# Towards Wireless Environment Cognizance Through Incremental Learning

Aniqua Baset
*University of Utah*
aniqua@cs.utah.edu

Christopher Becker, Kurt Derr, Samuel Ramirez
*Idaho National Lab*
christopher.becker, kurt.derr, samuel.ramirez@inl.gov

Sneha Kasera, Aditya Bhaskara
*University of Utah*
kasera, bhaskara@cs.utah.edu

***Abstract*—With the tremendous increase in the use of wireless devices, understanding the surrounding wireless/RF environment is becoming essential for many application areas. In this work, we develop the technical building blocks needed for a spectrum monitoring system that can incrementally learn about the signals present in a deployed environment. We achieve "incremental learning (IL)" by identifying and grouping the new/unknown signals and, automatically building new machine learning (ML) models for detecting them. A thorough evaluation of our approach demonstrates its adaptability and high accuracy with signal data from several over-the-air scenarios.**

## I. Introduction

Monitoring and understanding the surrounding wireless/RF environment is a long-coveted ability. Past motivations for such ability mainly came from defense/military needs. With the tremendous growth in the use of wireless devices, such ability for wireless-environment cognizance (WEC) is becoming relevant in nonmilitary application areas as well. For instance, modern infrastructure and industrial facilities are increasingly relying on wireless devices for automation, monitoring and control of equipment, inventory tracking, etc [1]. There have been several reports of industrial process disruption by interference from personal/external devices, leading to downtime and safety hazards [1]. Concerns like these have created a growing need for not just continuous monitoring, but cognizance of the surrounding wireless environment to distinguish between the new/unexpected signals and the known/expected signals.

In this work, we take an incremental learning (IL) approach to realize WEC. We achieve IL by identifying and grouping the new/unknown signals present in the deployed environment and automatically building new machine learning (ML) models for detecting them[1]. Fig. 1 depicts our spectrum monitoring system for WEC using IL. Our system comprises of three modules: *learning*, *known signal(s) classification with unknown signal detection (KSC+USD)*, and *clustering*. At the start, the *learning* module learns the necessary ML models for classification of known signal classes from labeled input data. The KSC+USD module utilizes the current learned models to determine whether or not an input signal, to be classified, belongs to one of the known classes. If the input signal does not belong to any known class, the KSC+USD module classifies it as *unknown* and feeds it to the *clustering* module. After receiving sufficient number of unknown signals, the *clustering* module uses unsupervised ML to cluster them, not necessarily in real-time, into groups. Next, this group data is fed into the *learning* module which learns the necessary ML models for each group. TheKSC+USD module can now classify an input as belonging to one of the known classes or one of the found unknown classes.

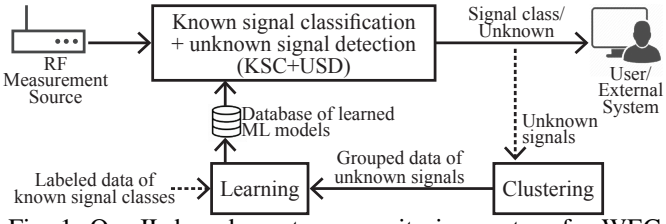


Fig. 1: Our IL based spectrum monitoring system for WEC

The benefits of our IL based spectrum monitoring system for WEC are manifold. IL can help a monitoring system to adapt to the anomalies/deviations present in different deployed environments by adding ML classification models for the detected deviations. IL can also reduce manual effort in preparing a monitoring system for a new deployed environment—instead of manually collecting new signal data, labeling it, and training ML models, we can just let the system run and learn about the environment. To the best of our knowledge, we are the first to present a spectrum monitoring system with IL capability for realization of WEC.

**Key challenges and our approach:** We describe the key challenges in designing different modules of our system and our approaches to tackle those below.

First, for the KSC+USD module, we need a classification process that can classify an incoming measurement or input to one of the known/expected signal classes and at the same time, can detect the presence of new/unknown signals. Techniques for signal type and/or modulation classification have been heavily researched in the past [2]–[10]. These techniques, however, are designed to classify signals only as one of the trained or known signal-classes. When fed with an unknown signal class or simply background noise, these techniques would incorrectly classify the input as one of the known classes, rather than acknowledging it as 'new'. To address this challenge, we design our KSC+USD module to use a combination of novelty detectors each dedicated to a different known signal class. Novelty detection is a class of machine

[1]Different than IL in ML nomenclatures where traditionally IL refers to updating models based on new data. On the contrary, we are updating the classification support hence the list of the ML models.

learning algorithm that can determine whether an input is novel or not-novel (i.e., whether the input belongs to the same class as the training data or not). If all of the novelty detectors for known classes predict an input as novel, our system concludes that the input is either from an unknown signal class or is just background noise. Our system uses an ML-based approach to distinguish between noise and signal to filter out noise inputs. Our use of a combination of novelty detectors serves a second purpose—it provides us with a flexible classification system that can easily incorporate support for classifying unknown signals for the IL capability. Our design for KSC+USD is easily extensible to new signal classes since adding or removing signal classification support does not require retraining or changing previously learned novelty detectors.

The second challenge is the clustering of the detected unknown signals. Though some of the recent works on unsupervised learning for wireless signals [11], [12] develop data representation for clustering, they do not investigate the clustering process itself in details. We develop a custom, hierarchical clustering approach that does not require prior knowledge of the possible number of unknown signal clusters in the wireless environment. Our clustering method initially puts all signal data points in a single cluster, and iteratively divides the initial cluster and subsequent clusters until no significantly different clusters could be formed. Besides the clustering algorithm, we also need to tackle some practical challenges. For instance, the dataset that we need to work with for clustering can be very large depending on the target environment (e.g., we can get ∼60,000 128-bin Fast Fourier Transforms (FFTs) in just one minute from IEEE802.11g beacons alone). In order to reduce the data samples to practical levels, we exploit the fact that sequential data points are most likely part of the same signal transmission, and hence belong to the same cluster. Therefore, instead of clustering all the data points, we first group consecutive data points and only consider the group means for clustering. This grouping tremendously reduces the data points that we must cluster. Furthermore, we find that the use of group means, instead of all the data samples, reduces the focus on variations present in data points and puts more attention on the variations across different signal classes.

