LE3D: A Lightweight Ensemble Framework of Data Drift Detectors for Resource-Constrained Devices

Ioannis Mavromatis*, Adrian Sanchez-Mompo*, Francesco Raimondo†, James Pope§, Marcello Bullo*, Ingram Weeks*, Vijay Kumar*, Pietro Carnelli*, George Oikonomou†, Theodoros Spyridopoulos‡, and Aftab Khan*
* Bristol Research and Innovation Laboratory (BRIL), Toshiba Europe Ltd., Bristol, UK
† Department of Electrical and Electronic Engineering, University of Bristol, Bristol, UK
§ Department of Engineering Mathematics, University of Bristol, Bristol, UK
‡ School of Computer Science and Informatics, Cardiff University, Cardiff, UK
Emails: {Ioannis.Mavromatis, Aftab.Khan}@toshiba-bril.com, {F.Raimondo, James.Pope}@bristol.ac.uk

***Abstract*—Data integrity becomes paramount as the number of Internet of Things (IoT) sensor deployments increases. Sensor data can be altered by benign causes or malicious actions. Mechanisms that detect drifts and irregularities can prevent disruptions and data bias in the state of an IoT application. This paper presents LE3D, an ensemble framework of data drift estimators capable of detecting abnormal sensor behaviours. Working collaboratively with surrounding IoT devices, the type of drift (natural/abnormal) can also be identified and reported to the end-user. The proposed framework is a lightweight and unsupervised implementation able to run on resource-constrained IoT devices. Our framework is also generalisable, adapting to new sensor streams and environments with minimal online reconfiguration. We compare our method against state-of-the-art ensemble data drift detection frameworks, evaluating both the real-world detection accuracy as well as the resource utilisation of the implementation. Experimenting with real-world data and emulated drifts, we show the effectiveness of our method, which achieves up to 97% of detection accuracy while requiring minimal resources to run.**

***Index Terms*—Data Drift, IoT, Drift Detector, Resource-Constrained, Ensemble Learning**

I. Introduction

Internet of Things (IoT) is becoming synonymous with everyday computing. It has led to the deployment of billions of interconnected sensors and devices that sense, monitor, and interact with the environments [1]. IoT sensors are found in numerous domains, ranging from air pollution monitoring, farming, smart cities, and many more [2]. Since these applications rely on the fidelity of the collected data, it is fundamental to preserve the data integrity [3]. The observed data can be altered by benign causes (e.g., faulty sensors) or malicious actions (e.g., unauthorised data tampering). Both cases can disrupt or bias the states of applications and may result in widespread damage and outages. To prevent that, mechanisms for detecting drifts and irregularities are essential [4]. Based on this idea, we present Lightweight Ensemble of Data Drift Detectors (LE3D[1]), a novel lightweight data drift detection framework able to identify data irregularities in sensor streams.

Current IoT systems are characterised by the cost and complexity of their installation, favouring Low-Cost Sensors (LCSs) due to their availability and cost [5]. However, differences between sensor manufacturers, silicon, and installed environments introduce increased variability in the observed data. This leads to inconsistencies when data from different devices are compared [5]. Therefore, as described in [6], LCSs are better compared using their relative measurements and not their absolute values. Building on this idea, LE3D can detect data stream abnormalities using the observed trends in data streams. Moreover, the comparison against other devices is consolidated upon statistical trends, the goodness of fit, and the relative measurements rather than the absolute values.

Various Machine Learning (ML)-based drift detection approaches can be found in the literature, e.g., [4], [7]. In such approaches, models "learn" about the abnormal behaviour from a given dataset, or train on normal data and classify everything else as abnormal. However, when new variables are introduced, ML models tend to require retraining to improve their accuracy [8]. This incremental online learning is usually not possible on resource-constrained IoT devices. Instead, the data are usually sent to a more powerful device (e.g., a cloud server), the model is updated and is returned to the device for inference. However, this not only increases the communication cost due to the increased data exchange, but can introduce additional threats during the transit (e.g., data tampered, lost, leaked, etc.) [9]. To overcome these limitations, LE3D provides an online training mechanism that runs on-the-fly and directly on the resource-confined IoT device while requiring minimal resources.

All the above pave the way for this paper's contribution. LE3D is a framework able to identify irregularities in sensor streams. Operating as an ensemble framework, decisions are based on three estimators, these being the ADaptive WINdowing (ADWIN) [10], Page-Hinkley Test (PHT) [11], and Kolmogorov-Smirnov Windowing (KSWIN) [12] algorithms. Even though these estimators individually do not achieve ideal performance, an adaptive voting mechanism leveraging their decisions enhances the framework's accuracy. Our framework is built and optimised for real-world resource-confined IoT devices, introducing minimal overheads and dynamically adapting to new sensor types and streams. Working as a distributed

[1]Read as "leed"

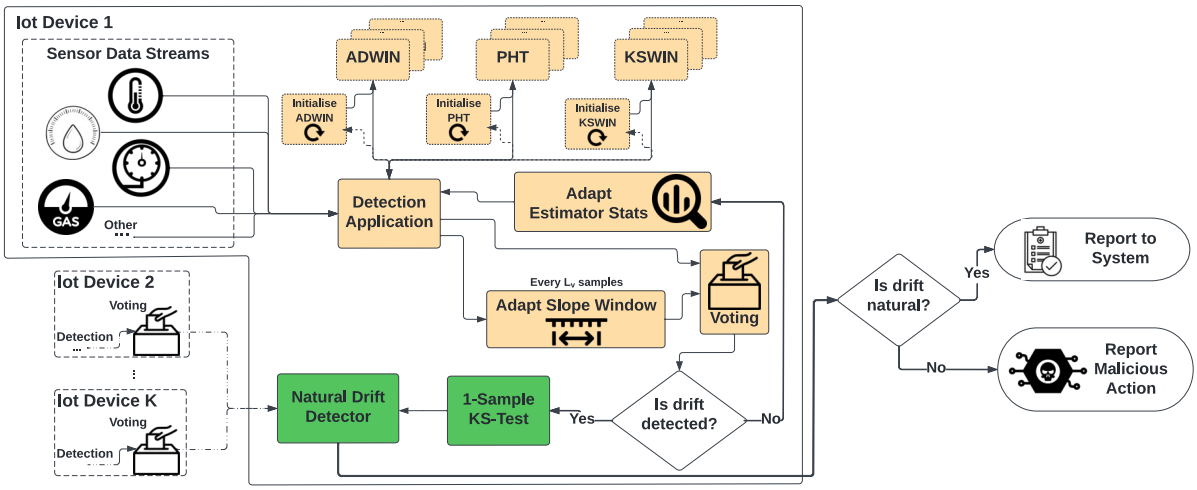

Fig. 1: LE3D: the proposed ensemble data drift detection framework. Each IoT device running a detector can process multiple sensor streams and identify abnormalities in the data (functions highlighted in orange). Later, collaboratively with other devices, the type of drift (natural/abnormal) can be classified (functions highlighted in green). The decision can be later reported to the end-user.

system later, detected drifts can be further classified as natural (e.g., a temperature is increased between day and night) or abnormal (e.g., only a single sensor reporting different than expected temperature).

