We execute k-means with maximum of 300 iterations, however, in practice k-means is able to converge in significantly fewer iterations. Next, CIM training executes Algorithm 1 to find representative windows for each cluster. The key step in representative window selection is the pairwise distance calculation between windows in a cluster. The complexity of this step is given by $O(T * W^2)$ where T is the number of samples in a window and W is the number of windows in a cluster. The missing data detection threshold computations can also be performed along with representative

window selection since it depends on pairwise distance between windows. Finally, the mapping table generation requires a linear search through all training windows in a dataset. This results in a complexity of $O(N)$ for obtaining the mapping table. Combining all steps, the overall complexity for CIM training is in the order of $O(T * W^2)$, since it is the dominant factor.

Runtime complexity: We evaluate the runtime complexity of imputation-aware CIM since it has higher complexity among the two CIM versions. The missing data detection requires obtaining clusters for available sensor data, which in turn requires calculation of distance for each centroid. Considering M sensors, this results in a complexity of $O(M * C_{max})$, where C_{max} is the maximum number of clusters. Next, CIM obtains the matching rows in the mapping table as a function of the sensor data clusters. This requires a linear search through the mapping table, resulting in a complexity of $O(l_M)$ where l_M is length of the mapping table. Finally, CIM obtains classification with each matching row of the mapping table. In the worst case, this results in l_M classifications when all rows of the table are matching. Since each classification takes a constant time, the complexity for classification is $O(l_M)$. Overall, considering all steps, the dominant step is classification, resulting in a complexity of $O(l_M)$ for runtime imputation. We note that these are worst case complexities and the execution times are significantly lower in practice.

5 EXPERIMENTAL EVALUATION

5.1 Experimental Setup

5.1.1 Wearable Device Model. We use the Odroid-XU3 development board [12] to implement CIM for runtime imputation. Odroid-XU3 integrates four high-performance ARM Cortex-A15 cores and four low-power ARM Cortex-A7 cores. The board also includes sensors to measure power consumption of A15 and A7 cores. We choose the Odroid-XU3 board for implementation since it provides sufficient memory to implement a generative approach as a baseline comparison.

We measure the execution time and energy consumption of CIM by running it on the A7 cores. A7 cores are chosen for measurements since our goal is to perform energy-efficient imputation. Moreover, A7 cores are closer to processors used in commercial wearable devices, such as Apple watch [7], when compared to A15 cores. We also note that CIM is not dependent on a particular processor and any other embedded processor can be used to evaluate CIM.

5.1.2 Datasets. We use three publicly available HAR datasets for evaluating CIM. Brief descriptions for each dataset are provided below.

w-HAR [3]: w-HAR is a multi-modal HAR dataset that provides sensor data from a wearable stretch sensor and a 3-axis accelerometer. w-HAR includes data from 22 users for 8 activities: *{Jump, lie down, sit, stand, walk, stairs up/down, and transition}*. The w-HAR dataset is unbalanced and number of examples for stairs up/down and transition are significantly lower. Therefore, we use five activities for our analysis while leaving out stairs up/down and transition.

We simulate missing data in w-HAR by making individual axes of the accelerometer or stretch sensor unavailable at runtime. For instance, in one of the scenarios, it is assumed that x-axis is unavailable while other axes of the accelerometer and stretch sensor are available. We choose to implement missing data for individual axes in w-HAR since it only includes two sensors. Overall, we obtain 14 missing data scenarios for w-HAR by making one, two, or three axes missing.

Shoaib et al. [40]: The dataset from Shoaib et al. [40] includes data from five accelerometers while 10 users perform seven activities: *{Biking, Down stairs, Jogging, Sitting, Standing, Up stairs, and Walking}*. The sensors are mounted in the left pocket, right pocket, wrist, belt, and upper arm. The number of missing sensors for Shoaib dataset is varied from one to four, resulting in a total of 30 scenarios. We assume that all three axes of a sensor are missing for Shoaib et al. dataset.

Table 2. Summary of 1D-CNN Parameters for Three Datasets

Dataset	Input	Conv1	Conv2	Conv3	FC1	FC2
PAMAP2	(3, 1536)	(8, 1536)	(16, 768)	(32, 384)	64	5
w-HAR	(4, 64)	(8, 64)	(16, 32)	(32, 16)	64	5
Shoaib et al.	(5, 600)	(8, 600)	(16, 300)	(32, 150)	64	7

PAMAP2 [36]: The PAMAP2 dataset includes data from three accelerometers for nine users performing five activities: {*lying, sitting, walking, running and cycling*}. The sensors are placed on the wrist of the dominant arm, chest, and the dominant side's ankle. The number of missing sensors is varied from one to two, resulting in a total of 6 missing data scenarios.

We use the window lengths specified in each dataset to evaluate CIM. Specifically, window length for w-HAR is 64 samples, while it is 200×3 and 512×3 samples for Shoaib et al., and PAMAP2 datasets, respectively.

5.1.3 CIM Training and Scalability. Training CIM involves obtaining sensor data clusters and their respective representative windows. CIM also learns the mapping between sensor data clusters for imputation at runtime. We perform the training process *offline* on a Nvidia T4 GPU [29] unit using the Pytorch library [32]. The number of clusters used for sensors in each dataset are as follows: PAMAP2 – 5, w-HAR – 16, Shoaib et al. – 8. The training process for all three datasets takes less than 11 minutes, highlighting the fact that CIM has minimal training overhead.

