# Customizing Pre-Processing Algorithms for Streaming Sensor Data on Embedded Networked Devices

Dragoș Lazea, Tudor Cioara, Anca Hangan
{firstname.lastname}@cs.utcluj.ro
Technical University of Cluj-Napoca, Romania

Zsolt István
zsolt.istvan@cs.tu-darmstadt.de
Systems Group, TU Darmstadt, Germany

## Abstract

As Internet of Things (IoT) devices generate vast amounts of data, and most edge intelligence-enabled Machine Learning (ML) models deployed in the IoT infrastructures require cleaned and pre-processed data, we propose implementing data pre-processing techniques directly on IoT devices such as smart sensors and $\mu$controllers, using the already available computing resources. This enables performing procedures like data cleaning, outlier detection and data aggregation to be performed in a streaming fashion in a predictable way and meeting real-time requirements. This offers support for more complex data analysis that happens on the edge nodes which are placed further from the devices. In this work, we propose a guideline of tailoring data pre-processing algorithms for operating on a limited window of most recent sensor readings and for allowing processing data at a high rate to guarantee real-time execution. We illustrate the tailoring process using two simple statistical outlier detection techniques and evaluate the implementations in a real online setting.

## CCS Concepts

• **Computer systems organization** → **Embedded software**; • **General and reference** → *Measurement*.

## Keywords

Anomaly Detection, Internet of Things (IoT), Streaming Data Processing

## 1 Introduction

Networked sensors and IoT devices are increasingly deployed in various domains and enable innovative applications but they also face several challenges in terms of providing online and real-time behavior using severely limited computational and storage resources [11]. Furthermore, the high amount of data which is continuously generated and processed by such devices and sensors can easily conduct to a network bottleneck [2]. This data often contains outliers generated by several causes such as device malfunctions, malicious attacks, extreme or unexpected events and harsh environmental conditions in which the devices are deployed. These anomalies should be identified and filtered out as soon as they are generated in order to ensure both the expected behavior of the system and the decrease of the volume of data that moves across the network [8].

Therefore, implementing efficient online anomaly detection procedures closer to the source of the data is of major importance in modern sensor infrastructures in order to provide the desired functionalities and meet the real-time constraints without creating catastrophic network bottlenecks. However, deploying such algorithms on low-power existing processors that are usually used at collecting and relaying sensor measurements is a challenging

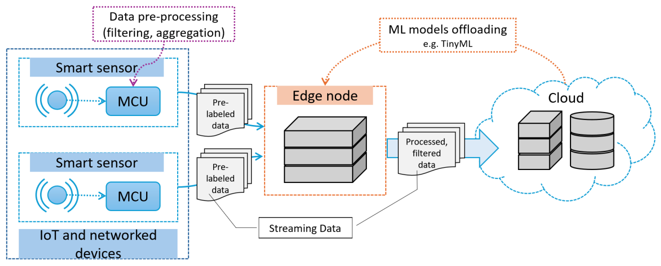

**Figure 1: A data pre-processing step performed at the level of smart sensors offers support for more complex Machine Learning anomaly detection models deployed on edge servers or in the cloud**

opportunity which faces several issues and involves satisfying two key conditions. First of all, anomaly detection models designed for such resource-constrained devices should be able to operate on reduced amounts of data while providing significant results [4]. In addition, the algorithms should be themselves simple in terms of performed operations and amount of memory needed to store the instructions. This is why computationally expensive Machine Learning (ML) anomaly models are not suitable in this context. Secondly, such implementations should provide online anomaly detection, by identifying and filtering outliers as the data is collected. Moreover, the majority of sensor-based systems are time-sensitive. Thus, low latency has to be highly ensured [1, 2].