Finally, in practical scenarios, signals will not always align with our monitoring center frequency and can appear shifted in the frequency domain depending on their operating channels. For instance, when observing a 25MHz band centered at 2.437GHz, we can encounter ZigBee signals centered at 2.430GHz, 2.435GHz, or 2.440GHz. In the KSC+USD module, the novelty detector for ZigBee must predict ZigBee from all of these three channels as not novel even if they are shifted from those in the training dataset. Likewise, the *clustering* module should place ZigBee signal data from all of these channels in same cluster rather than forming three separate clusters. To this end, we use a *shift-invariant* feature representation of signal data which is insensitive to frequency shifts for both KSC+USD and *clustering* modules. Again, this practical challenge of detecting shifted signals were not investigated previously in the traditional signal/modulation detection works cited earlier.

We evaluate our KSC+USD module with IEEE802.11g and ZigBee as known signals. Using over-the-air signal data from different bands and environments, we show that the KSC+USD can classify known and unknown signals with 98.29% and 99.65-100% accuracy, respectively. Our signal versus noise classification in KSC+USD also shows 99% or higher accuracy, even when the Signal-to-Noise-Ratio (SNR) is very low. For the *clustering* module, we show that our clustering process can find the exact number of signal clusters or a close number in various over-the-air settings. Finally, our novelty detectors built automatically from clustering results achieve similar performance as in the case of labeled data—98.45% accuracy for known and 91.4% or higher accuracy for unknown signals.

**Summary of our contributions:**

- A novel methodology for unknown signal detection along with known signal classification.
- A novel methodology for unknown signal classification aka clustering along with a set of pre- and post-clustering steps that leverage the domain information.
- A thorough evaluation of our approach that demonstrates its adaptability and high accuracy with signal data from several over-the-air scenarios.

## II. Known Signal Classification + Unknown Signal Detection (KSC+USD) Module

The workflow of our KSC+USD module is presented in Fig. 2. This module continuously receives In-phase and Quadrature (IQ) samples from an RF measurement source. It forms an IQ set with $|S_{IQ}|=N_L \times N_F$ samples and makes a classification decision for each such sample set. It first enqueues each of the IQ sets to a fixed size queue, $Q$ using a smart enqueuing algorithm, SENQUEUE, outlined in Algorithm 1. When the classification process for a IQ set is completed, our module dequeues the next IQ set and computes $N_L$ FFTs where the number of bins for each of the FFTs is $N_F$ (denoted as $S_{FFT}$ in Fig. 2). Next, our module uses a set of Novelty Detectors (ND) for the known signal classes. The ND for signal class $k_i$ predicts if an input is novel (i.e., it does not belong to class $k_i$) or not-novel (i.e., it belongs to class $k_i$). If all of the NDs find an input to be novel, the input is either from an unknown signal class or background noise. Then, our module also forwards the $S_{FFT}$ to a Signal/Noise Classifier (SNC) block to determine whether an input is signal or noise. Finally, our module makes the final classification decision based on the results from the SNC and the NDs. In the rest of this section, we discuss the details of SENQUEUE, the NDs, the SNC, and the final classification decision process.

### A. Smart enqueuing algorithm, SENQUEUE

The purpose of the queue, $Q$, is to temporarily hold incoming IQ sets until these are classified by the classification process. The arrival rate at $Q$ can be very high if we want to

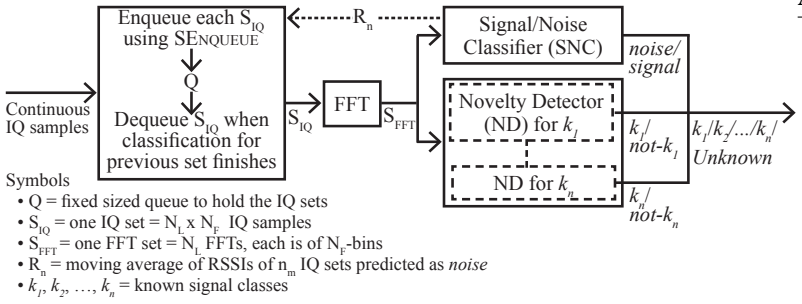


Fig. 2: Workflow of the KSC+USD module

**Algorithm 1** Smart sample enqueuing algorithm

**procedure** SENQUEUE($S_{IQ}$, $R_n$, $p_s$, $p_n$, $p_r$, $p_a$)
  $R \leftarrow$ RSSI of the input IQ set, $S_{IQ}$
  $r \leftarrow$ RAND([0, 1])
  **if** $(R > R_n \ \& \ r < p_s) \ || \ (R < R_n \ \& \ r < p_n)$ **then**
    $r' \leftarrow$ RAND([0, 1])
    **if** $Q$ is full $\& \ r' < p_r$ **then**
      Remove one IQ set from a random position of $Q$ and append $S_{IQ}$ to $Q$
    **else if** $Q$ is not full $\& \ r' < p_a$ **then**
      Append $S_{IQ}$ to $Q$

utilize the maximum capacity of an RF measurement source and monitor comparatively wider spectrum at once (e.g., we use an USRP X310 with a sample rate of 25e6 samples/sec as the RF measurement source in our system). However, the processing rate of IQ sets at $Q$ is likely to be comparatively slower because of the ML computations needed for the classification process. Therefore, $Q$ will overflow occasionally and incoming IQ sets can be dropped. While using higher compute power and/or efficient implementation of the classification process can lead to reducing the loss of IQ sets, this loss cannot be totally avoided. Hence, we devise SENQUEUE (Algorithm 1) to allow the KSC+USD module to intelligently drop IQ sets when needed. SENQUEUE considers an IQ set to be queued in $Q$ with probability $p_s$ if the computed Received Signal Strength Indicator (RSSI) of the set is higher than the average noise RSSI, $R_n$; otherwise, queues the set with probability $p_n$. We set $p_s > p_n$, so SENQUEUE favors IQ sets that are more likely to contain signals over noise. Initially, $R_n$ is initialized to a very low value. After some IQ sets are classified as *noise* by the SNC block, the value of $R_n$ is automatically updated accordingly. This allows the SENQUEUE to adapt to the noise level of the deployed environment. Next, to avoid the $Q$ to be filled up quickly at the beginning, an input set is dropped with probability $p_a$ even when there is room in $Q$. When $Q$ is full, an incoming IQ set replaces the last element of $Q$ with probability $p_r$. In summary, SENQUEUE allows the KSC+USD module to automatically balance the rate of incoming samples and rate of the classification process without affecting the RF measurement source. Such a design makes our system usable across different compute platforms.