The rest of the paper is structured as follows. Sec. II summarises various solutions found in the literature and describes how LE3D extends the state-of-the-art. Sec. III describes our data drift detection framework, the drift estimation mechanisms and their configuration steps. Then, our real-world implementation is briefly described in Sec. IV, where the system architecture and sensor data collected are presented. Our performance investigation can be found in Sec. V. Finally, our work concludes in Sec. VI with our final remarks.

II. Related Work

Related frameworks are found in the literature. An optimised ML approach to detecting botnets has been presented in [13]. Based on decision trees and Bayesian optimisation with Gaussian Process algorithms, this work achieves an accuracy of $> 99\%$. In [14], authors present an ensemble framework based on offline classifiers and imbalanced data that achieved $94\% - 97\%$ accuracy for the different data classes. Even though both [13] and [14] achieve high accuracy, they are based on offline learning approaches not adaptable to the fast-changing environments of an IoT ecosystem. LE3D is an online learning framework able to adapt and accommodate new data streams fed into the system in real time.

The online drift detection approach in [4] uses deep learning techniques and achieves an accuracy of ~96%. However, several thousands of training samples are required for training as well as increased training time. Our approach relies on just a few tens of samples for the detectors' initialisation without compromising the accuracy. Finally, a lightweight Performance Weighted Probability Averaging Ensemble (PWPAE) framework is presented in [15]. PWPAE is a four-party supervised ensemble data drift detector backed by adaptive weights that change in real-time. LE3D moves a step further and, using the individual decisions of each detector, can collaboratively later classify the drift as normal or abnormal. Moreover, being an unsupervised method makes it optimal for resource-constrained deployments. Due to the similarities between our framework and PWPAE, we will use it for our performance comparison.

III. Proposed Framework

Consider an indoor air quality monitoring use case as an example IoT application. Example sensor data collected are the environmental temperature, humidity, pressure, Volatile Organic Compounds (VOC), etc. All these readings can vary greatly in terms of their absolute values and standard deviation. For example the average temperature for an airconditioned room could be between 20–22 °C with a standard deviation of 1–2 °C, the average humidity could be between 40 %–50 % with a standard deviation of 4 %–6 % and the pressure can vary between 99–103 kPa with a standard deviation of 200–250 Pa [16]. When considering the data distributions of the above data, it is evident that they are not easily generalisable.

A. System Overview

LE3D framework works as a two-layer hierarchical system. Fig. 1 provides an overview of the proposed framework. Initially, each IoT device is responsible for detecting drift in individual sensor streams; at this stage the edge device cannot distinguish between natural and malicious drifts. When a new sensor stream is detected, all the estimators' statistical hyperparameters are fine-tuned using a two-step grid search. Later, for each sample received, the three estimators (ADWIN, PHT, and KSWIN) individually detect whether a drift has occured. Using an adaptive sliding window and the decisions from all estimators, a detector decides whether the drift is valid

or not (voting with equal weights for each estimator). The adaptive window and the voting can enhance the effectiveness and efficiency of the individual estimators.

As a second step, collaboratively with its surrounding neighbouring IoT devices, an IoT node can further classify a drift as natural (i.e., when a similar drift is observed in a number of devices with similar properties) or abnormal (i.e., when only a single sensor stream on a single device presents drifting behaviour) based on the outcome of voting decisions and the statistical significance of the drift. More detailed definition of natural and abnormal drifts are given in Sec. III-F. This decision is later reported to a backend system for inspection and mitigation by an end-user. In the following sections, we describe in more details the individual system components and their functionality.

B. Different Types of Data Drift

Data drift happens when the data distribution changes in a non-stationary environment. Table I summarises the key notation used in the paper, for easier comprehension. There exists an index $N \in \mathbb{N}^*$ of the sequence $\mathcal{X}$ such that all samples $x_i \in \mathcal{X}_{1:N-1} = \{x_i\}_{i=1}^{N-1}$ share the same stability properties (e.g., same probability distribution). Sequence $\mathcal{X}$ denotes a single sensor stream arriving at an IoT device. Sample with index N is the sample where a sudden or continuous drift occurs. Values can stabilise again after a number of samples $k \in \mathbb{N}^*$, and converge to the same or a new stability concept. The instances between $\mathcal{X}_N$ and $\mathcal{X}_{N+k}$ are considered to be drifting. According to the length of k, different types of drift can be described.

When a drift suddenly occurs, i.e., $k = 1$, the drift is considered as abrupt [17] when the samples before and after the drift stabilise to two different stability concepts. Gradual and incremental drifts appear when the changes occur steadily. An incremental drift occurs when the observed values change progressively between $\mathcal{X}_N$ and $\mathcal{X}_{N+k}$, moving from one stability concept to another. Gradual is the drift where a new concept gradually replaces an old one after k samples. Values in gradual drift alternate between two or more stability concepts and stabilise to one. When the instances of a stability concept appear for a short period and disappear afterwards, the drift is considered recurring. Finally, a blip drift event appears when a single sample is outside the stability concept (i.e., all samples before and after that follow the same distribution) and is considered an outlier. More information about the types of drift can be found in [17].

Our performance evaluation is based on two types of drift, i.e., abrupt and incremental. These are the most prevalent types found in time-series sensor data. Recurring drifts are a priory considered by our framework, and each drift is reported as an independent event. Gradual drift detection is more suitable for categorical data, while incremental drift suits more time-series data. Continuous time-series data can be easily encoded in categorical data. For our evaluation, we focus on raw time-series data. Integration with categorical data drift estimators is considered a future extension. Finally, blip drift events are considered outliers and are ignored.

TABLE I: Key Notations.

Notation	Explanation
$\mathcal{X}$	Sequence of sensor samples.
N	Index of last non-drifted sample.
k	Number of sensor samples drifting.
x_i	Sensor sample with index i.
$W_{\mathbb{A}}$, $W_{\mathbb{R}}$	Sliding windows (ADWIN, KSWIN).
$L_{\mathbb{A}}$, $L_{\mathbb{R}}$	Length of sliding windows $W_{\mathbb{A}}$, $W_{\mathbb{R}}$.
W_{v}	Adaptive sliding window for voting.
L_{v}	Length of voting window W_{v}.
W_{m}	Non-overlapping sliding window for trend calculations.
L_{m}	Length of trend calculation window W_{m}.
$\mu_{\mathbb{A}}$, $\mu_{\mathbb{P}}$, $\mu_{\mathbb{K}}$	Mean sample value of different estimators.
$\sigma_{\mathbb{A}}$, $\sigma_{\mathbb{P}}$, $\sigma_{\mathbb{K}}$	Std. Deviation of sample value of different estimators.
$\mu_{W_{\mathrm{m}}}$	Mean value of samples in window W_{m}.
$\theta_{W_{\mathrm{m}}}$	Mean value of samples in window W_{m}.
b	Number of samples used for initialising an estimator.
$\mathcal{E}$	List of estimators for sequence $\mathcal{X}$.
$\mathcal{V}$	Voting decisions of all estimators and collectively.
$\mathcal{D}$	List of detectors in the system.
$\mathcal{S}$	List of sensor streams fed into each detector.