In this work, we show how lightweight anomaly detection algorithms can be tailored to fit on low-power and resource constrained devices to ensure the quality of sensor data and to perform automatic data pre-labeling in real-time before feeding it into more complex edge embedded ML models, as depicted in Figure 1. Our proposal aims at handling the streaming data by processing a sub-window of samples and ensuring low execution time to allow increasing the sensors sampling rate without exceeding the real-time constraints. We implement alternatives of two statistical anomaly detection techniques on a Microcontroller Unit (MCU) to guarantee deterministic execution and predictable behavior and point out the optimization opportunities that result in lower response time. To this aim, we look at both data structures and algorithm particularities, also exploiting the previously computed results. We comparatively evaluate our implementations regarding both the execution time and the memory usage, discussing how many sensor data samples can be stored on such a device, depending on the algorithm implementation with the focus on data structure that we use at storing the most recent sensor readings. In summary, our contributions are the following:

- We propose a guideline for tailoring anomaly detection algorithms for online execution on resource limited devices with the focus on the data structures and reusing previous computations in order to provide a step of data pre-processing at the level of smart sensors.

- We exemplify the tailoring process on two specific outlier detection techniques and compare the proposed implementations with the naive versions.
- We perform an experimental evaluation of the implementations in an online setting using a $\mu$controller unit and a temperature sensor and also estimate how many sensor data samples can be stored at once on the device for each particular implementation to ensure that the results are significant in the context of filtering anomalies out.

## 2 Background and Related Work

Detecting anomalies in sensor networks refers to identifying measurements that notably diverge from the typical data patterns. These abnormal values can stem from various sources such as noise and errors, abnormal events, or malicious attacks on the network [11]. In this research we focus on point anomalies or outliers which are observations placed far from the mean of values taken into account [7]. We aim at filtering outliers out or labeling them as abnormal values to ensure a first step of sensor data pre-processing.

The main challenges of detecting anomalies in sensor infrastructures which emerge from the streaming nature of sensor data and the fast increase in the number of networked sensors and IoT devices were addressed in several research works [3, 8, 11]. The authors of [11] and [3] identify resource constraints, high communication costs, dynamic network topology, distributed streaming data and large-scale deployments as the major difficulties encountered when looking for outliers in sensor data, while energy consumption and devices heterogeneity are also pointed out in [8] as significant issues which have to be taken into account during the anomaly detection process within IoT and sensor-based networks.

Anomaly detection methods for IoT and embedded sensors data are described and classified by many related works [1–3, 9, 11]. Even there is no standardized taxonomy for outlier detection techniques used to identify abnormal values within the sensor data, five main classes of methods can be easily identified: statistical methods, unsupervised methods, supervised methods, semi-supervised methods and deep learning-based methods [2]. However, low availability of pre-classified or pre-labeled sensor data makes supervised techniques difficult to deploy [11]. Thus, a step of pre-processing or labeling observations at the level of sensor or IoT device would offer the necessary support for employing supervised methods to detect anomalous observations.

Furthermore, as IoT sensors and networked devices generate increasingly high amounts of data, distilling anomaly detection processes at the edge of the network and hybrid anomaly detection becomes the focus of many research papers [1, 3–6]. Deploying anomaly detection procedures closer to the source of the data in IoT and sensor-based networks is highlighted in [4] as a solution of reducing the bottleneck of transferring huge amount of sensor data to the cloud. Thus, the authors propose filtering data at the edge of the network by removing outliers to reduce the volume of data sent to the cloud. The importance of sensor data cleaning and filtering before it is fed to more computationally intensive algorithms and ML models deployed on more powerful devices is also highlighted by the authors of [3]. The ability to identify outliers in an online fashion as the data is generated is stated by the authors of [1] as the fundamental requirement of detecting abnormal values in sensor networks. Additionally, the edge intelligence-powered deploying of ML anomaly detection models models on novel $\mu$controllers featuring powerful computational and storage capabilities is explored by the authors in [9] as an emerging opportunity of TinyML.