### *B. Novelty Detectors (NDs)*

An ND for a signal class $k_i$ determines whether or not an input belongs to $k_i$. The first challenge for the ND is that it needs to use an ML algorithm that can learn about the characteristics of $k_i$ only from training data of $k_i$. Traditional binary classification algorithms are thus not suitable for our purpose since they require training data from both positive ($k_i$) and negative (not $k_i$) cases. Therefore, we use a *novelty detection* algorithm for the ND.

The next challenge for the ND is that it must be frequency shift-invariant, i.e., it should be able to detect a signal even if the signal appears to be shifted in the frequency domain in comparison to those used for training. We require that the ML model for the ND learns the characteristics or patterns of $k_i$ irrespective of the frequency shift. We can achieve frequency shift-invariancy either by using (i) a shift-invariant feature representation as input to a novelty detection algorithm or (ii) a more complex algorithm that itself can learn shift-invariancy of input data. Learning the shift-invariancy can be hard since there will be more hyperparameters to tweak. Moreover, we must make sure that the ML model indeed learns a shift-invariant representation of the input signal. Therefore, we opt for a simple algorithm with the shift-invariant feature representation. This approach is especially useful for the case when we need to automatically learn NDs for unknown signal classes that we find (in Sec. IV).

As the shift-invariant feature representation for an ND, we use the m-SCF that we derive from the Spectral Correlation Function (SCF)[2]. We first compute SCF using the *time smoothing* [13] method as follows:

$$\text{SCF}(f, \alpha) = \frac{1}{N_L} \sum_{l=1}^{N_L} \text{FFT}_l[f] \times \text{FFT}_l^*[f-\alpha] \qquad (1)$$

where, $\text{FFT}_l[f]$ is the $l$th FFT of the signal at frequency $f$, $\text{FFT}_l^*[f-\alpha]$ is the complex conjugate of the FFT of the signal at frequency bin, $f$, shifted by $\alpha$. Next, we compute m-SCF from the magnitude of complex SCF as follows:

$$\text{m-SCF} = minmax\_norm_{01}(\max_f |\text{SCF}(f, \alpha)|) \qquad (2)$$

where, $minmax\_norm_{01}$ normalizes input to new range [0,1]

The resulting m-SCF is an $N_F$-sized array where $N_F$ is the number of FFT bins being used. We do a min-max normalization of the m-SCF so that same signals with different power levels result in a similar m-SCF. The m-SCF of different signals results in different patterns (Fig. 3a) as a specific pattern depends on the signal waveform as well as the bandwidth. Most importantly, same signals result in almost similar pattern even if the signals are shifted in frequencies. For example, ZigBee signals of different channels (Fig. 4a) have almost similar m-SCF (Fig. 4b). However, different target signals can have similar looking patterns in their corresponding m-SCF when they have comparatively narrower bandwidths than the monitoring bandwidth. For example, if we use 25MHz as the monitoring bandwidth, the m-SCF patterns of ZigBee (2MHz) and Bluetooth (1MHz) are quite similar (Fig. 3b).

Finally, we need to choose a novelty detection algorithm

[2]Different than $\alpha$-profile from [7], [8] which is computed from Spectral Coherence Function, not from SCF as in our case.

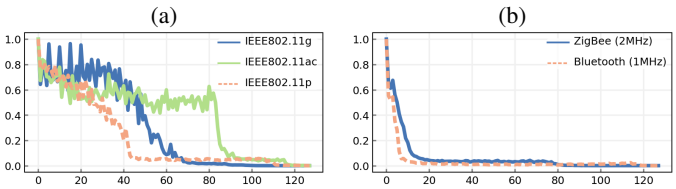


Fig. 3: m-SCF of different signals

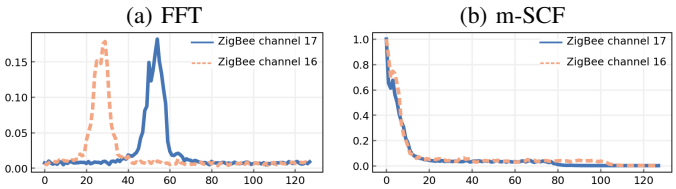


Fig. 4: Shift-invariency of m-SCF

for the ND that can detect such small variances in the m-SCF of different signals and at the same time is invariant to the differences in m-SCF of same signal. For this purpose, we investigate a one-class SVM [14] and an autoencoder [15] based novelty detection. We find that the one-class SVM is inadequate for the ZigBee vs Bluetooth case, while the autoencoder based novelty detection is very accurate. Therefore, we use an autoencoder based approach for the ND. An autoencoder is a type of neural network that has the same number of input and output nodes and tries to learn to output a vector $\hat{\boldsymbol{X}}$ from an input vector $\boldsymbol{X}$ such that $\hat{\boldsymbol{X}} \approx \boldsymbol{X}$ [16]. The autoencoder network can be used for novelty detection based on the reconstruction error ($\|\boldsymbol{X}-\hat{\boldsymbol{X}}\|^2$). When an autoencoder is trained only on data from a particular class, the reconstruction errors will be lower for inputs that belong to same class as the training data since the trained network has learned how to represent the training data. In contrast, the errors will be higher for data from other classes than the trained one. Therefore, our autoencoder based ND for signal class $k_i$ uses the reconstruction error to decide if an input belongs to $k_i$ or not. Specifically, we consider an input m-SCF, $\boldsymbol{X}$ to be of $k_i$ if the weighted reconstruction error, ($\|(\boldsymbol{X}-\hat{\boldsymbol{X}})*\boldsymbol{W}\|^2$) is less than that of the 98th percentile of all training data. We use linearly decreasing weights for $\boldsymbol{W}$ to emphasize the differences/similarities in $\boldsymbol{X}$ vs $\hat{\boldsymbol{X}}$ at the beginning than at the end. This is because the m-SCF of the same signal but of different SNRs have differences at the end (as shown in Fig. 5) whereas max-SCFs of different signals mainly differ at the beginning (Figs. 3a, 3b). Therefore, the reconstruction error at the beginning part is more crucial than at the end in discerning novelty.