C. Different Detection Algorithms

Various lightweight drift estimators are presented in the literature. Some rely on continuous data stream (e.g., a time-series of temperature), e.g., ADWIN [10], while others, e.g., Hoeffding Drift Detection Method (HDDM) [18], work with discrete values and real predictions. The nature of the sensor types considered in our system and the absence of knowledge whether there is a drift, led us to consider estimators from the first category. The three detection algorithms used are ADWIN [10], PHT [11], and KSWIN [12] (based on the Kolmogorov-Smirnov (KS) statistical test). In the future, if more estimators are required for different use-cases can be easily integrated in LE3D.

1) ADaptive WINdowing (ADWIN) algorithm: ADWIN can detect distribution changes and drifts in data that vary with time. It uses an adaptive sliding window $W_{\mathbb{A}}(n, L_{\mathbb{A}}) = \mathcal{X}_{n:n+L_{\mathbb{A}}}$ with $n, L_{\mathbb{A}} \in \mathbb{N}^*$, that is recalculated online according to the rate of change observed from the data $\mathcal{X}_{n:n+L_{\mathbb{A}}} \subset \mathcal{X}$. $W_{\mathbb{A}}$ is discretised in two sub-windows $W_{\mathbb{A}} = [W_{\mathrm{hist}}, W_{\mathrm{new}}]$ with $W_{\mathrm{hist}}(n, L_{\mathrm{hist}}) = \mathcal{X}_{n:n+L_{\mathrm{hist}}}$ and $W_{\mathrm{new}}(L_{\mathrm{hist}}, L_{\mathrm{new}}) = \mathcal{X}_{L_{\mathrm{hist}}:L_{\mathrm{hist}}+L_{\mathrm{new}}}$ such that $L_{\mathrm{hist}} + L_{\mathrm{new}} = L_{\mathbb{A}}$. When a new sample is received, ADWIN examines all possible cuts for $W_{\mathbb{A}}$ calculating the mean values μ_{hist} and μ_{new} and the absolute difference $\phi_{\mathbb{A}} = |\mu_{\mathrm{hist}} - \mu_{\mathrm{new}}|$. The optimal lengths L_{hist} and L_{new} for the two sub-windows are given comparing a threshold ϵ_{cut} against all values $\phi_{\mathbb{A}}$. It is given from $\epsilon_{\mathrm{cut}} > |\max(\phi_{\mathbb{A}})|$. Finally, ϵ_{cut} is defined as:

$$m = \frac{1}{1/L_{\mathrm{hist}} + 1/L_{\mathrm{new}}}, \quad \text{and} \quad \delta' = \frac{\delta}{W_{\mathbb{A}}} \tag{1a}$$

$$\epsilon_{\mathrm{cut}} = \sqrt{\frac{2}{m}\,\sigma_W^2\,\frac{2}{\delta'}} + \frac{2}{3m}\ln\frac{2}{\delta'} \tag{1b}$$

where σ_W^2 is the observed variance of the elements in window $W_{\mathbb{A}}$ and $\delta \in (0,1)$ is the user-defined confidence value. Once a drift is detected, all the old data samples within W_{hist} are discarded. ADWIN can effectively detect gradual drift since the sliding window can be extended to a large-sized window and identify long-term changes. Abrupt changes can again be identified with a small number of samples due to the big difference introduced in the mean values. More information about ADWIN can be found in [10].

2) Page-Hinkley Test (PHT) algorithm: PHT is a variant of the CUmulative SUM (CUSUM) test. It has optimal properties in detecting changes in the mean value of a normal process. By default, PHT is a one-sided drift detector and only detects changes when the mean increases. We extended PHT with symmetry to work as a two-sided estimator for our implementation. For every sample received, PHT recalculates the mean value $\mu_{\mathbb{P}}$ and the cumulative sum:

$$U(i) = \sum_{i=0}^{N} \left(x_i - \mu_{\mathbb{P}}^i - \frac{\beta}{2} \right) \tag{2}$$

where $\beta \in \mathbb{R}^+$ is user-defined and $\mu_{\mathbb{P}}^0 = 0$.

The result of the estimator is given by $\max(U) - U(i) \geq \lambda$, indicating an increase in the observed mean value, or by $U(i) - \min(U) \geq \lambda$, indicating a decrease in $\mu_{\mathbb{P}}^i$. $\lambda \in \mathbb{N}^*$ is a user-defined threshold. The magnitude of β describes the tolerated changes that will not raise an alarm, while λ tunes the false alarm rate. Larger λ entails fewer false-positives detections while increasing the false negatives. PHT easily identifies abrupt drifts due to the sudden change in the mean value. In contrast, incremental drift can be identified by sporadically sampling the time-series stream data.

3) Kolmogorov-Smirnov Windowing (KSWIN) algorithm: Our final estimator is KSWIN and is based on a KS statistical test. KS test is a non-parametric test, accepting one-dimensional data and operating with no assumption of the underlying data distribution. KSWIN maintains a fixed size sliding window $W_{\mathbb{K}}(n, L_{\mathbb{K}}) = \mathcal{X}_{n:n+L_{\mathbb{K}}}$ with $n, L_{\mathbb{K}} \in \mathbb{N}^*$. $W_{\mathbb{K}}$ is discretised in two sub-windows $W_{\mathbb{K}} = [W_\Omega, W_{\text{R}}]$ with $W_\Omega(n, L_\Omega) = \mathcal{X}_{n:n+L_\Omega}$ and $W_{\text{R}}(L_\Omega, L_{\text{R}}) = \mathcal{X}_{L_\Omega:L_\Omega+L_{\text{R}}}$ such that $L_\Omega + L_{\text{R}} = L_{\mathbb{K}}$. A two-sampled KS test is performed on W_{R} and W_Ω. It compares the absolute distance $dist_{W_\Omega, W_{\text{R}}}$ between two empirical cumulative data distribution, i.e., $dist_{W_\Omega, W_{\text{R}}} = \sup_x |F_{W_{\text{R}}}(x) - F_{W_\Omega}(x)|$ where $\sup_x$ is the least upper bound of the distance. $F_{(\cdot)}(x)$ represents the empirical distribution function.