However, there are numerous recent studies which state that deploying a single model to detect anomalies in sensor data at the edge does not prevent sending erroneous data to the cloud [5, 6]. Therefore, hierarchical and hybrid anomaly detection models are getting more attention in the context of sensor data. A hierarchical solution of deploying anomaly detection models of different complexities on different hierarchical layers of IoT-Edge-Cloud infrastructures is proposed in [5] to provide multi-layer IoT data filtering. The authors of [6] also propose a three-step hybrid approach of hierarchical anonaly detection model designed to operate on an IoT device, incorporating the most relevant anomaly detection steps (noise reduction, outlier detection, and threat detection).

Recent research shows that deploying anomaly detection procedures or even ML models closer to the data sources in sensor networks is a promising solution aiming to reduce the data movement to the cloud as distilling intelligence at the edge of the network is facilitated by novel powerful $\mu$controller units (MCUs). Even so, most of existing work does not show how anomaly detection methods have to be customized to be deployed at the level of smart sensors and neglect the data pre-processing step. Our work aims to illustrate how such simple techniques can be tailored to perform a data pre-labeling step at the level of embedded sensors and networked devices to offer the necessary support for more complex supervised ML models deployed at level of edge or cloud servers.

## 3 Tailoring Anomaly Detection Algorithms for the Online Setting

Detecting anomalies in sensor networks aims at catching outliers as soon as they are generated. The streaming nature of sensor data and the high rate at which it is generated makes traditional anomaly detection techniques inefficient in such a setting. Furthermore, the considerably limited storage and computing capabilities of networked devices responsible for collecting and forwarding the sensor measurements do not offer the necessary support for deploying computationally intensive anomaly detection algorithms and storing large amounts of data. Sliding window techniques are usually employed to allow storing only the most recent observations [3]. However, to accurately detect outliers within sensor readings, the window should be large enough to reflect the real evolution of the system or monitored phenomenon over time.

On the other hand, low response time of the outlier detection procedures has to be guaranteed in such an online context [2]. Detecting anomalies on the fly implies labeling the most recent measurement before another observation is collected. This also imposes limitations on the number of data samples considered when the label of the current reading is generated. Therefore, in most cases there is a trade-off between how many sensor observations are stored on the device and the time limit for labeling a single data sample. In what follows, we point out the optimization opportunities which can be exploited to ensure efficient outlier detection performed during the data collection process, focusing on the data structures used at storing the observations and on reusing the previously computed results. We illustrate the tailoring process by showing how two simple statistical outlier detection methods can be improved to guarantee the time constraints imposed when processing sensor data on networked devices.

## 3.1 Data Structures Selection

The shape in which sensor data is stored when employing sliding window techniques can affect the response time of the anomaly detection process, depending on the operations that are performed and on how the window is updated when a new sensor reading is encountered. Because of simple nature of sensor data and storage limitations, elementary data structures as arrays or linked lists are used in most cases to store the most recent observations. However, the complexity of basic operations, such as insertions and deletions, performed when updating the window of data samples or when labeling the most recent observation has a major impact on the response time. Therefore, using different data structures can highly influence the performance of the anomaly detection algorithm.

In the particular case of time series data, when using sliding window approaches, linked lists with pointers for both first and last elements allow performing both deletion of the oldest element (i. e. first element in the list) and insertion of a newly read sample (i. e. at the end of the list) in constant time: $O(1)$. Thus, using singly linked list defined using `head` and `tail` pointers show great improvements in this sliding window scenarios compared to using arrays allowing deleting the first element only in linear time (i. e. $O(n)$, for a window of size $n$) as it involves shifting all elements placed to its right. Moreover, linked lists also offer better support for handling dynamic workloads. However, as we show in what follows, in some cases simple singly linked lists are not efficient and have to be augmented to provide real performance enhancements, at the cost of additional memory.

We show the impact of the data structures on the anomaly detection process in an online setting using *Interquartile Range* (IQR) technique [1] for identifying outliers. The Interquartile Range method involves ordering the dataset and determining three reference values within the sorted dataset (the quartiles): $Q_2$ - the value in the middle position within the sorted dataset (the median or $2^{nd}$ quartile), $Q_1$ - the value on the middle position of the sorted subset of data between the minimum and median value (the $1^{st}$ quartile), and $Q_3$ - the value on the middle of the sorted subset of data between the median and the largest value (the $3^{rd}$ quartile). Based on these values, $IQR$ is computed as:

$$IQR = Q_3 - Q_1 \tag{1}$$

Using this value, the range of normal values is determined as:

$$[Q_1 - 1.5 \cdot IQR, Q_3 + 1.5 \cdot IQR] \tag{2}$$

The values placed outside this range are considered to be outliers. The pseudocode of the method is shown in Algorithm 1.