### C. Signal/Noise Classifier (SNC)

The SNC determines whether an input is a *signal* or *noise*, i.e., when signals are absents and we are actually observing just background or thermal noise. We use Spectrum Flatness Measure (SFM) for the SNC. SFM [17] is widely used in audio applications to detect voice activity vs silence in audio data, which is similar to our task of discerning signal vs noise. SFM

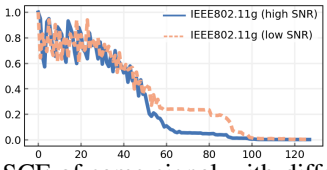


Fig. 5: m-SCF of same signal with different SNR

is defined as the ratio of the geometric mean to the arithmetic mean of the FFT magnitudes. The SFM measures whether spectrum energy is concentrated at a single frequency or it is distributed evenly over all the frequencies. In the extreme cases, SFM=0 for tones and SFM=1 for white noise. In our case, we find that, SFM $\geq$0.98 indicates noise and SFM $\leq$0.96 indicates the presence of signal. However, 0.96$<$ SFM $<$0.98 indicates that there might be noise or a very low SNR signal. Therefore, we also incorporate a ML based approach in the SNC to determine signal versus noise when computed SFM value for input FFT set is in the $(0.96, 0.98)$ range.

For this ML based approach, we use a binary classifier trained with noise and signal spectrograms (i.e., a set of FFTs). In the spectrogram of any signal, frequency bins over the time axis are correlated. In contrast, power is distributed, evenly for perfect White Gaussian Noise, or almost evenly for real-life noises, over the entire band in one FFT in noise spectrograms. Furthermore, for noise, from FFT to FFT, there is no correlation in frequency bin positions of peak powers. Hence, there is no correlation in time as well as the frequency axis in noise spectrograms . We exploit these distinguishable characteristics of noise and signal spectrograms and train a Convolutional Neural Network (CNN) for predicting signal versus noise. We choose CNN as it has been one of the most effective class of neural networks [16] for 2-dimensional data which is the case for the spectrograms.

While using only the CNN can give us the same accuracy as our combined (SFM+ML) approach, it is computationally expensive to run the CNN prediction for every input which in turn can lead to losing more samples at $Q$. This computational overhead justifies our combined approach.

### D. Final classification decision

The KSC+USD module finally combines the results of the SNC and the NDs to determine the signal class. This module generally gives precedence to the SNC result over the NDs' results. Therefore, it marks the output signal class as *noise* if SNC predicts *noise*. It marks the output signal class as *unknown* if all of the NDs predict *not-*$s_{k_i}$ and the SNC outputs *signal*. In the case of multiple NDs predicting $s_{k_i}$, the module can either mark the output as *miscellaneous* or output all of the signals classes that predicted the input as their signal with some low confidence score. The specific choice in this case will depend on the application of the classification system.

## III. Clustering Module

The *clustering* module determines the groups in unknown signals which is key for achieving the IL capability. The components of this module are summarized in Fig. 6. This module first performs several pre-clustering steps to prepare

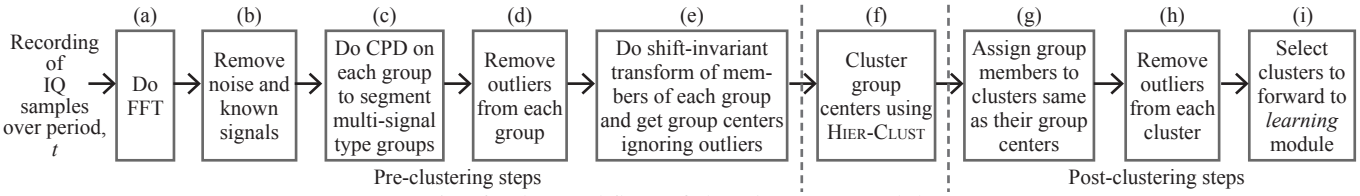


Fig. 6: The workflow of the *clustering* module

the input data for clustering, next it employs a clustering algorithm to identify the unknown signal groups, and finally it performs post-clustering steps to refine the clustering results to be forwarded to the *learning* module. We describe these pre-clustering steps, our clustering algorithm, and post-clustering steps in the following subsections.

### A. Pre-clustering Steps

Depending on the use cases and available resources, the type of input to the *clustering* module can vary. The input data can be a recording of IQ samples over some period of time for identifying unknown signal groups present in that time frame, or it can be just a recording of FFT sets that were classified as unknown by the KSC+USD module. The *clustering* module, therefore, skips some steps as needed based on input data type. Fig. 6 shows all the steps for the basic case when the input is a recording of IQ samples. The module first performs step (a) to get frequency domain data, and (b) to remove noise and known (if any) signal FFTs using the SNC and NDs of known signals of the KSC+USD module.

After step (b), the module ends up with groups of disjoint FFTs over time. Now, we exploit the fact that a single transmission will result in multiple continuous FFTs (e.g., one IEEE802.11g beacon corresponds to ∼100 continuous 128-bin FFTs) and consecutive FFTs are most likely from the same cluster. Therefore, we cluster only the group centers (i.e., mean of group members) and consider the cluster labels for group members the same as their centers. This approach greatly reduces the number of data points to work with. However, before using the group centers, we need to consider two possible cases—(I) all the FFTs in a group represent a single transmission and (II) a group comprises of multiple transmissions of different types. Case (I) is the most common one, though case (II) is not rare. To handle case (II), we employ Change Point Detection (CPD) on each of the FFT groups at step (c). CPD is a class of techniques to detect changes in a series of data [18]. Specifically, we use Pruned Exact Linear Time (PELT) [18] as the CPD method. Thus, at step (c), each of the FFT groups is divided into sub-groups based on change points identified by the PELT method. Next, step (d) removes outliers in a group in case the change points detected in step (c) do not exactly align with changes and introduce errors. A group member $g_i$ is considered as an outlier if EUCLIDEANDISTANCE($g_i$, $g_c$) $\geq m_g * \sigma$, where $g_c$ is the group center, $\sigma$ is the standard deviation of the group, $m_g$ is a multiplicative constant.

Finally, step (e) performs shift-invariant transformation on the group members so that frequency hopping signals end up in same cluster but different signals with the same frequency end up in different clusters. Specifically, we compute m-SCF using Equation 2—the same transformation used for the NDs as described in Sec. II-B. Then, we compute the group centers of the transformed group members and forward those to the clustering algorithm.