In general, the number of clusters may increase with the number of classes in a dataset. However, the learning complexity for CIM is primarily dependent on the number of sensor data clusters in the dataset. Specifically, CIM must learn representative windows for multiple clusters and relationship between them. The complexity of this process increases with the number of clusters since CIM has to process additional representative windows. The number of clusters, in turn, depends on the distribution of data for the dataset. For instance, a sensor that exhibits high degree of variations among activities and users may result in more clusters compared to a sensor that does not exhibit high variations. As such, the learning overhead of CIM does not directly depend on the number of activities. Our evaluations show that training CIM with as many as 40 clusters for each sensor takes less than 20 minutes on a Nvidia T4 GPU. The training cost is acceptable since it is a *one-time offline* cost. Once trained, CIM does not need any updates at runtime. Overall, these evaluations show that CIM is able to scale to larger number of clusters with minimal *offline* training cost.

5.1.4 HAR Classifier Representation. We employ a 1-D CNN as the HAR classifier. The 1D-CNN includes three convolutional layers and three maxpool layers followed by two fully connected layers with ReLU activation for activity classification. Table 2 shows the details of 1D-CNN for each dataset. We note that the core architecture is same for all three datasets with the exception of input and output layers due to changes in window length and number of activities. Raw data from sensors are used as input to the 1D-CNN classifier. We utilize 60% of the data for training and 40% is reserved for testing. The Adam optimizer [20] is used for 10 epochs to train both standard and reliable HAR classifiers. The baseline training accuracy for each dataset without any missing data is as follows: w-HAR – 97.2%, Shoaib et al. – 97.5%, PAMAP2 – 98.7%.

5.1.5 Baseline Imputation Methods. We use generative adversarial imputation networks (GAIN) and k-nearest neighbor imputation method as baseline approaches for data recovery. In the following, we provide brief descriptions of each approach.

Table 3. Summary of Clusters for Each Dataset

Dataset	Number of clusters
PAMAP2	5
w-HAR	16
Shoaib et al.	8

Baseline GAIN model: GAIN [44] uses generative adversarial training to obtain a deep neural network that takes observed data and a mask of missing sensor instances as input. Given this input, the generator in GAIN outputs a matrix of imputed sensor values. The primary challenge with GAIN is high memory requirement for storage of generator parameters and computational overhead for each imputation. Furthermore, it has been shown that generative networks need extensive fine-tuning of hyperparameters, which makes their training challenging in nature [4].

k-Nearest neighbors (KNN) imputation: KNN has been widely used for data imputation in prior work [33]. The KNN algorithm first learns distribution of data and nearest neighbors for each missing sample. At the end of training, the KNN imputation algorithm stores a set of training data for runtime usage. At runtime, when there are missing sensors, KNN imputation first calculates k nearest neighbors for a window with missing data using data from available sensors. Then, data for missing sensors are recovered using the k neighbors. The number of neighbors k is a hyperparameter in KNN where higher k generally results in better performance [9]. We implement KNN-based imputation for HAR. We start with a k of 5 and gradually increase it until there are no further improvements in accuracy. After this analysis we choose the value of k as 10 since we did not observe significant accuracy improvements with higher k. While KNN is useful and has been used widely, it suffers from high memory and execution time overhead for runtime usage. Specifically, KNN must store sufficient examples in memory to obtain suitable neighbors, which increases the memory overhead. KNN must obtain distance for sensor data from all stored examples, which can also be computationally expensive.

5.1.6 Evaluation Metrics. Classification accuracy and energy overhead are used as the primary evaluation metrics in this paper since accuracy is crucial for end users, especially in health applications. In addition to accuracy, we calculate precision, recall, and F1 scores for each dataset. Similarly, energy overhead determines the overall battery lifetime of wearable devices and energy-efficient operation can significantly improve adoption of wearable devices [8].

5.2 Analysis of Sensor Data Clusters

We start our experimental validation with an analysis of sensor data clusters and their respective representative windows. For each dataset, we perform a design space exploration with different number of clusters in the k-means algorithm to ensure that we obtain balanced and well-separated clusters. Table 3 shows the number of clusters obtained for each dataset at end of the design space exploration. Same number of clusters are chosen for each sensor for simplicity and ease of presentation. Moreover, we note that clusters are obtained at the level of axes for w-HAR since we consider each axis as a separate sensor. We obtain the largest number of clusters for w-HAR since it shows higher variability in data patterns. We note that CIM does not depend on a particular number of clusters and works with variable number of clusters.

After obtaining sensor data clusters, we obtain representative windows for each cluster. Figure 3 shows windows and representative windows for three clusters of y-axis accelerometer data in the w-HAR dataset. We see that windows of each cluster follow a distinct pattern with some variations.

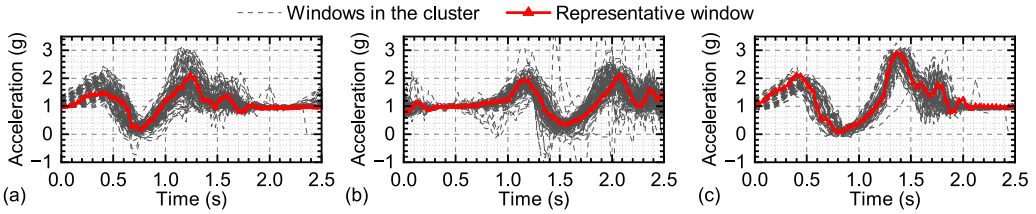

Fig. 3. Illustration of windows in three clusters and their respective representative windows for y-axis in the w-HAR dataset.