The time complexity of the technique depends on the sorting algorithm that is used and on the need to resort the dataset each time a new sensor measurement is recorded. Taking into consideration the fact that at each moment the most recent sensor readings have to be stored on the device, the observations must be stored both sorted and ordered by the moment of time they were read by the sensor to allow deleting the oldest sample each time a new value is recorded.

**Algorithm 1** Interquartile Range

```
1:  procedure IQR($x$)                                  ▷ $x$ is de unlabeled dataset
2:      $xs \leftarrow$ Sort($x$)                       ▷ $xs$ is the sorted dataset
3:      $n \leftarrow x.length$
4:      $Q_1 \leftarrow xs[\frac{n}{4}]$
5:      $Q_3 \leftarrow xs[\frac{3 \cdot n}{4}]$
6:      $IQR \leftarrow Q_3 - Q_1$
7:      for $t \leftarrow 0, x.length$ do
8:          if $x[t] > Q_3 + 1.5 \cdot IQR$ or $x[t] < Q_1 - 1.5 \cdot IQR$ then
9:              $x[t].label \leftarrow abnormal$
10:         else
11:             $x[t].label \leftarrow normal$
12:         end if
13:     end for
14: end procedure
```

**Arrays.** Using an array as a data structure for storing the values within the window imposes storing an additional array to store the sorted samples. A single array is not sufficient as the time ordering of measurements is lost after sorting the observations. This also requires resorting the window when recording each measurement or identifying its position in the sorted window and moving to the right of all the elements that are on the following positions in order to be able to insert the value on the correct position. Therefore, two arrays are required, one that stores the data samples in the order in which they are recorded and an auxiliary array that stores the sorted data samples. When each new value is recorded, array array containing the data in the order of registration is updated by removing the first item and inserting the most recent measurement at the last position. Subsequently, the values are sorted and stored in an auxiliary array, based on which the $Q_1$, $Q_3$, $IQR$ reference values are computed and the most recent observation is labeled.

**Ordered Linked Lists.** To avoid sorting the window each time a new value is recorded by the sensor, the measurements can be stored in a singly linked list ordered in ascending order. Even so, a simple linked list is not sufficient to keep track of the order in which the samples were recorded. Thus, we propose augmenting the list as shown in Figure 2 so that each element consists of a three-field structure:

- `value`, representing the actual value of the data sample;
- `index`, the unique number identifying the moment of time at which the sample was recorded;
- `next`, a pointer to the next element in the linked list.

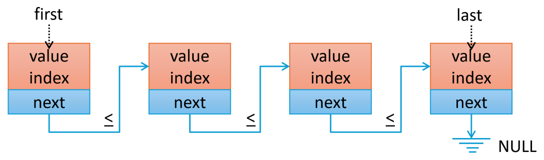

**Figure 2: Augmented singly linked list for storing the window of observations.**

The advantage of such a data structure is that the window can be updated in linear time and it does not require resorting the data after inserting a new value as each value is smaller than or equal to the value of the next element in the list. The actual deletion and insertion operations occur in constant time (i. e. $O(1)$), but a search for the position where the element will be inserted or for the element to be removed is necessary and it can be performed in linear time. However, this data structure comes at the cost of $O(n)$ (for a list of $n$ elements) additional memory to store the `next` pointer and `index` fields.