**Algorithm 2** Our hierarchical clustering algorithm

```
procedure HIER-CLUST(D, r_t, v_max, BaseAlgo)
  S ← single cluster, c_0 with all data of D
  R ← empty
  while S is not empty do
    c ← remove one cluster from S
    c_1, c_2 ← divide c in 2 clusters using BaseAlgo
    v_p ← variance of parent cluster c
    v_1, v_2 ← variances of children clusters c_1, c_2
    v_c ← (|c_1|*v_1+|c_2|*v_2)/|c|
    if v_c/v_p >= r_t and v_p <= v_max then
      R ← c
    else
      S ← c_1, c_2
  return R
```

### B. Our Clustering Algorithm

There is a plethora of clustering algorithms available in the literature. The choice of a specific clustering algorithm depends on the context and specific application at hand. In our case, we need an efficient algorithm that does not need to know the number of clusters a priori. One common approach to handle an unknown number of clusters is to try different cluster numbers with standard approaches, such as, $k$-means [19], and use an "elbow method" to determine the actual number of clusters [20]. The intuition behind the elbow method is that if we monitor the changes of "cost" of clustering with different number of clusters, after the "elbow" point, i.e., the right number of clusters, the cost will not change significantly. However, we find that this method does not work well in our setting. Indeed, settings where there is no definite elbow in the number of clusters vs cost plot have also been reported earlier [20]. In our setting, we find that a divisive hierarchical clustering method works better. There are multiple ways to achieve divisive clustering [19]. The high-level idea in all of them is to first consider all the data points to belong to the same cluster, and recursively partition the clusters until a terminating criterion has been met. It is crucial to determine a suitable terminating criterion. For our case, we find that we can use the variance of a cluster (i.e., the average squared distance of a point to its center) to determine the quality of a cluster and to decide whether or not more partitions are needed. The process of our hierarchical algorithm, HIER-CLUST, is

TABLE I: Dataset

| Name | Environment | Controlled equipment setup | Signal types |
|---|---|---|---|
| AnechB | Anechoic chamber | Bluetooth headphone, cell phone | Bluetooth |
| AnechW | | Linksys WRT54GL running DD-WRT | IEEE802.11g beacons |
| Open2.4GHz | Office room | Linksys WRT54GL, ZigBee dongles | IEEE802.11g beacons and ZigBee as well as other 2.4GHz signals from the surrounding environment from sources unknown to us |
| Open5GHz | | N/A | Campus WiFi (5GHz) |
| Open5.6GHz | | Two vehicular On-Board Units | IEEE802.11p |

outlined in Algorithm 2. HIER-CLUST first considers all the input data $D$ as a single cluster. At each step, HIER-CLUST divides a current cluster $c$ into two sub-clusters using a simple base algorithm (e.g., $k$-means). Next, it computes a weighted combination of the variances of the child clusters (denoted $v_c$). If $v_c$ is fairly close to the variance of the original cluster, $v_p$, it indicates that there is not much to gain by dividing cluster $c$. In that case, HIER-CLUST ignores the division and does not split cluster $c$. However, at this step, HIER-CLUST also checks whether or not $v_p$ is smaller than the allowed maximum variance $v_{max}$ for a final cluster. A comparative higher $v_p$ indicates that a final cluster has not been attained yet and further divisions are needed although $v_c/v_p$ indicates otherwise.

We empirically find the appropriate values for $r_t$ and $v_{max}$. The value of $v_c/v_p$, hence $r_t$ can be from range $[0, 1]$. If we choose $r_t$ closer to 1, HIER-CLUST will over-cluster, i.e., will result in more than the "true" number of clusters. Conversely, it will under-cluster when $r_t$ is close to 0. We find that $0.42 \leq r_t \leq 0.46$ consistently finds good clusters. We choose the higher value (i.e., 0.46) as our threshold as we favor over-clustering than the under-clustering. Intuitively, $r_t = 0.46$ means that we do not sub-divide a cluster unless the variance improves by 54% or more by dividing. By examining the inherent variances of different signal classes, we find that the variances are always $\leq 0.5$, hence we choose $v_{max} = 0.5$.

### C. Post-clustering Steps

After clustering the group centers, the group members (except the ones ignored in step (d)) are assigned same cluster labels as that of their group centers. The clustering results might not be perfectly accurate. Therefore, an outlier detection step is performed on the found clusters before proceeding further. The same approach as step (d) is used to find and remove the outliers in a cluster. A cluster member $c_i$ is considered as an outlier if EUCLIDEANDISTANCE($c_i$, $c_c$) $\geq m_c * \sigma$, where $c_c$ is the cluster center, $\sigma$ is the standard deviation of the cluster, $m_c$ is a multiplicative constant.

Next, we need to choose which clusters to forward to the *learning* module. Specifics of choosing clusters will vary with the application. Generally, we ignore a resultant cluster if the cluster size is below a threshold.

## IV. LEARNING MODULE

The *learning* module trains an autoencoder to be used as the ND for a found cluster or a known signal class $s_i$ from a dataset of FFT sets ($N_L * N_F$ FFTs), all belonging to $s_i$. We describe the methodologies of this module below.

First, the module randomly chooses some FFT sets of the received dataset and adds random noise to generate some low SNR examples. This aids in learning what part of the m-SCF for the current class is unique to noise variations versus what part will be affected by such variations. After adding noise, the module computes the m-SCFs of the modified dataset. This process also helps the module to generate new data when the received number of data points are low.

The next step is to choose the right model parameters for the autoencoder. An autoencoder is a combination of an encoder and decoder network. We must choose different hyperparameters (e.g., batch size, the values of $n_j$ i.e., the number of nodes in layer $j$, optimizer, activation function of nodes, loss function, etc.) as well as the the network architecture (number of layers in encoder/decoder network). In order to choose the "best" values of the model parameters, we use $k$-fold cross validation [21]. We use $(k-1)$ folds of the training data to train an autoencoder for a particular architecture and hyperparameter values. We use the remaining fold as the test dataset and compute reconstruction error. We repeat the process $k$ times and use the average of the reconstruction errors of each run as the performance for that particular setting i.e., architecture and hyperparameter values. We choose the setting that maximize the average performance on $k$ runs. By withholding some part of the data and testing with the withheld data, we check the generalization of the learned model for detecting unseen data.

Finally, we need to train the final autoencoder network based on the best performing setting. In the previous step, we use a fixed epoch number. Now, for the final model we need to find out the appropriate epoch number needed for training the model. For this, we randomly split the training data in training and validation set and monitor validation loss for each epoch. We train the autoencoder till the loss for the validation data stops improving. This process is known as *early stopping* and prevents overfitting [16].

## V. EVALUATION

*Implementation.* We implement our KSC+USD module as a standalone Qt based C++ application. In our application, we use GNU Radio [22] blocks for acquiring IQ samples from a USRP X310. However, we implement the rest of our application without using GNU Radio, for maximizing customization and overcoming the limitations in developing a real-time system with GNU Radio that we discussed in [23]. We implement the offline *clustering* and *learning* modules using Python. We use Keras [24] with Tensorflow [25] backend to train our ML models for the NDs and the SNC. We use

frugally-deep [26] to run the prediction on these Keras models in our C++ application for the KSC+USD module.