For instance, the cluster in Figure 3(a) has a peak around 1.5 s in each window and flattens after about 1.75 s. On the other hand, the cluster in Figure 3(b) has peaks around 1.2 s and 2.0 s. The representative windows for each cluster are shown using a red line with triangle markers. We see that the representative window closely follows data pattern in the cluster for all three cases. Consequently, the representative window will be able to substitute windows in the cluster if a sensor is unavailable at runtime. This shows that our approach is able to separate data into clusters and identify a single representative window for each cluster. Representative windows are used for data imputation and classification as we show in the following sections.

5.3 Validation of Missing Data Detection

One of the key aspects of accurate data imputation and classification is missing data detection. The missing data detection algorithm must identify missing sensors while avoiding false positives. We analyze the performance of missing data detection when missing sensor values are observed as Gaussian noise with zero mean and variance that is representative of actual data range. Gaussian noise is chosen for evaluation since it is more challenging compared to constant data cases. We choose missing detection threshold of 95^{th} percentile for w-HAR and Shoaib et al. and 1.05 for PAMAP2.

Figure 4 shows accuracy of missing data detection for all three datasets. For each dataset, we vary the number of missing channels and variance of Gaussian noise. The variance is chosen as a function of range of actual observations for each sensor. Each bar in Figure 4 shows the average detection accuracy with a given number of missing sensors. For instance, the first bar in Figure 4(a) shows the average detection accuracy for three scenarios of one missing sensor for PAMAP2 dataset. The accuracy number includes both correct predictions and false positives. That is, we flag a detection error whenever a missing sensor is not detected and when a sensor is identified as missing in spite of it being available. We see that the accuracy increases with larger variance values for all three datasets. The accuracy for w-HAR and Shoaib et al. datasets is more than 80% for most of the scenarios, including when there are multiple sensors missing. The detection accuracy for w-HAR is lower when variance is 0.5 because the missing data patterns are similar to actual sensor data. As such, the missing data patterns fall within valid regions of clusters and the algorithm is unable to detect the missing data. At the same time, we note that lower detection accuracy does not significantly impact the classification accuracy. This is because the reliable classifier trained with data augmentation is able to handle small perturbations in data. PAMAP2 has lower accuracies when compared to w-HAR and Shoaib datasets, especially when the variance is lower than 10. This is because PAMAP2 has higher range of data values, which makes missing data detection more challenging with smaller variance. At the same time, we note that classification accuracies are not significantly affected since the 1D-CNN is able to handle small variations in missing data due to our training with augmented data.

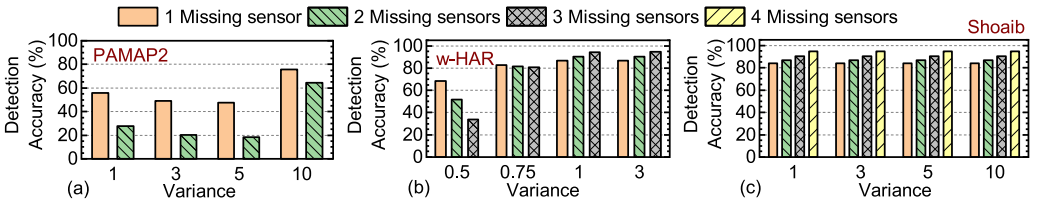

Fig. 4. Missing data detection accuracy for (a) PAMAP2 (b) w-HAR and (c) Shoaib et al. datasets with varying Gaussian noise variance.

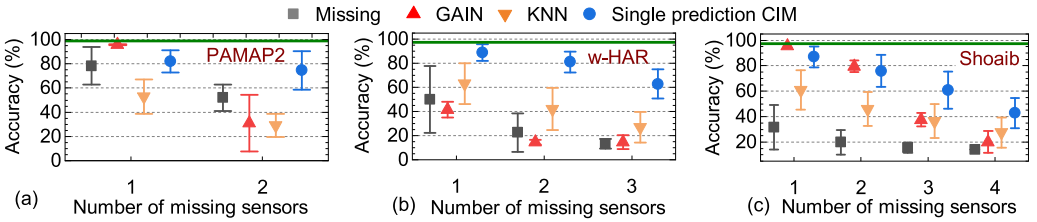

Fig. 5. Comparison of classification accuracy for (a) PAMAP2, (b) w-HAR and (c) Shoaib et al. datasets. We assume that missing data are observed as Gaussian noise with zero mean and unit variance. Accuracy with no missing data is represented with a green reference line.

5.4 Accuracy with CIM Imputation

The next step after missing data detection is data imputation and classification. The missing data configuration for each dataset is Gaussian noise with zero mean and unit variance. We use the proposed mapping table and representative windows to perform imputation for each dataset. This section analyzes accuracy with a single label prediction and results for imputation-aware classification are provided in the next section.