The result of the estimator is given from $dist_{W_\Omega, W_{\text{R}}} > \sqrt{-\ln \alpha / L_{\text{R}}}$, where $\alpha \in (0,1)$ defines the parameter sensitivity of the test statistic, and is user-defined. Data with increased periodicity and a large window make KSWIN too sensitive and return many false positives. Relatively small L_{R}, i.e., $L_{\text{R}} \cong 30$, and an optimised α significantly improve the performance. As described in [12], KSWIN is capable of detecting gradual and abrupt drifts, but falsely classifies many samples as false positives. However, considering the criticality of an error, false positives are not as critical and can be removed in post-processing.

D. Voting Mechanism for Enhanced Detection Performance

Let $\mathcal{E} \triangleq \{1, \ldots, E\}$ with $E \in \mathbb{N}^*$ define the estimators for a sequence $\mathcal{X}$. Considering the estimators from Sec. III-C, we have $\mathcal{E} \triangleq \{1, 2, 3\}$ in our system. A single estimator is not always able to accurately detect all drifts. We enhance the performance of the framework introducing a more systemic approach. More specifically, for each x_i, our framework updates the statistics of all $\mathcal{E}$. Each $\mathcal{E}$ decides whether x_i is normal or abnormal. Operating on a per-sample fashion, it is unlikely that a single sample will be flagged as abnormal by more than one $\mathcal{E}$. Introducing an adaptive sliding window $W_{\text{v}}(n, L_{\text{v}}) = \mathcal{X}_{n:n+L_{\text{v}}}$ with $n, L_{\text{v}} \in \mathbb{N}^*$, $L_{\text{v}} > L_{\mathbb{K}}$ and a voting mechanism, we holistically examine the behaviour of the last L_{v} samples. For all $\mathcal{E}$ we maintain a sequence $v_i \in \mathcal{V}_{1:N}^{\mathcal{E}} = \{v_i\}_{i=1}^N$ with $N \in \mathbb{N}^*$, $N \geq L_{\text{v}}$, and $v_i \in \{0, 1\}$, where 0 demonstrates normal behaviour and 1 an abnormal sample.

From all $\mathcal{E}$ we collectively decide whether the last L_{v} samples present a drift. All $\mathcal{E}$ participate in the voting with equal weights and with the condition that:

$$\mathcal{V}_{W_{\text{v}}} = \begin{cases} 1, \text{ if } \sum\limits_{i \in N} \mathcal{V}^e(v_i) \geq 2, \ \forall e \in \mathcal{E} \\ 0, \text{ otherwise} \end{cases} \tag{3}$$

If the majority of $\mathcal{E}$ reported a drift, the values within W_{v} are perceived as drifted.

E. Adaptive Voting Window Length

W_{v} is adaptively modified, according to the mean value of the data received. This ensures that all types of drifts will be correctly identified. To adapt L_{v} we utilise a separate window of values, where the trend of the data is calculated. More specifically, we define a non-overlapping sliding window $W_{\text{m}}(n, L_{\text{m}}) = \mathcal{X}_{n:n+L_{\text{m}}}$ with $n, L_{\text{m}} \in \mathbb{N}^*$, grouping the last L_{m} samples. Applying a linear least-squares regression, we calculate the slope s with the best goodness of fit. We later calculate the trend of the data $\theta_{W_{\text{m}}} = \arctan s$ (measured in degrees °). When $\theta_{W_{\text{m}}} < 0$, the data trend is downwards; thus, their mean $\mu_{W_{\text{s}}}$ value is expected to decrease. Similarly, $\theta_{W_{\text{m}}} > 0$ implies an increase in $\mu_{W_{\text{s}}}$. A small drift of any type will introduce a small change in the mean $\mu_{W_{\text{s}}}$. These drifts are more difficult to be identified, and require a larger W_{m} to observe a drift. On the other hand, a sharp change is easily detected and is usually associated with an abrupt drift. Thus a smaller W_{m} can be used for that. Finally, using $\Upsilon = |\mu_{W_{\text{s}}} - \mu'_{W_{\text{s}}}| / \mu'_{W_{\text{s}}}$ we correlate $\mu_{W_{\text{s}}}$ with the mean value of the previous window $\mu'_{W_{\text{s}}}$.

For all sensor types in our system, we ran a non-linear regression analysis to model the relationship between the window size W_{v} and Υ. The exponential equation:

$$\Upsilon(x) = \zeta \exp(\eta\, x) + \gamma \tag{4}$$

with coefficients ζ, η and γ, was chosen after the nonlinear regression. This equation achieves high Root Mean Square

TABLE II: Non-Linear Regression and Coefficients.

Sensor	Measure Tests			Coefficients		
	RMSE	χ^2	r^2	ζ	η	γ
Temperature	42.599	52626.14	0.923	8.782	−5.021	1.468
Humidity	34.226	33972.54	0.954	9.641	−4.117	1.508
Pressure	28.065	22842.04	0.961	7.590	−5.132	1.829

Error (RMSE) and r-squared r^2 for all the different sensor streams; thus, it was considered the best fit for our system. In Table II we present the statistical test measures for the above equation and the coefficients found for each sensor stream. The dataset used for that is described in Sec. IV-C[2].

F. Natural and Malicious Drift

Natural is considered a drift observed in several devices with a common sensor type (e.g., temperature sensor) and characteristics (e.g., installed in the same room). On the other hand, malicious (or abnormal) is considered the drift detected only on a single sensor stream reported from a single IoT device. By sharing only the outcome of the voting mechanism, we can ensure data confidentiality and integrity (as the data never leave the device), cross-validate the results in a distributed fashion and classify the type of drift observed.

Let $\mathcal{D} \triangleq \{1, \ldots, D\}$ with $D \in \mathbb{N}^*$ define the detectors in our system. Let $\mathcal{S} \triangleq \{1, \ldots, S\}$ with $S \in \mathbb{N}^*$ define the number of sensor streams per device. In our architecture, we assume that each IoT device runs a single $\mathcal{D}$ for each sensor stream $\mathcal{S}$ fed to the device. As discussed in Sec. III-D, for each $\mathcal{S}$ there are three estimators that calculate $\mathcal{V}_{W_\mathrm{v}}$. Using the outcome of the estimators and the voting mechanism, a detector calculates an one-sample KS test using $Z(x) = x_\mathrm{ks} - \mu'$ where $x_\mathrm{ks} \in W_\mathrm{v}$ and μ' is the mean value used for initialising the estimator.

Each detector later shares $\mathcal{V}_{W_\mathrm{v}}$, Z, the sensor type, and some pre-defined metadata for the device with neighbouring nodes. Example metadata could be the room number, the zone of the building a sensor is installed, the sensor model, etc. The discovery of the neighbouring nodes is out of the scope of this work. Traditional routing protocols and Data Distribution Service (DDS) buses can provide such functionality. Using this information, devices can later decide whether the perceived drift is natural or not.