## 3.2 Reusing Previous Computations

Maintaining the context in resource-constrained environments is a major challenge of processing streaming sensor data. Employing sliding window techniques is a promising solution of preserving context in sensor data and keeping track of the historical evolution only of data which is still relevant to the current state of the system. Even though the number of samples taken into consideration when performing computations in a sliding window fashion is fairly small, the time constrains imposed by high sampling rates of sensors make online data processing a challenging task. Thus, reusing the computations performed in previous steps, when possible, leads to a significant gain in term of execution time. In the following, we propose looking for possibilities to discover recurrent formulas in algorithms and to use them in order to reduce the execution time in the case of time-constrained online execution. However, not all algorithms allow for reusing previous results in a recurrent manner.

We show how already computed results can be used to lower the computational cost of a simple statistical anomaly detection method called *Z-Score* [1], also known as *Grubb's test*. The main idea of this technique is that, assuming that the data follows a normal distribution, the majority of the data samples are expected to be located around the mean. More specifically, according to this method, the data samples found at a distance of at least three times the standard deviation of the data set from the mean are outliers.

In an online setting, applying this technique requires computing the z-score ($z$) for each newly sampled value ($x_{t_n}$), as:

$$z = \frac{x_{t_n} - \mu}{\sigma} \tag{3}$$

where $\mu$ is the mean and $\sigma$ is the standard deviation of the window ($\{x_{t_1}, x_{t_2}, ..., x_{t_n}\}$) of most recent observation. If the computed score, $z$, is less than $-3$ or greater than 3, the value is placed outside the interval $[\mu - 3 \cdot \sigma, \mu + 3 \cdot \sigma]$ and is labeled as anomalous. However, $\mu$ and $\sigma$ have to be recomputed at each update of the window, as:

$$\mu = \frac{\sum_{k=1}^{n} x_{t_k}}{n} \tag{4}$$

$$\sigma = \sqrt{\frac{\sum_{k=1}^{n} (x_{t_k} - \mu)^2}{n}} \tag{5}$$

Recomputing $\mu$ and $\sigma$ as shown above at each update of the window of values is time consuming and can lead to considerable increases in the response time. Thus, we propose recurrently computing the mean and standard deviation as an adaption of *Welford's online algorithm* [10] for sliding window scenarios. This online algorithm is used to incrementally compute the mean ($\mu_{n+1}$) and

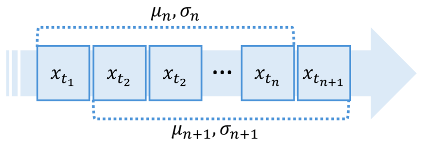

**Figure 3: Recurrent approach for updating the mean and standard deviation.**

standard deviation ($\sigma_{n+1}$) of a data set consisting of $n + 1$ samples using the previously computed mean ($\mu_n$) and standard deviation ($\sigma_n$), corresponding to the subset containing only the first $n$ samples. The recurrent formulas for $\mu_{n+1}$ and $\sigma_{n+1}$ are given by:

$$\mu_{n+1} = \mu_n + \frac{x_{t_{n+1}} - \mu_n}{n+1} \tag{6}$$

$$\sigma_{n+1}^2 = \sigma_n^2 + \frac{(x_{t_{n+1}} - \mu_n) \cdot (x_{t_{n+1}} - \mu_{n+1}) - \sigma_n^2}{n+1} \tag{7}$$

However, employing this technique in the sliding window fashion to handle the streaming nature of sensor data requires removing the effect of the oldest data sample in the mean and standard deviation when it is removed from the window. Therefore, each time a new sensor reading is encountered, we have to update the mean and standard deviation not only by including it in the computation of the mean and standard deviation but also by removing the effect of the measurement which is deleted from the set of most recent observations, as depicted in Figure 3. Moreover, in a sliding window context where each time a new data sample is recorded the oldest observation is removed from the data set, the size of the data set is constant, unlike the general context for which the Welford's algorithm was designed. Thus, we propose updating the mean and the standard deviation after recording a new observation $x_{t_{n+1}}$ and deleting the oldest observation $x_{t_1}$, as:

$$\mu_{n+1} = \mu_n + \frac{x_{t_{n+1}} - x_{t_1}}{n} \tag{8}$$

$$\begin{aligned} \sigma_{n+1}^2 = \sigma_n^2 &- \frac{(x_{t_1} - \mu_n) \cdot \left(x_{t_1} - \left(\mu_n - \frac{x_{t_{n+1}} - x_{t_1}}{n}\right)\right)}{n} \\ &+ \frac{(x_{t_{n+1}} - \mu_{n+1}) \cdot \left(x_{t_{n+1}} - \left(\mu_{n+1} - \frac{x_{t_{n+1}} - x_{t_1}}{n}\right)\right)}{n} \end{aligned} \tag{9}$$