Figure 5 shows a comparison of classification accuracy of CIM against the accuracy with missing data, GAIN, and KNN for all three datasets. Specifically, each point in the figure shows mean and standard deviation of classification accuracy over all possible scenarios with a given number of missing sensors. For example, in the case of two missing sensors in the Shoaib dataset, we obtain the average and standard deviation over (5 choose 2) combinations of possible scenarios. We also show the classification accuracy with actual data using a green reference line. It is seen that CIM achieves higher accuracy compared to GAIN and KNN for most of the scenarios. In particular, as the number of missing sensors increases, the gap between accuracy of CIM and baseline approaches increases significantly. For instance, with two missing sensors for the PAMAP2 dataset, CIM has close to 40% higher mean accuracy than GAIN. Similar accuracy gains are observed with respect to KNN as well. CIM is able to achieve these accuracies with significantly lower overhead compared to GAIN, as we show in Section 5.8. The gap between CIM and GAIN is lower for one missing sensor since GAIN is able to leverage a generator network with a larger number of parameters to impute data more accurately. However, GAIN fails once the number of missing sensors is higher than one. The accuracy for GAIN reduces with increasing number of missing sensors because GAIN depends on observed sensor data to generate missing values. As the number of missing sensors increases, available data for GAIN reduces, thus leading to higher error. Similarly, KNN is unable to find appropriate neighbors in the dataset when multiple sensors are missing, leading to accuracy drop.

Table 4. Comparison of Precision, F1, and Recall Scores for CIM, GAIN, and KNN for PAMAP2 Dataset

	Actual	Number of missing sensors					
		1			2		
		CIM	GAIN	KNN	CIM	GAIN	KNN
Precision	0.988	0.800	0.060	0.736	0.829	0.050	0.448
Recall	0.987	0.791	0.080	0.510	0.711	0.076	0.238
F1-score	0.987	0.819	0.196	0.527	0.746	0.190	0.291

Table 5. Comparison of Precision, F1, and Recall scores for CIM, GAIN, and KNN for w-HAR Dataset

	Actual	Number of missing sensors								
		1			2			3		
		CIM	GAIN	KNN	CIM	GAIN	KNN	CIM	GAIN	KNN
Precision	0.971	0.898	0.800	0.756	0.837	0.163	0.605	0.741	0.136	0.434
Recall	0.971	0.888	0.413	0.628	0.809	0.150	0.420	0.628	0.146	0.268
F1-score	0.971	0.886	0.386	0.641	0.809	0.060	0.395	0.636	0.060	0.222

Table 6. Comparison of Precision, F1, and Recall scores for CIM, GAIN, and KNN for Shoaib Dataset

	Actual	Number of missing sensors											
		1			2			3			4		
		CIM	GAIN	KNN	CIM	GAIN	KNN	CIM	GAIN	KNN	CIM	GAIN	KNN
Precision	0.975	0.876	0.02	0.651	0.769	0.02	0.559	0.631	0.02	0.464	0.446	0.02	0.330
Recall	0.975	0.869	0.14	0.609	0.759	0.14	0.459	0.608	0.14	0.364	0.428	0.14	0.276
F1-score	0.975	0.864	0.04	0.573	0.743	0.04	0.399	0.570	0.04	0.293	0.357	0.04	0.197

CIM has higher accuracy, on average, compared to classification with observed Gaussian noise. In particular, for the Shoaib et al. dataset, the accuracy with missing data is less than 40%, which is a significant degradation. Similar trends are observed for other datasets as well.

Precision, recall, and F1-scores: We also analyze the precision, recall, and F1-scores of all methods for the three datasets as a function of the number of missing sensors. Tables 4–6 show the precision, recall, and F1-scores for PAMAP2, w-HAR, and Shoaib et al. datasets, respectively. The scores are grouped as per the number of missing sensors and each entry in the table gives the average score for the respective missing sensor scenario. We see that the proposed single-prediction CIM approach is able to achieve consistently higher precision, recall, and F1-scores compared to GAIN and KNN. In particular, CIM has significantly higher scores compared to GAIN and KNN when one or two sensors are missing. This is important because the probability of one sensor missing is higher in practice when compared to other scenarios. CIM also maintains higher performance when more than two sensors are missing, as shown in Tables 5 and 6, albeit with a lower gap in scores. Overall, these results show that CIM is able to provide higher precision, recall, and F1-scores in addition to higher accuracy. Higher precision and recall imply that CIM obtains better sensitivity and specificity compared to GAIN and KNN.

In summary, these results show that CIM is able to impute data accurately with low overhead while GAIN is unable to recover accuracy with more than one missing sensor despite having a more complex generator network. CIM also outperforms KNN in all scenarios while avoiding the high memory and computational overhead of KNN.

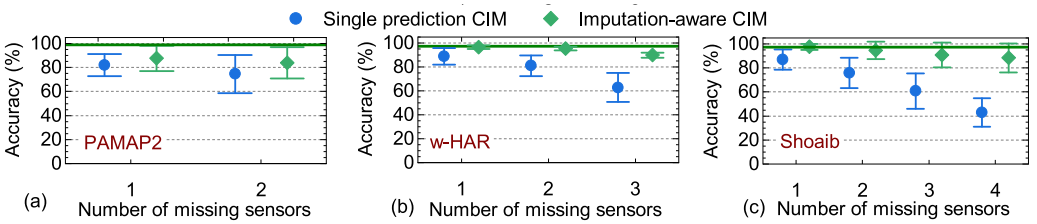

Fig. 6. Comparison of classification accuracy with single prediction CIM and imputation aware CIM for (a) PAMAP2, (b) w-HAR and (c) Shoaib et al. datasets. Accuracy with no missing data is represented with a green reference line.