The cross-validation of the observed drift relies on the metadata exchanged, the voting decision outcome, and the result of the one-sample KS test. With regards to the metadata, sensor streams with common properties (i.e., the same metadata reported) are expected to be compared. Regarding the KS test, if the distance observed is statistically insignificant for all the received streams, we assume the drift is natural. On the other hand, if the outcome of a specific KS test presents a statistical significance compared to the rest, this sensor drifts abnormally. The detectors then report the observed behaviour to a backend system for further investigation by a system administrator. The cross-correlation of the metadata between different sensors is outside of the scope of this work.

[2]The dataset is available at https://tinyurl.com/ensembledetection

G. Estimator Initialisation

As discussed in Sec. III-C, each estimator requires a number of input hyperparameters during its initialisation. Furthermore, all estimators update their statistical models using the received samples. Therefore, to enhance the performance of our system, we introduce an initialisation phase for each detector, where the "normal" behaviour is established.

For the initialisation, either the first b received samples can be used (assuming that no drift will be added to the system during this course) or a "trusted" dataset that accompanies the detector. As described from the Central Limit Theorem (CLT), the distribution of sample means approximate a normal distribution as the sample size gets larger, regardless of the population's distribution. Based on CLT, and as described in [19], this sample size can be between 30 to 50. Our system considers the first 100 samples for the initialisation. Based on them, we calculate the μ' and the variance σ^2 and run an exploratory investigation to find the best initialisation parameters for each estimator.

Our exploration is based on two grid searches. The first one narrows the search space returning an estimated value for the hyperparameters. The second micro-grid search fine-tunes all hyperparameters by evaluating various parameters within a smaller, more precise exploration space. Given the simplicity of the estimators, the notion of "error score" is not introduced in our optimisation. Instead, a pre-defined logic is hardcoded in the system. For example, higher δ values for ADWIN increase the sensitivity of the estimator. Our chosen δ is a value that does not return candidate drift within a "normal" sample distribution. Following such an approach, our estimator becomes sensitive enough to detect drift when introduced effectively. Similarly, for PHT, the lowest values for λ and β are preferred. For KSWIN, lower α improves the estimator's confidence, while a value of $L_\Omega \approxeq 30$ is preferred for the statistical window. Fixing $L_\Omega = 30$ we later fine-tune L_R and α as before. Finally, the precision of the grid search (defining the grid steps) can be fine-tuned based on the available hardware and the time constraints of each use case.

IV. LE3D: Framework Implementation

A. System Architecture and Implementation

We assume a standard three-tier architecture: cloud, edge and endpoint. At the top, a central "cloud server" component is responsible for the application and service deployment to the edge/endpoint tiers and for visualising the results. The "edge" tier consists of several resource-constrained IoT devices, acting as the "edge" nodes deployed close to the endpoints or sensors. The "edge" tier accommodates the data bus and message protocols, the detection and voting mechanisms and is responsible for sending the natural or abnormal drift decisions to the cloud. Finally, our endpoints collect and disseminate the sensor data to the edge and incorporate no intelligence.

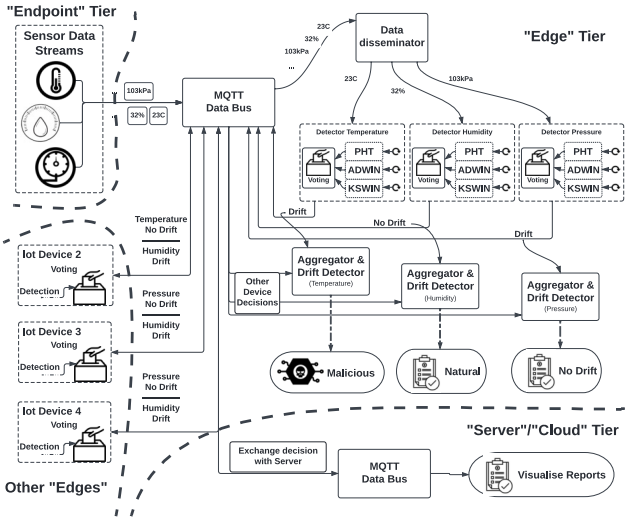

Fig. 2: A diagram visualising the different system components and the interactions between them.

Our implemented framework consists of five components, i.e., detectors, aggregators, streamers, emulators and data/message buses. Initially, a detection application operates on each "edge" IoT node, maintaining multiple detectors. Each sensor stream is assigned its own detector responsible for updating and maintaining the estimators, handling the voting and adapting W_{v}. Later, an aggregation application also runs on each "edge" IoT node. This aggregation application collects the results and metadata from the current node and neighbouring "edge" nodes and decides whether the drift detected is natural or abnormal. The decision is later sent to the "cloud" tier for visualisation. All our interactions and messages are exchanged via an MQTT data bus running at the "edge" tier and a set of pre-defined topics.

As access to multiple drifting real-world endpoints was infeasible, we developed a set of supporting tools for our performance evaluation, i.e., streaming and emulator applications. The streaming application streams "real-world" data from a pre-existing dataset (CSV files). An emulator generates "realistic" emulated data streams and drifts on demand. The data generated follow a given distribution calculated from a real-world dataset (Sec. IV-C). The emulators expose various RestAPI interfaces for controlling the drifts introduced. Multiple streamers and emulators were spun up during our evaluation and drifts were introduced at random intervals. A high-level representation of the framework and the interactions can be seen in Fig. 2.

Our framework[3] was implemented in Python 3.9.12. The estimators' functionality is based on River online/streaming ML package [20]. We overrode various functions to adapt them to our framework and achieve the desired functionality introduced earlier (Sec. III). The linear regression, the grid search and the statistical calculations are based on SciPy and NumPy open-source libraries. The MQTT messaging is based on the Paho MQTT client implementation provided by Eclipse [21]. The non-linear regression was performed using the ZunZun curve fitting library [22]. The rest of the framework was developed from scratch, and the various components were integrated into that. Our codebase was built on a Raspberry Pi (RPi) Compute Module 3b+ [23], with a BCM2837B0 Cortex-A53 64-bit 1.2 GHz System-on-a-Chip (SoC) and 1 GB of RAM. This RPi was chosen as a representative resource-constrained IoT device.

[3]The code will be uploaded to a public GitHub repository in the camera-ready paper. Currently accessible from: https://tinyurl.com/ensembledetection

B. System Scalability

Even though evaluating the scalability of our solution is out of the scope of this work, in this section, we address some ideas considered during our implementation phase. First, as lightweight operation is essential for real-world frameworks, the statistical models and software libraries considered were validated for their resource utilisation before their integration into the system, always considering the accuracy of the predictions as well. Second, in terms of response and execution time, the requests that can be served per second, and memory usage, we quantified the system's performance in Sec. V-B. Third, regarding network usage, reducing the exchange of sensor data and exchanging only the voting decisions not only preserves the data integrity and confidentiality but can also reduce the exchange of network data. Fourth, in terms of the horizontal scaling of the system, as discussed in Sec. III-F, the voting decisions should be compared and exchanged with close proximity or look-alike neighbours (with common features). Even though not considered at this stage, a neighbour discovery mechanism can optimise the exchange of the voting decisions even for large-scale deployments. Finally, in terms of the number of sensor streams supported per detector, our multithreaded implementation makes the operating system's kernel and the number of threads supported there the only limiting factor.