*Datasets.* We capture over-the-air signals in different bands and environments using USRP X310 and GNU Radio[3]. Specifically, we record IQ values using a sampling rate of 25 million samples/sec (Msps) and a monitoring bandwidth of 25MHz. We list our datasets in Table I; each of the datasets contains several such IQ recordings.

We first evaluate the real-time performance of our KSC+USD module in Sec. V-A1. We present the classification performance of the SNC and known signal NDs in Sections V-A2, V-A3 using the over-the-air datasets of Table I. Next, we evaluate the performance of our clustering method (HIER-CLUST) in Sec. V-B. Finally, we evaluate the IL capability of our overall system in Sec. V-C. For all of these evaluations, we use $N_F$=128 and $N_L$=8.

### A. Evaluating the KSC+USD module

As a proof-of-concept, we train NDs for ZigBee and IEEE802.11g to be used as known signal NDs in our KSC+USD module. We choose ZigBee as an example of a narrow bandwidth signal with simple modulation and, IEEE802.11g as an example of a wider signal with multi-carrier modulation. We evaluate the time and classification performance of the module below.

*1) Real-time performance:* For this evaluation, we use an USRP X310 as the RF measurement source. We run our KSC+USD module on a Dell M4800 laptop (2.8GHz, 4-core 64-bit Intel i7-4810MQ processor, 32GB RAM) running Ubuntu 18.04.1 LTS and connected to the X310 using a 1Gbps Ethernet link. We set the X310 to monitor a spectrum of bandwidth 25MHz with 25Msps sample rate.

We find the time needed for overall classification for one IQ set (from FFT and onward in Fig. 2) to be 4.6ms on average. The SNC is the key contributor to this time. Therefore, the classification time can be improved if we just use SFM for the SNC instead of using SFM+ML. With just SFM, classification takes 0.8ms on average. However, the use of just SFM can result in missing some very low SNR signals ($<$ 0dB as discussed in Sec. V-A2). The classification time can also be improved by using higher performing computing platform than ours. Therefore, depending on the computation resources available and the desired detection resolution, one can choose to run our system with just SFM based SNC or with SFM+ML based SNC.

Next, we evaluate the capability of the KSC+USD to handle the high sample rate of our RF measurement source, USRP X310. We set the values of $p_s$, $p_n$, $p_r$, $p_a$ of Algorithm 1 to 0.9, 0.1, 0.5, 0.8 respectively. GNU Radio reports overflows and gets into an unstable state when a process cannot process the data as fast as it is being sent from the USRP. We run our system for about 2 hours and GNU Radio does not report any overflows. To keep up with our 25Msps sample rate, for one classification decision with $N_F \times N_L$=128×8=1024 samples, the classification process must finish by ∼0.041ms to avoid overflows. However, as mentioned above, for our case this takes 0.8/4.2ms. Therefore, not getting overflows in our 2 hour run indicates that our SENQUEUE algorithm is helping our ML-heavy KSC+USD module to keep up with the USRP hardware.

[3]To reduce the the center frequency spike due to DC offset in samples collected using USRP, we use *tune request* and calibration files.

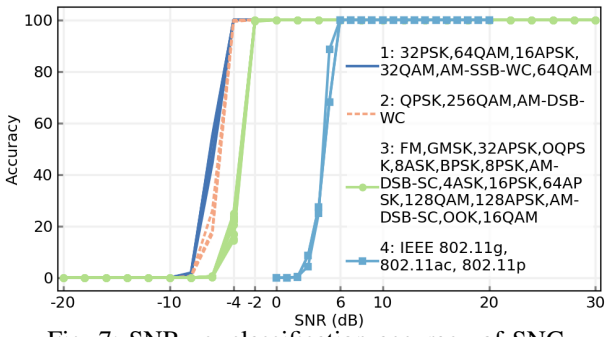


Fig. 7: SNR vs. classification accuracy of SNC

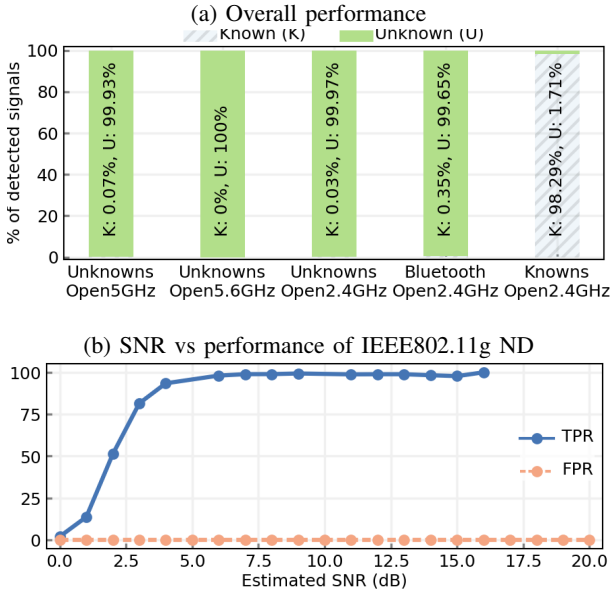


Fig. 8: Evaluation of KSC+USD with IEEE802.11g and ZigBee as known signals

*2) Classification performance of SNC:* First, we test the performance of the SNC with 257,198 labeled noise spectrograms and 430,205 labeled signal spectrograms from Open2.4GHz. For both cases, we get ∼99% accuracy.