5.5 Validation of Imputation-Aware Activity Classification

This section analyzes accuracy of imputation-aware classification proposed in CIM. Recall that due to uncertainty in predictions with data imputation, we provide a set of labels in the imputation-aware classification. Specifically, we provide a set of labels with a 90% confidence level by setting $\alpha = 0.90$. The primary advantage of this method is that it provides a larger set of activity predictions when uncertainty is higher, which enables users to make informed decisions from activity labels.

Figure 6 shows the accuracy of imputation-aware CIM and single prediction CIM outlined in the previous section. We see that for all three datasets, imputation-aware CIM is able to achieve significantly higher accuracy when compared to the single-prediction CIM. Indeed, for the Shoaib dataset, accuracy of imputation-aware CIM is close to baseline accuracy with no missing data. Even for PAMAP2 dataset where single prediction version already provides high accuracy, imputation-aware CIM is able to improve the classification accuracy. Our evaluations also show that the prediction set for all three datasets contains, on average, two activities (out of a potential five to seven activities). Moreover, the activity set consists of at most two labels when a single sensor is missing for all datasets. The size of the prediction set intermittently increases to three or four when two or more sensors are missing. Length of the prediction is an important metric because larger prediction sets are not as useful as smaller prediction sets and CIM is able to maintain tight prediction sets for all three datasets.

In summary, our experimental evaluations show that CIM is able to accurately learn representative windows for sensor data, detect missing sensors, and provide imputation to enable accurate activity classification. CIM is able to achieve significantly higher accuracy than state-of-the-art GAIN and KNN methods while avoiding expensive computations and memory overhead at runtime.

5.6 CIM Accuracy with Random Missing Data

Experimental evaluations so far have assumed that missing sensors are not available for the entire window. While this is useful, missing data may occur randomly in real world scenarios. Therefore, this section evaluates performance of CIM in the random missing data scenario.

The evaluation in this section uses the following configurations for missing data percentage: {10%, 20%, 30%, 40% and 50%}. The random missing data are applied to all possible missing data scenarios in all three datasets. That is, in each window we make a randomly chosen set of samples missing. For instance, 10% of samples in a window are made missing when the missing data percentage is 10%. Following random missing data generation, we use the CIM representative windows and classifier to impute missing data and perform classification. That is, we *do not re-train CIM* to handle random missing data. We use both single-prediction and imputation-aware versions of CIM to evaluate the accuracy with random missing data.

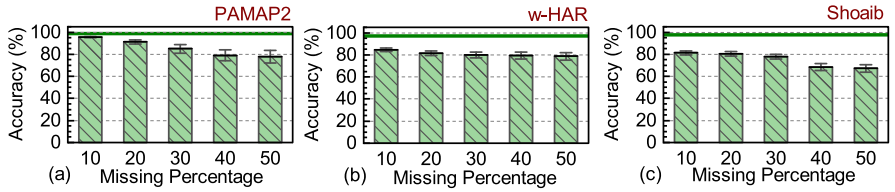

Fig. 7. Classification accuracy with single prediction CIM with random missing percentages for (a) PAMAP2, (b) w-HAR and (c) Shoaib et al. datasets.

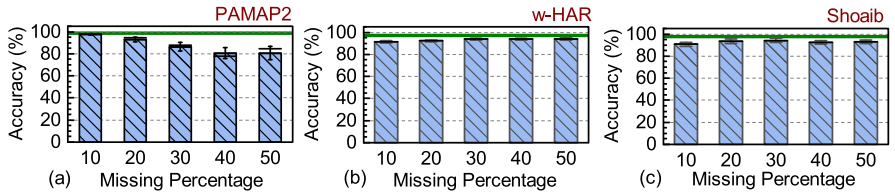

Fig. 8. Classification accuracy with Imputation-aware CIM with random missing percentages for (a) PAMAP2, (b) w-HAR and (c) Shoaib et al. datasets.

Figure 7 shows the average accuracy and associated standard deviation with random missing data for all datasets. Single-prediction CIM is used to impute data and predict activities in the presence of random missing data. The x-axis denotes the percentage of random missing data in each window while the y-axis shows the average accuracy and standard deviation. The figure also shows the baseline accuracy with no missing data using a green line. We take the average of accuracy across all missing data scenarios in each dataset. For instance, for w-HAR, each point is an average of $2^4 - 2$ missing data scenarios. The figure shows that CIM is able to handle random missing data with average accuracies that are within 20% of baseline accuracy even when 50% of data are randomly missing. Overall, this analysis shows that CIM performs similar or better than the case of missing data in the entire window.

Next, Figure 8 shows the average accuracy and associated standard deviation with random missing data for all datasets when using imputation-aware CIM. Similar to the previous figure, each point on the figure shows average accuracy across all missing data scenarios with a given percentage of random missing data. The figure shows that imputation-aware CIM is able to achieve accuracy that is within 10% of the baseline accuracy with no missing data for all scenarios. In summary, analysis in this section shows that CIM is able to handle both block and random missing data scenarios with just one set of training, making it widely applicable.