C. IoT Endpoints and Sensors

The data used for the initial model was collected at the Communication Systems & Networks Laboratory at the University of Bristol between 15–22 February 2022. The data collection effort went through an ethics approval process. A total of eight IoT endpoint devices (Fig. 3) were deployed in an office environment, all equipped with commercial off-the-shelf sensors and a wireless micro-controller. The devices were spread around the lab and office space areas, with roughly 10–12 researchers usually present during normal working hours. All devices were USB powered and were equipped with:

1) A Nordic nRF52480 Bluetooth SoC.
2) A Light Sensor (ISL29125): Collects both colour and light intensity values.
3) An Accelerometer Sensor (MMA8452Q).
4) An Environmental Sensor (BME680): Collects temperature, humidity, pressure, & gas (VOC/CO_2) values.

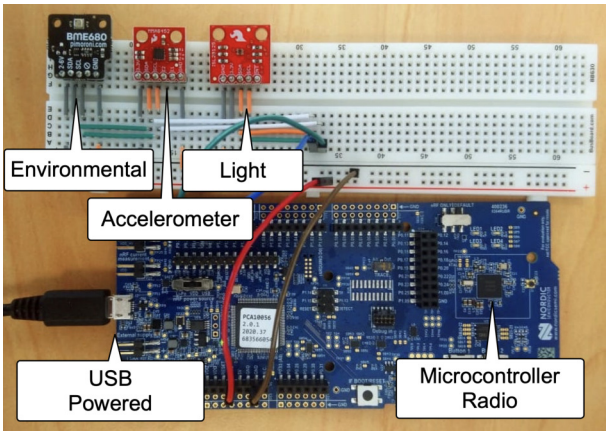

Fig. 3: Our developed IoT Endpoint and Sensors boards.

The endpoints collected sensor samples every 10 s, the data were stored on a nearby root controller/desktop and were exchanged via an IEEE 802.15.4 Time Synchronised Channel Hopping (TSCH) mesh network. This dataset was used for our real-world performance investigation to generate real-world "streams" of data, construct realistic emulators for demonstrating drifting behaviour and for the initialisation of the estimators (as described in Secs. III-G and IV-A).

V. Performance Evaluation

Our performance investigation is two-fold. Initially, we compare LE3D against PWPAE [15] and the performance of each individual estimator introduced in LE3D. PWPAE was chosen due to its similarity to our framework, the high accuracy it achieves, and its lightweight nature. Later, we conduct a detailed performance profiling to measure LE3D execution time and the perceived resource utilisation. PWPAE is an ensemble drift detection based on four classifiers, these being the Streaming Random Patches (SRP) classifier (using either ADWIN or Drift Detection Method (DDM) as its base estimators), and Adaptive Random Forest (ARF) classifier (again using either ADWIN or DDM as its base estimators). The four classifiers are, by default, ensemble methods, using multiple instances of the same estimator in the background. Following the authors' recommendation, PWPAE classifiers were configured with three estimators for our performance comparison.

Our evaluation is based on both real-world and emulated data. We conducted experiments for three different sensor types, i.e., temperature, pressure and humidity. The data streams are generated on a desktop PC and fed into the detectors. We generate a data sequence of 40 timeslots for each emulated stream. Each timeslot has a random length $l \in [500, 1500]$. This generates approximately ~40k samples per experiment. Each timeslot is assigned a type with equal probabilities, i.e., "normal", "incremental", and "abrupt". The first timeslot is always "normal", as it corresponds to the time frame that the estimators are initialised. The averaged outcome of 1000 experiments will be presented later in this section.

TABLE III: Hyperparameters for all Estimators and Sensors.

Sensor	ADWIN	PHT	KSWIN	Stats
Temperature	δ=0.44	β=0.095 λ=480	α=0.001 L_R=300 L_Ω=30	μ'=20.32° σ^2=1.178
Humidity	δ=0.44	β=0.095 λ=560	α=0.001 L_R=300 L_Ω=30	μ'=30.14% σ^2=0.966
Pressure	δ=0.34	β=2.9 λ=29000	α=0.0001 L_R=300 L_Ω=30	μ'=102.4kPa σ^2=224.52

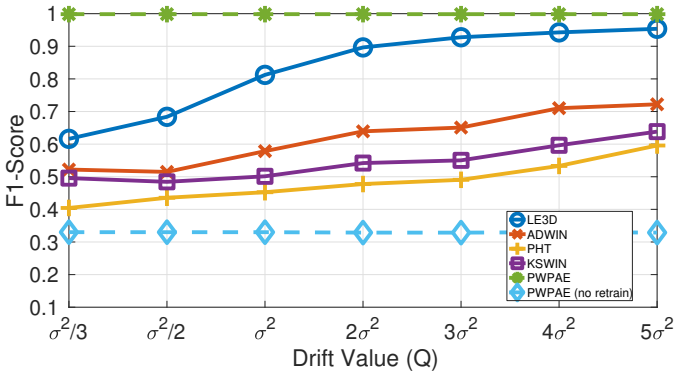

Fig. 4: F1-score for humidity and all drift values.

At the beginning of each experiment with define $q \in \{\sigma^2/3, \sigma^2/2, \sigma^2, 2\,\sigma^2, 3\,\sigma^2, 4\,\sigma^2, 5\,\sigma^2\}$ where σ^2 is the variance of each sensor type (Table III). q is fixed for the rest of the experiment. For each timeslot, a value is drown from a distribution that dictates the drift per timeslot. More specifically, for an abrupt drift, samples are drawn from $x \in \mathcal{N}(\mu' + Q, \sigma^2)$ where $Q \in [-q, q]$. For an incremental drift, we calculate the step z as $z = Q/l$, and the samples are given from the equation $y(x) = z\,x + x$, where $x \in \mathcal{N}(\mu', \sigma^2)$.

A. Experimental Evaluation

As discussed, LE3D execution starts with initialising the estimators' hyperparameters. In Table III we present the grid search optimised hyperparameters (as described in Sec. III-G), and the μ' and σ^2 for all sensors. Investigating the PWPAE codebase, we pinpointed that after the initial training, authors also continue the training during the inference phase. Their model is updated on a per sample basis using the expected labels and not the predicted ones (as is usually the case in supervised learning methods). Such an implementation is rather infeasible for a real-world system as the expected labels will never be available for online training. Thus for our evaluation, we assessed PWPAE in two different scenarios: 1) with the online training enabled, 2) with the online training disabled.