Next, we evaluate the performance on signal spectrograms of different SNRs. As depicted in Fig. 7, the SNC can detect signals of SNR as low as -4dB (groups 1, 2), -2dB (group 3), and 6dB (group 4). We use the RadioML dataset of [2] for groups 1-3 and a custom dataset that is created by adding noise to signal spectrograms from Open2.4GHz, Open5GHz, Open5.6GHz for group 4. We separately evaluate for group 4 since differentiating signal versus noise spectrogram is comparatively harder for low SNR signals of multi-carrier modulation scheme like Orthogonal Frequency Division Multiplexing (OFDM) and wider bandwidth due to less correlation

TABLE II: Performance of HIER-CLUST vs $k$-means with elbow method (KEM)

(a) On unlabeled data

| Data from | # of data points after | | $k$ | HIER-CLUST | | KEM |
|---|---|---|---|---|---|---|
| | Step (b) | Step (e) | | $k'$ | Result | $k'$ |
| Open5GHz | 184,323 | 2,922 | 1 | 2 | C1: 1,615 WiFi, C2:1,307 WiFi with different sub-carrier activities than C1 | 4 |
| Open5.6GHz | 106,284 | 1,804 | 1 | 2 | C1: 1,802 IEEE802.11p, C2: 2 noise from some wideband signal | 4 |
| AnechB | 1,492,988 | 14,996 | 1 | 3 | C1: 9,268 Bluetooth, C2+C3: 5,728 noisy Bluetooth, most likely mixed with DC offset spikes | 5 |

(b) On labeled data from Open2.4GHz

| # of data points after | | Ratio of # of data points to cluster in largest to smallest class | $k$ | Best NMI | HIER-CLUST | | KEM | |
|---|---|---|---|---|---|---|---|---|
| Step (b) | Step (e) | | | | $k'$ | NMI | $k'$ | NMI |
| 1,326,244 | 13,251 | 20 | 5 | 0.9834 at 6 | 8 | 0.9609 | 3 | 0.8166 |
| 551,447 | 5,500 | 10 | 2 | 1.0 at 2 | 2 | 1.0 | 3 | 0.9459 |
| 501,752 | 5,000 | 5 | 4 | 0.9456 at 5 | 5 | 0.9298 | 4 | 0.8085 |

Note: $k$ = Expected # of signals, $k'$ = Found # of clusters in both Tables (a) and (b)

among FFTs in a spectrogram than that of a spectrogram of a simple modulation as in groups 1-3. Fig. 7 shows that the accuracy for these difficult scenarios is still high for signals of SNR as low as 6dB. If we use just the SFM for the SNC instead of our combined SFM+ML approach, the SNC can accurately detect signals of SNR $>$ 0dB for groups 1-3 and the performance is similar to Fig. 7 for group 4.

*3) Classification performance of known signal NDs:* For both ZigBee and IEEE802.11g NDs, we use training data from anechoic chamber recordings while we use regular indoor environment recordings for the final performance evaluation. This shows that the NDs do not necessarily need to be trained on data obtained from the deployed environment. We present our results for the following scenarios:

- *Varied environments.* Fig. 8a presents the overall performance of ZigBee and IEEE802.11g NDs. 99.93% of the Open5GHz and 100% of the Open5.6GHz signals are predicted as unknowns. Since there is no chance of encountering ZigBee and IEEE802.11g in 5 and 5.6GHz band, it shows that the NDs are very accurate in these environments. There is only a 0.03% error for non-ZigBee, non-IEEE802.11g signals from Open2.4GHz. Furthermore, there is only a 0.35% error in detecting Bluetooth as an unknown signal which indicates that the ZigBee ND performs accurately even in the problematic case of Bluetooth as mentioned earlier in Sec. II-B. The overall accuracy for known signals from Open2.4GHz is 98.29%. Individually, the accuracy of the ZigBee ND for Open2.4GHz ZigBee signals is 99.9% and the accuracy of the IEEE802.11g ND for Open2.4GHz IEEE802.11g signals is 96.68%.
- *Varied SNRs.* We create a custom dataset of different SNRs with signals from AnechW, Open5GHz, and Open5.6GHz by adding Gaussian noise. We approximate the SNRs of the original signal data as well as the custom noisy data. We present the performance of the IEEE802.11g ND for different SNR values in Fig. 8b which show that the ND classifies IEEE802.11g signals of SNR as low as 5dB with high accuracy.

### B. Evaluating the clustering module

We present the results of our clustering method, HIER-CLUST in Tables IIa, IIb. We also present the results of using $k$-means with elbow method (KEM), the common approach for clustering when the number of clusters is not known a priori. In all the cases, data is fed into the clustering methods after performing pre-clustering steps (a)-(e) of Fig. 6. For step (b), we consider the set of known signal classes, $K{=}\emptyset$. As is evident from Tables IIa, IIb, our pre-clustering steps tremendously reduce the number of data points to cluster.

For Table IIa, we use recordings from Open5GHz, Open5.6GHz and AnechB, and we expect one signal class: 5GHz campus WiFi, IEEE802.11p, and Bluetooth respectively. Though our HIER-CLUST method actually finds more than one cluster for them, we find that the resultant clusters are actually correct since there are variations in the over-the-air recordings as we describe in Table IIa. We find that KEM does not perform well in these cases.

For Table IIb, we use different combinations of labeled data from Open2.4GHz. The labeled set includes: IEEE802.11g, ZigBee, some 10MHz signals, partial signals from adjacent bands, DC offset spikes from our receiver hardware, and some narrow bandwidth signals. We vary the ratio of number of data points of the largest to smallest class to evaluate the performance of clustering in an unbalanced dataset. We use Normalized Mutual Index (NMI) [27] to measure the quality of the resultant clusters compared to the signal labels. Along with the expected cluster number, we also report the best NMIs and corresponding cluster numbers which represent the appropriate cluster numbers in the data. Table IIb shows that HIER-CLUST outperforms KEM and cluster numbers are close to those of the best NMIs.

### C. Evaluating the IL capability

We use one of the recordings from Open2.4GHz for this evaluation. This recording contains signals from two ZigBee channels, one IEEE802.11g channel, and other 2.4GHz devices in the environment. We assume that the known signal list is empty, therefore all the signals present in the environment are *unknown* for the system. After the pre-clustering steps and clustering with HIER-CLUST, we get a total of 6 clusters from the recording: one with ZigBee signals (of two channels), one with IEEE802.11g signals, one with partial signals from an adjacent band, and three others for some narrow bandwidth

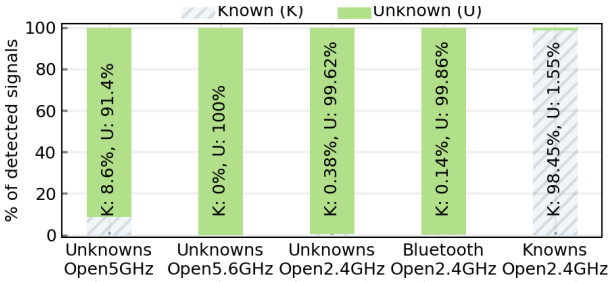

Fig. 9: Performance of NDs from clustering on same data as Fig. 8a with NDs developed based on clustering result

frequency hopping signals. To compare with the performance of NDs trained from labeled known signal data, we use our *learning* module to train NDs for the first and second cluster. Fig. 9 shows the performance of the resultant NDs on the same data as Fig. 8a, which is based on the labeled data. As can be seen from Figures 9, 8a, the NDs from clustering performs almost identically to those from the labeled data. This demonstrates that we can automatically build good quality NDs for the detected unknown signals, thus showing the efficiency of our overall incremental learning framework.