5.7 Training Classifiers with Missing Data

Another option to handle missing data is to train classifiers with missing data examples so that runtime imputation is not needed. To this end, we train a classifier with zero substituted missing data. Specifically, we augment training data with windows that have zeros substituted for missing sensors. The trained classifier is used at runtime to recognize activities in the presence of missing data. Classifier with zeros in training for missing data shows greater than 90% for all datasets. Test accuracy is also greater than 90% when missing data are represented with zeros. However, if the missing data distribution changes from training, the accuracy drops, as shown in Figure 9. Specifically, the figure shows accuracy of classification when missing data are observed as Gaussian noise. The accuracy for all three datasets except one missing sensor in PAMAP2 drops significantly to

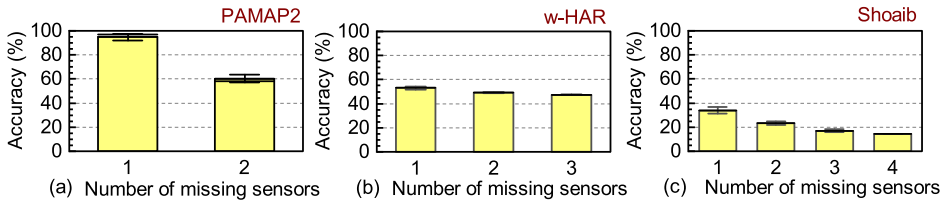

Fig. 9. Accuracy of classifier trained with missing data for (a) PAMAP2, (b) w-HAR, and (c) Shoaib et al.

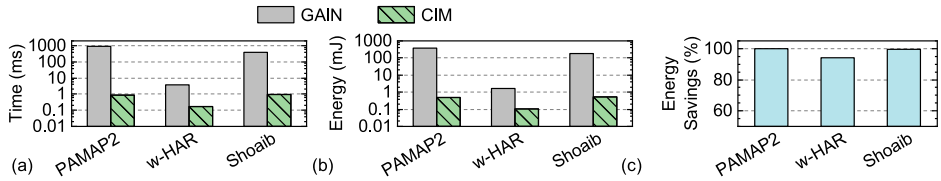

Fig. 10. Comparison of overhead for (a) execution time and (b) energy overhead for GAIN and CIM. Y-axis is represented in log scale for parts (a) and (b). (c) Energy savings achieved by CIM when compared to GAIN.

less than 50%. In contrast, CIM is able to maintain higher accuracies, as shown in Figure 5. The key challenge in training classifier with missing data is that it requires prior knowledge of missing data patterns for all scenarios. In summary, proposed single-imputation and imputation-aware CIM approaches are able to both detect and impute missing data before classification, thus providing reliable classification with missing data.

5.8 Implementation Overhead

We implement the proposed CIM approach on the Odroid-XU3 board to measure execution time, energy consumption, and memory overhead. We also compare the overhead of CIM with GAIN to understand potential of energy savings with CIM. The primary execution time and energy consumption overhead for single prediction CIM consists of missing data detection, cluster prediction, and imputation. Figure 10 shows a comparison of the execution time and energy consumption overhead per imputation for the three datasets. We see that execution time overhead of CIM is lower than 1 ms for all three datasets while execution time for GAIN is close to 1 s for PAMAP2 and Shoaib et al. datasets. Execution time of GAIN for w-HAR is lower since window sizes for w-HAR are smaller. Similar to execution time, CIM has significantly lower energy consumption compared to GAIN. For instance, for the PAMAP2 dataset GAIN consumes 372 mJ for each imputation while CIM consumes just 0.5 mJ. Overall, CIM achieves close to 100% energy savings compared to GAIN, as shown in Figure 10(a). These savings can significantly increase the operating time of wearable devices and improve their adoption for health applications.

The energy overhead for imputation-aware CIM is slightly higher since we have to perform multiple activity classifications to obtain prediction sets. Each classification using the 1D-CNN takes about 13 ms and 8 mJ of energy. This additional overhead is still lower than GAIN and we can tune the number of classifications at runtime to lower the overhead, as noted in Section 4.6.

Next, we analyze the memory overhead of CIM and GAIN. Memory overhead for CIM consists of the mapping table and cluster information. Similarly, the memory overhead for GAIN consists of generator weights. Table 7 compares the memory overhead for both approaches. We see that CIM has significantly lower overhead compared to GAIN. For instance, GAIN consumes 43 MB for the Shoaib et al. dataset, which is more than the memory available in a number of low-power

Table 7. Summary of Memory Overhead for CIM and GAIN

Dataset	CIM			GAIN (MB)
	Mapping table (KB)	Cluster information (KB)	Total (KB)	
PAMAP2	2	185	187	102
w-HAR	46	45	91	0.33
Shoaib et al.	22	197	219	43

processors. In contrast, the overhead for CIM is only 219 KB, which is almost 200 times lower than GAIN. The memory savings for w-HAR dataset are lower since the input size for GAIN is smaller and CIM stores a larger number of clusters. Even with more clusters, CIM is able to have memory overhead that is about four times lower than GAIN. In summary, CIM outperforms state-of-the-art imputation approaches in terms of accuracy while having significantly lower overhead.

6 CONCLUSIONS AND FUTURE WORK

HAR forms an important component of several health applications such as movement disorders, rehabilitation, and fitness monitoring. HAR using multiple sensors is susceptible to missing data due to energy constraints, user error, or communication challenges. This paper proposed CIM, a clustering-based imputation method for energy-efficient missing data recovery in HAR. CIM first defined a set of possible clusters and representative data patterns for each sensor in HAR. Then, we created and stored a mapping between clusters across sensors. At runtime, when data from a sensor are missing, we utilized the stored mapping table to obtain the most likely cluster for the missing sensor. The representative window for the identified cluster is then used as imputation to perform activity classification. We also proposed an imputation-aware HAR classification that provides a set of activities due to higher uncertainty with missing data. Experiments with three HAR datasets showed that single-prediction version of CIM achieves accuracy that is within 10% of accuracy with no missing data for one missing sensor. The accuracy gap reduces to about 1% with imputation-aware classification. Energy measurements on the Odroid-XU3 board showed that CIM has close to 100% energy savings compared to state-of-the-art generative approaches. Our immediate future work includes extending the approach to other health applications.