Our results are presented in Figs. 4 - 6. F1-score was chosen as a good metric due to the imbalanced nature of our data (i.e., more drifts are generated than normal data). Our evalution show that LE3D performs very well for all sensor types, can provide a generalisable approach, and achieves high F1-scores

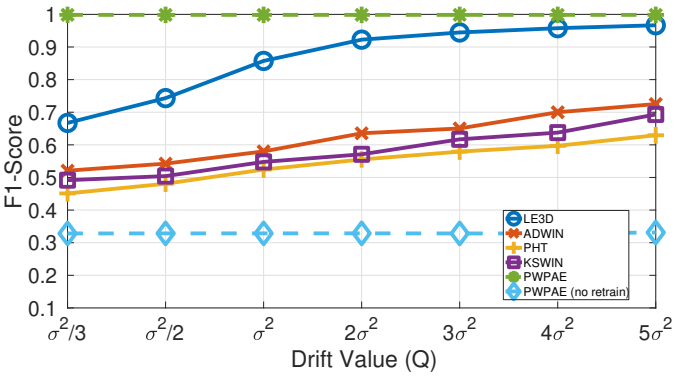

Fig. 5: F1-score for temperature and all drift values.

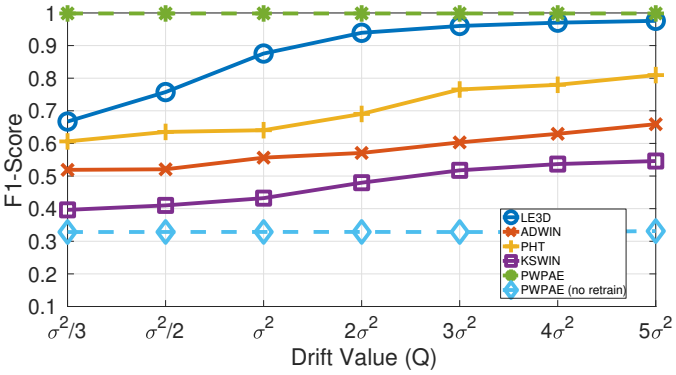

Fig. 6: F1-score for pressure and all drift values.

up to 97%. Even for smaller drifts (when $q \geq \sigma^2$), the F1-score slightly decreases but is always above 83%.

Considering PWPAE, we see that when the training is enabled, the model always performs great, achieving an F1-score of 99.8%. PWPAE requires a very small number of labelled samples to "learn" that a drift is introduced within a timeslot. These results, even though on paper, are great, as discussed above, are infeasible to be achieved in a real-world system where the expected labels for each sample are unavailable. This is apparent when the online training is disabled, and PWPAE's F1-score is reduced to ~32%. During that scenario PWPAE always reports a "normal" behaviour. Investigating further the classifiers proposed from PWPAE, it was identified that they benefit from feature-rich datasets and do not perform well with time-series raw sensor data (like in our case). Of course, even though post-processing the data is feasible, it will introduce more overhead in the system.

Finally, considering the individual estimators, we see that PHT performs better for pressure streams. This is because the observed $\mu_{\mathbb{P}}$ is a very prominent component in the PHT algorithm. When the absolute value of a sensor steams is not modified significantly (e.g., temperature and humidity), KSWIN and ADWIN outperform PHT. However, as observed, all estimators independently perform worse than our ensemble framework. From the results, it is evident that the collective decision taken using our voting mechanism can enhance the detection accuracy. From the above, and considering the requirements for real-world deployments, LE3D significantly outperforms individual estimators and other state-of-the-art solutions when only raw data are available.

TABLE IV: Execution Time Comparison against PWPAE [15].

Test	Detector Initialisation	Sample processing
PWPAE	~4.5 ms	6500–8500 µs
PWPAE (no training)		2000–2800 µs
Our approach	~3 ms	150–600 µs

TABLE V: Performance Profiling on a Raspberry Pi 3B+.

Test	Incoming Sample Rate	CPU	RAM	System Load
Idle RPi	-	0.1%	1.4%	0.0025
Just MQTT	-	0.3%	1.4%	0.005
Ensemble f/w & MQTT	-	0.4%	7.8%	0.0075
Sampling Rate	1 Hz	0.45%	7.8%	0.0075
	10 Hz	0.7%	7.8%	0.01
	100 Hz	1.2%	7.9%	0.015
	500 Hz	3.9%	8.1%	0.035
	1000 Hz	7.6%	8.7%	0.08
	2000 Hz	9.85%	9.8%	0.12
	4000 Hz	19.5%	11.4%	0.24

B. Execution Time and Resource Consumption Evaluation

We profiled LE3D and PWPAE codes, measuring the execution time (Table IV) and the resource utilisation (Table V). Both experiments were conducted on an RPi 3b+. As discussed in Sec. IV-A, the edge nodes host an MQTT broker and the detection frameworks.

As execution time, we quantify: 1) the time required for initialising a detector (e.g., when a new sensor stream is fed into the frameworks), and 2) the time required to process an individual sample. As resource utilisation, we measured the CPU and RAM utilisation (using *ps* command on Unix, sampled every 0.1 s), and the average system load (using *uptime* at the end of an experiment). The values are normalised to the four-core architecture of the RPi 3b+ (i.e., divided by four). Each experiment lasted for 20 min capturing the code profile between the 5 to 20 min range.

Table IV summarises the execution time in comparison with PWPAE. PWPAE was evaluated for both scenarios (with and without online training). When compared we see that LE3D requires ~3 ms for initialising a detector, whereas PWPAE requires ~4.5 ms. This is not a big difference considering that a detector is initialised only once when a new sensor stream is received. However, the individual sample processing time presents a significant difference. Our approach requires about 20 times less time (comparing the average values) when compared to PWPAE with online training and less than eight times when the training is disabled. As seen, the performance of PWPAE degrades significantly when the training is disabled. Considering the case where the training is enabled, it can be approximated that PWPAE can process ~150 $^{\text{samples}}/_{\text{s}}$.

LE3D is later evaluated with up to $4000\ ^{\text{samples}}/_{\text{s}}$ (Table V), a value significantly greater than PWPAE. As seen, our framework is lightweight and achieves low CPU, i.e., $< 10\%$ and $< 20\%$ for 2000 Hz and 4000 Hz sampling rates, and RAM utilisation, i.e., always less than $< 11.5\%$, even when the number of samples increases. In our framework, historic sensor samples are kept after processing, increasing the RAM usage slightly. This can be regulated by discarding historical data after expiration or when a predefined queue is full. Even though the system is still not saturated, we can see that at ~$2000\ ^{\text{samples}}/_{\text{s}}$, we approach the limits of our implementation, if a real-time operation is required. Investigating further, we concluded that the single-threaded implementation of the Eclipse Paho MQTT library in Python creates a bottleneck on the number of samples one can subscribe to per second. Workarounds will be considered as future extensions of the system. This could either be replacing the Paho MQTT library with another implementation, using a different messaging protocol, or subscribing to batches of data, keeping them in a separate queue, and processing them asynchronously from when they arrive.