## VI. Related Work

To the best of our knowledge, we are the first to design a spectrum monitoring system with IL capability, toward the realization of effective WEC. However, in this section, we discuss existing works that are related to some of the components of our full system in addition to the related works we cite in Sec. I. There is a recent line of research that presents methodologies for identifying anomalous behavior in spectrum usage with respect to time and/or frequency [28]–[32]. For example, [31] detects anomalous scenarios like presence of chirp signals when expecting only fixed-frequency signals, and presence of infrequent signals. These past works provide a binary answer—anomaly or not. In comparison, our KSC+USD module also determines the class of the input. Conceptually, these past works can be used as individual NDs in our KSC+USD module. However, lack of discussion/results on frequency-shift invariancy and time taken for classification in these past works makes it harder to determine the feasibility of using them as the NDs of our KSC+USD module.

## VII. Conclusion

We designed a spectrum monitoring system for wireless-environment cognizance that incrementally learns about the wireless environment in which it is deployed. We have presented not just new classification methods but a full system design that is adaptive and robust against changes in deployed environment and/or computing resources. We addressed several practical challenges including supporting flexible addition/removal of new signal classes without retraining existing models, detecting shifted signals, and clustering without knowing the number of the unknown signals present a priori. A thorough evaluation of our approach demonstrated its adaptability and high accuracy with signal data from several over-the-air scenarios.

## VIII. Acknowledgement

This research made use of the resources of the High Performance Computing Center at Idaho National Laboratory, which is supported by the Office of Nuclear Energy of the U.S. Department of Energy and the Nuclear Science User Facilities under Contract No. DE-AC07-05ID14517. This research has also been supported by the National Science Foundation under Grant No. 1564287.

## References

[1] M. Aksu *et al.*, *Requirements for Spectrum Monitoring in Industrial Environments*. US Department of Commerce, NIST, 2017.
[2] T. J. OShea *et al.*, "Over-the-air deep learning based radio signal classification," *IEEE J-STSP*, vol. 12, no. 1, pp. 168–179, 2018.
[3] N. E. West *et al.*, "Deep architectures for modulation recognition," in *IEEE DySPAN*, 2017.
[4] S. Rayanchu *et al.*, "Airshark: detecting non-WiFi RF devices using commodity WiFi hardware," in *ACM SIGCOMM IMC*, 2011.
[5] T. J. OShea *et al.*, "Convolutional radio modulation recognition networks," in *EANN*. Springer, 2016, pp. 213–226.
[6] G. J. Mendis *et al.*, "Deep learning-based automated modulation classification for cognitive radio," in *IEEE ICCS 2016*, pp. 1–6.
[7] A. Fehske *et al.*, "A new approach to signal classification using spectral correlation and neural networks," in *IEEE DySPAN*, 2005.
[8] T. J. Oshea *et al.*, "Practical signal detection and classification in GNU radio," in *SDR Forum Technical Conference (SDR)*, 2007.
[9] S. S. Hong *et al.*, "DOF: a local wireless information plane," in *ACM SIGCOMM*, 2011.
[10] S. Rajendran *et al.*, "Deep learning models for wireless signal classification with distributed low-cost spectrum sensors," *IEEE TCCN*, 2018.
[11] T. J. O'Shea *et al.*, "Semi-supervised radio signal identification," in *ICACT*, 2017.
[12] Y. Gwon *et al.*, "Blind signal classification via sparse coding," in *IEEE GLOBECOM*, 2016.
[13] R. S. Roberts *et al.*, "Computationally efficient algorithms for cyclic spectral analysis," *IEEE Signal Processing Magazine*, 1991.
[14] B. Schölkopf *et al.*, "Estimating the support of a high-dimensional distribution," *Neural computation*, vol. 13, no. 7, pp. 1443–1471, 2001.
[15] N. Japkowicz, C. Myers, M. Gluck *et al.*, "A novelty detection approach to classification," in *IJCAI*, vol. 1, 1995, pp. 518–523.
[16] I. Goodfellow *et al.*, *Deep Learning*, http://www.deeplearningbook.org.
[17] J. D. Johnston, "Transform coding of audio signals using perceptual noise criteria," *IEEE J-STSP*, vol. 6, no. 2, pp. 314–323, 1988.
[18] C. Truong *et al.*, "A review of change point detection methods," *CoRR*, 2018. [Online]. Available: http://arxiv.org/abs/1801.00718
[19] J. Friedman *et al.*, *The elements of statistical learning*. Springer, 2009.
[20] D. J. Ketchen *et al.*, "The application of cluster analysis in strategic management research: an analysis and critique," *SMJ*, 1996.
[21] E. Alpaydin, *Introduction to machine learning*. MIT press, 2014.
[22] "GNU Radio," http://gnuradio.org/, 2017.
[23] C. Becker *et al.*, "Experiences with using GNU Radio for real-time wireless signal classification," in *GNU Radio Conference*, 2018.
[24] F. Chollet *et al.*, "Keras," https://keras.io, 2015.
[25] M. Abadi *et al.*, "TensorFlow," https://www.tensorflow.org/.
[26] "frugally-deep," https://github.com/Dobiasd/frugally-deep.
[27] D. Pfitzner *et al.*, "Characterization and evaluation of similarity measures for pairs of clusterings," *Springer KAIS*, 2009.
[28] T. J. O'Shea *et al.*, "Recurrent neural radio anomaly detection," *arXiv:1611.00301*, 2016.
[29] M. Walton *et al.*, "Unsupervised anomaly detection for digital radio frequency transmissions," in *ICMLA*, 2017.
[30] Q. Feng *et al.*, "Anomaly detection of spectrum in wireless communication via deep auto-encoders," *The Journal of Supercomputing*, 2017.
[31] N. Tandiya *et al.*, "Deep Predictive Coding Neural Network for RF Anomaly Detection in Wireless Networks," *arXiv:1803.06054*, 2018.
[32] S. Rajendran *et al.*, "SAIFE: Unsupervised Wireless Spectrum Anomaly Detection with Interpretable Features," in *IEEE DySPAN*, 2018.