REFERENCES

[1] Antonio A. Aguileta, Ramon F. Brena, Oscar Mayora, Erik Molino-Minero-Re, and Luis A. Trejo. 2019. Multi-sensor fusion for activity recognition–a survey. *Sensors* 19, 17 (2019), 3808.
[2] Muhammad Arif, Mohsin Bilal, Ahmed Kattan, and S. Iqbal Ahamed. 2014. Better physical activity classification using smartphone acceleration sensor. *J. of Med. Syst.* 38, 9 (2014), 95.
[3] Ganapati Bhat, Nicholas Tran, Holly Shill, and Umit Y. Ogras. 2020. w-HAR: An activity recognition dataset and framework using low-power wearable devices. *Sensors* 20, 18 (2020), 5356.
[4] Eoin Brophy, Zhengwei Wang, Qi She, and Tomas Ward. 2021. Generative adversarial networks in time series: A survey and taxonomy. *arXiv preprint arXiv:2107.11098* (2021), 25.
[5] Seungeun Chung, Jiyoun Lim, Kyoung Ju Noh, Gague Kim, and Hyuntae Jeong. 2019. Sensor data acquisition and multimodal sensor fusion for human activity recognition using deep learning. *Sensors* 19, 7 (2019), 1716.
[6] Ton De Waal, Jeroen Pannekoek, and Sander Scholtus. 2011. *Handbook of Statistical Data Editing and Imputation.* Vol. 563. John Wiley & Sons.
[7] Paul Dempsey. 2015. The teardown: Apple watch. *Engg. & Tech.* 10, 6 (2015), 88–89.
[8] Alberto J. Espay et al. 2016. Technology in parkinson's disease: Challenges and opportunities. *Movt. Disorders* 31, 9 (2016), 1272–1282.
[9] Jerome Friedman, Trevor Hastie, and Robert Tibshirani. 2001. *The Elements of Statistical Learning*, Vol. 1. Springer.
[10] Zijian Guo, Yiming Wan, and Hao Ye. 2019. A data imputation method for multivariate time series based on generative adversarial network. *Neurocomputing* 360 (2019), 185–197.