VI. Conclusion

This paper presents LE3D; a lightweight ensemble data drift detection framework for resource-confined IoT environments. Working in a distributed two-tier hierarchical fashion can detect irregularities in the received sensor streams and classify them as natural or abnormal. Our framework is generalisable and performs with high accuracy for different sensor types and data streams, achieving up to 97% in F1-score. Its strength relies on the system dynamically adapting to new sensor streams without any accuracy reductions, while the collective decisions taken from different devices can provide more in-depth knowledge on the types of drift observed. Moreover, its distributed nature and the fact that the data never leave the device preserves the data confidentiality and integrity. Compared to other state-of-the-art solutions, we can see that the detection accuracy is almost on par with more resource-heavy methods. Furthermore, conducting an extensive performance profiling of our proposed framework, we demonstrated its lightweight operation in a resource-constrained IoT device. The detection accuracy, the minimal overhead introduced from the implementation, and the framework's scalability make it a great candidate for data drift detection frameworks for real-world IoT sensor applications.

Acknowledgment

This work was supported in part by Toshiba Europe Ltd. and in part by the SYNERGIA project (grant no. 53707, UK Research and Innovation, Innovate UK).

References

[1] P. Fraga-Lamas, T. M. Fernández-Caramés, M. Suárez-Albela, L. Castedo, and M. González-López, "A Review on Internet of Things for Defence and Public Safety," *Sensors*, vol. 16, Oct. 2016.

[2] H. Arasteh, V. Hosseinnezhad, V. Loia, A. Tommasetti, O. Troisi, M. Shafie-khah, and P. Siano, "IoT-based Smart Cities: A Survey," in *Proc. of IEEE EEEIC 2016*, Jun. 2016, pp. 1–6.

[3] M. N. Aman, B. Sikdar, K. C. Chua, and A. Ali, "Low Power Data Integrity in IoT Systems," *IEEE Internet Things J.*, vol. 5, no. 4, May 2018.

[4] O. A. Wahab, "Intrusion Detection in the IoT under Data and Concept Drifts: Online Deep Learning Approach," *IEEE Internet Things J.*, pp. 1–1, Apr. 2022.

[5] P. Ferrer-Cid, J. M. Barcelo-Ordinas, J. Garcia-Vidal, A. Ripoll, and M. Viana, "A Comparative Study of Calibration Methods for Low-Cost Ozone Sensors in IoT Platforms," *IEEE Internet Things J.*, vol. 6, no. 6, pp. 9563–9571, Jul. 2019.

[6] F. M. J. Bulot, S. J. Johnston, P. J. Basford, N. H. C. Easton, M. Apetroaie-Cristea, G. L. Foster, A. K. R. Morris, S. J. Cox., and M. Loxham, "Long-term Field Comparison of Multiple Low-cost Particulate Matter Sensors in an Outdoor Urban Environment," *Scientific Reports*, vol. 9, no. 1, Oct. 2019.

[7] B. Friedrich, T. Sawabe, and A. Hein, "Unsupervised Statistical Concept Drift Detection for Behaviour Abnormality Detection," *Applied Intelligence*, vol. 58, no. 3, pp. 509–523, May 2022.

[8] D. Nallaperuma, R. Nawaratne, T. Bandaragoda, A. Adikari, S. Nguyen, T. Kempitiya, D. De Silva, D. Alahakoon, and D. Pothuhera, "Online Incremental Machine Learning Platform for Big Data-Driven Smart Traffic Management," *IEEE Trans. Intell. Transp. Syst.*, vol. 20, no. 12, Jul. 2019.

[9] M. Abomhara and G. M. Køien, "Security and Privacy in the Internet of Things: Current Status and Open Issues," in *Proc. of Int. Conf. PRISMS 2014*, Dec. 2014, pp. 1–8.

[10] A. Bifet and R. Gavaldà, "Learning from Time-Changing Data with Adaptive Windowing," in *Proc. of Int. Conf. SDM 2007*, Apr. 2007.

[11] D. V. Hinkley, "Inference about the Change-point from Cumulative Sum Tests," *Biometrika*, vol. 58, no. 3, pp. 509–523, Dec. 1971.

[12] C. Raab, M. Heusinger, and F.-M. Schleif, "Reactive Soft Prototype Computing for Concept Drift Streams," *Neurocomputing*, vol. 416, Nov. 2020.

[13] M. Injadat, A. Moubayed, and A. Shami, "Detecting Botnet Attacks in IoT Environments: An Optimized Machine Learning Approach," in *Proc. of ICM 2020*, Dec. 2022, p. 1–4.

[14] C.-C. Lin, D.-J. Deng, C.-H. Kuo, and L. Chen, "Concept Drift Detection and Adaption in Big Imbalance Industrial IoT Data Using an Ensemble Learning Method of Offline Classifiers," *IEEE Access*, vol. 7, Apr. 2019.

[15] L. Yang, D. M. Manias, and A. Shami, "PWPAE: An Ensemble Framework for Concept Drift Adaptation in IoT Data Streams," in *Proc. of IEEE GLOBECOM 2021*, Dec. 2021, pp. 01–06.

[16] H. Ritchie and M. Roser, "Indoor Air Pollution," *Our World in Data*, 2013, https://ourworldindata.org/indoor-air-pollution.

[17] A. S. Iwashita and J. P. Papa, "An Overview on Concept Drift Learning," *IEEE Access*, vol. 7, pp. 1532–1547, Dec. 2019.

[18] I. Frías-Blanco, J. d. Campo-Ávila, G. Ramos-Jiménez, R. Morales-Bueno, A. Ortiz-Díaz, and Y. Caballero-Mota, "Online and Non-Parametric Drift Detection Methods Based on Hoeffding's Bounds," *IEEE Trans. Knowl. Data Eng.*, vol. 27, no. 3, pp. 810–823, Aug. 2015.

[19] S. M. Ross, "Chapter 7 - Distributions of Sampling Statistics," in *Introductory Statistics*, 4th ed., S. M. Ross, Ed. Oxford: Academic Press, Jan. 2017, pp. 297–328.

[20] J. Montiel, M. Halford, S. M. Mastelini, G. Bolmier, R. Sourty, R. Vaysse, A. Zouitine, H. M. Gomes, J. Read, T. Abdessalem, and A. Bifet, "River: Machine Learning for Streaming Data in Python," *arXiv:2012.04740 [cs.LG]*, Dec. 2020.

[21] "Eclipse Paho MQTT Client]," https://www.eclipse.org/paho/index.php?page=clients/python/index.php, 2022, Accessed: 2022-8-9.

[22] J. R. Phillips, "ZunZun Curve Fitting Library [Online]," https://bitbucket.org/zunzuncode/pyeq3/src/master/, 2021, Accessed: 2022-8-9.

[23] RaspberryPi, "Compute Module 3b+ [Online]," https://www.raspberrypi.org/products/compute-module-3-plus/, Accessed: 2022-8-9.