Equation 8 shows how we update the mean by adjusting the previous mean with the difference between the latest sampled value and the removed value, divided by the window size. In Equation 9 we propose updating the standard deviation by subtracting the effect produced by the oldest observation (i. e. the one which was removed when the most recent value was sampled) and add the effect produced by the most recent sample. To avoid cases in which the results of the computations conduct to a negative value of $\sigma_n^2$, we set the value to 0 if the result is negative. As *Welford's online algorithm*, our method leads to an estimated value of the standard deviation, but the error is nearly irrelevant given the precision of computations performed on resource constrained devices.

The major gain of such a recurrent approach is that it reduces both execution time and computational overhead by reusing previously computed values. Thus, the response time can be considerably improved by reducing the time complexity of the algorithm: from linear time to constant time in the case of the described technique. Nevertheless, not all algorithms allow for recurrent computations.

## 4 Experimental Evaluation

We evaluate the performance and scalability of our proposed solutions on a NodeMCU development board which integrates an ESP8266 $\mu$controller having 4 MB of Flash memory for storing the program, 98 KB of DRAM for storing variables and constants and a clock frequency of 80 MHz. The DRAM memory is split into stack, used at storing statically allocated data, and heap, which stores the dynamically allocated data. To evaluate the implementations in a real online setting, we use a DHT11 humidity and temperature sensor, which we connect to the NodeMCU board through the serial interface.

We comparatively evaluate the implementations using arrays and ordered linked lists of the *IQR* and the straightforward and recurrent versions of *Z-Score* in terms of both execution time and memory usage and show the improvements of our tailored solutions for detecting anomalies in sensor data on resource-constrained devices during the data collection process. We also perform a quantitative evaluation to determine how many sensor measurements can be stored at once on the device depending on the data structure used to implement the sliding window. For this purpose, we compute the maximum number of measurements which can be stored on the device depending on the implementation as:

$$N_{max} = \left\lfloor \frac{DRAM_{total} - DRAM_{used}}{DRAM_{sample}} \right\rfloor \tag{10}$$

where $DRAM_{total}$ is the total available DRAM (only around 80192 B out of the 98000 B can be used to store the application data due to a constant overhead of around 18000 B), $DRAM_{used}$ is the amount of memory used by the data excluding the window of sensor measurements and $DRAM_{sample}$ is the implementation-dependant size of a sample.

**Interquartile Range.** We demonstrate the impact of data structures on the performance of online anomaly detection in the case of *IQR* technique by measuring the memory the time taken to label a sensor reading in case of both implementations (using arrays and using ordered singly linked lists) for several sizes of the window.

As shown in Figure 4, the execution time is significantly shorter for the implementation of the IQR algorithm that uses linked lists ordered in ascending order, compared to the alternative that stores the values in arrays. The second alternative is considerably slower as it requires resorting the values each time a new measurement is recorded. We used *Bubble Sort* algorithm in the case of array-based implementation. Moreover, as the window size increases, the difference between the execution times of the two implementations is more noticeable. For instance, an increase of the size of the window from 120 to 150 samples leads to an increase from 3825$\mu$s to 5970$\mu$s in the execution time of the array-based version, while the same increase in the window size results in an increase from 180$\mu$s to only 184$\mu$s in the execution time when using the ordered linked lists alternative. This difference is the result of reducing the complexity of the algorithm from quadratic time (i.e. $O(n^2)$) in the case of array-based solution to linear time (i.e. $O(n)$), using ordered linked lists.