[11] Greg Hamerly and Jonathan Drake. 2015. Accelerating Lloyd's algorithm for k-means clustering. *Partitional Clustering Algorithms* (2015), 41–78.
[12] Hardkernel. 2014. ODROID-XU3. https://www.hardkernel.com/shop/odroid-xu3/ Accessed 11/20/2020. (2014).
[13] Dustin A. Heldman, Denzil A. Harris, Timothy Felong, Kelly L. Andrzejewski, E. Ray Dorsey, Joseph P. Giuffrida, Barry Goldberg, and Michelle A. Burack. 2017. Telehealth management of parkinson's disease using wearable sensors: An exploratory study. *Digital Biomarkers* 1, 1 (2017), 43–51.
[14] Tahera Hossain, Md Atiqur Rahman Ahad, and Sozo Inoue. 2020. A method for sensor-based activity recognition in missing data scenario. *Sensors* 20, 14 (2020), 3811.
[15] Tahera Hossain and Sozo Inoue. 2019. A comparative study on missing data handling using machine learning for human activity recognition. In *2019 Joint 8th Int. Conf. on Informatics, Electron.& Vision (ICIEV) and 2019 3rd Int. Conf. on Imaging, Vision & Pattern Recognition (icIVPR)*. 124–129.
[16] An-Lun Hsu, Pei-Fang Tang, and Mei-Hwa Jan. 2003. Analysis of impairments influencing gait velocity and asymmetry of hemiplegic patients after mild to moderate stroke. *Arch. Phys. Med. Rehabil.* 84, 8 (2003), 1185–1193.
[17] Dina Hussein, Taha Belkhouja, Ganapati Bhat, and Janardhan Rao Doppa. 2022. Reliable machine learning for wearable activity monitoring: Novel algorithms and theoretical guarantees. In *Proc. Int. Conf. on Comput.-Aided Des. (ICCAD'22)*. 1–9.
[18] Dina Hussein, Aaryan Jain, and Ganapati Bhat. 2022. Robust human activity recognition using generative adversarial imputation networks. In *2022 Design, Automation & Test in Europe Conference & Exhibition (DATE'22)*. 84–87.
[19] Gareth James, Daniela Witten, Trevor Hastie, and Robert Tibshirani. 2013. *An Introduction to Statistical Learning*, Vol. 112. Springer.
[20] Diederik P. Kingma and Jimmy Ba. 2015. Adam: A method for stochastic optimization. In *The Int. Conf. on Learning Representations (Poster'15)*.
[21] Linghe Kong, Mingyuan Xia, Xiao-Yang Liu, Min-You Wu, and Xue Liu. 2013. Data loss and reconstruction in sensor networks. In *2013 Proceedings IEEE INFOCOM*. 1654–1662.
[22] Kai Kunze and Paul Lukowicz. 2014. Sensor placement variations in wearable activity recognition. *IEEE Perv. Comput.* 13, 4 (2014).
[23] Jennifer R. Kwapisz, Gary M. Weiss, and Samuel A. Moore. 2011. Activity recognition using cell phone accelerometers. *SigKDD Explorations News.* 12, 2 (2011), 74–82.
[24] Oscar D. Lara and Miguel A. Labrador. 2012. A survey on human activity recognition using wearable sensors. *IEEE Commun. Surveys & Tut.* 15, 3 (2012), 1192–1209.
[25] Shengzhong Liu et al. 2020. Handling missing sensors in topology-aware IoT applications with gated graph neural network. *Proc. Interactive, Mobile, Wearable and Ubiquitous Tech.* 4, 3 (2020), 1–31.
[26] Yonghong Luo, Xiangrui Cai, Ying Zhang, Jun Xu, and Xiaojie Yuan. 2018. Multivariate time series imputation with generative adversarial networks. In *Proc. Advances in Neural Information Processing Systems*. 1603–1614.
[27] Walter Maetzler, Jochen Klucken, and Malcolm Horne. 2016. A clinical view on the development of technology-based tools in managing Parkinson's disease. *Movement Disorders* 31, 9 (2016), 1263–1271.
[28] Abdullah Mamun, Seyed Iman Mirzadeh, and Hassan Ghasemzadeh. 2022. Designing deep neural networks robust to sensor failure in mobile health environments. In *2022 44th Annual International Conference of the IEEE Engineering in Medicine & Biology Society (EMBC'22)*. 2442–2446.
[29] Nvidia. 2022. Nvidia Tesla T4 GPU. [Online] https://www.nvidia.com/en-in/data-center/tesla-t4/, accessed May 28, 2023. (2022).
[30] Henry Friday Nweke, Ying Wah Teh, Uzoma Rita Alo, and Ghulam Mujtaba. 2018. Analysis of multi-sensor fusion for mobile and wearable sensor based human activity recognition. In *Proceedings of the Int. Conf. on Data Proc. and Appl.* 22–26.
[31] Mohanad Odema, Nafiul Rashid, and Mohammad Abdullah Al Faruque. 2021. Energy-aware design methodology for myocardial infarction detection on low-power wearable devices. In *Proc. 26th Asia and South Pacific Des. Autom. Conf.* 621–626.
[32] Adam Paszke et al. 2019. Pytorch: An imperative style, high-performance deep learning library. *Proc. Advances in Neural Information Processing Systems* 32 (2019).
[33] Ivan Miguel Pires, Faisal Hussain, Nuno M. Garcia, and Eftim Zdravevski. 2020. Improving human activity monitoring by imputation of missing sensory data: Experimental study. *Future Internet* 12, 9 (2020), 155.
[34] Okyza M. Prabowo, Kusprasapta Mutijarsa, and Suhono Harso Supangkat. 2016. Missing data handling using machine learning for human activity recognition on mobile device. In *Proc. Int. Conf. on ICT for Smart Society*. 59–62.
[35] Nafiul Rashid, Berken Utku Demirel, and Mohammad Abdullah Al Faruque. 2022. AHAR: Adaptive CNN for energy-efficient human activity recognition in low-power edge devices. *IEEE Internet of Things J.* 9, 15 (2022), 13041–13051.
[36] Attila Reiss and Didier Stricker. 2012. Introducing a new benchmarked dataset for activity monitoring. In *Int. Symp. Wearable Comput.* 108–109.

[37] Aaqib Saeed, Tanir Ozcelebi, and Johan Lukkien. 2018. Synthesizing and reconstructing missing sensory modalities in behavioral context recognition. *Sensors* 18, 9 (2018), 2967.
[38] Farzad Samie, Lars Bauer, and Jörg Henkel. 2016. IoT technologies for embedded computing: A survey. In *Proc. Int. Conf. on Hardware/Software Codesign and System Synthesis*. 1–10.
[39] Glenn Shafer and Vladimir Vovk. 2008. A tutorial on conformal prediction. *J. Machine Learn. Research* 9, 3 (2008).
[40] Muhammad Shoaib, Stephan Bosch, Ozlem Durmaz Incel, Hans Scholten, and Paul J. M. Havinga. 2014. Fusion of smartphone motion sensors for physical activity recognition. *Sensors* 14, 6 (2014), 10146–10176.
[41] Sabera Talukder, Jennifer J. Sun, Matthew Leonard, Bingni W. Brunton, and Yisong Yue. 2022. Deep neural imputation: A framework for recovering incomplete brain recordings. *arXiv:2206.08094* (2022).
[42] Chen Xu and Yao Xie. 2020. Conformal prediction for dynamic time-series. *arXiv preprint arXiv:2010.09107* (2020).
[43] Hui-Shyong Yeo, Byung-Gook Lee, and Hyotaek Lim. 2015. Hand tracking and gesture recognition system for human-computer interaction using low-cost hardware. *Multimedia Tools and Applications* 74 (2015), 2687–2715.
[44] Jinsung Yoon, James Jordon, and Mihaela Schaar. 2018. GAIN: Missing data imputation using generative adversarial nets. In *Proc. Int. Conf. on Machine Learn.* 5689–5698.
[45] Zhao Zhao, S. Ali Etemad, and Ali Arya. 2016. Gamification of exercise and fitness using wearable activity trackers. In *Proc. Int. Symp. on Comput. Science in Sports*. 233–240.

Received 23 March 2023; revised 2 June 2023; accepted 13 July 2023