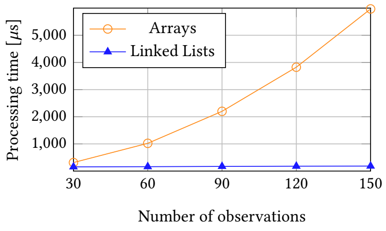

**Figure 4: IQR: Execution time of array-based (requiring re-sorting the window each time a new sample is recorded) and ordered linked lists.**

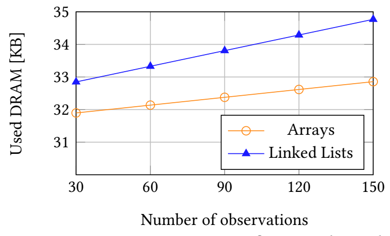

**Figure 5: IQR: DRAM Memory usage of array-based (requiring re-sorting the window each time a new sample is recorded) and ordered linked lists.**

Figure 5 shows the memory used to store the variables and constants for each of the two alternative solutions of the *IQR* algorithm. As the figure presents, even though both solutions show a constant increase in the amount of DRAM used to store the data, the ordered linked list not only uses more memory but also leads to a higher increase in the memory used to store the data when the size of the window storing the measurements grows. This is due to the fact that for each observation, besides the 4 bytes required to store the actual sensor reading as a floating point number represented in single precision, we need 4 more bytes to store the `next` pointer and 8 more bytes to store the `index` field of type `long`, while in the case of array-based implementation only 4 bytes are required to store each sensor reading in the time-ordered array and another 4 bytes to store it in the sorted array. Thus, each sensor measurement requires 16 bytes in the ordered lists-based solution and 8 bytes in the array-based one. The maximum allowed number of observations which can be stored on the device, computed as in Equation 10, is $N_{max}^{(A)} = 3003$ for the array-based version of *IQR* and $N_{max}^{(LL)} = 2929$ for the ordered linked lists solution.

**Z-Score.** We perform a similar execution time and memory usage analysis to demonstrate the performance improvements achieved by reusing prior computations with reference to *Z-Score* anomaly detection technique. We compare the amount of memory needed to store the data and the time required to process a sensor reading between the iterative solution which recomputes the mean and the standard deviation at each update of the window and our proposed recurrent solution.

As depicted in Figure 6, the execution time of the iterative solution grows linearly with the size of the window of values taken into account as it requires recomputing the mean and standard deviation each time a new sensor measurement is recorded. On the other hand, the recurrent solution which computes the mean and estimated standard deviation based on the previously computed mean and standard deviation has a constant and significantly lower

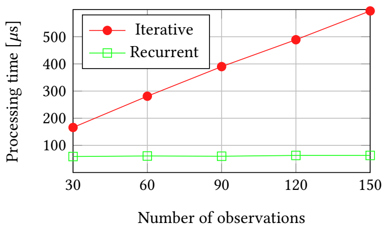

**Figure 6: Z-Score: Execution time of both iterative and recurrent implementations.**

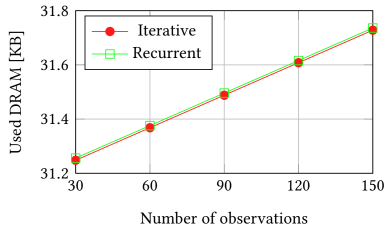

**Figure 7: Z-Score: DRAM Memory usage of both iterative and recurrent implementations.**

execution time. However, the data labeling process starts only after there are enough sensor readings to fill the window. Therefore, even the recurrent solution requires computing the standard deviation in an iterative way when the number of recorded measurements reaches the size of the window for the first time as there is no pre-computed standard deviation. Even so, the mean can be recurrently computed following *Welford's online algorithm*.

The memory usage of both versions of *Z-Score* is presented in Figure 7. It is noticeable that the amount of memory used to store the data increases linearly with the size of window for both solutions. Furthermore, the two alternatives of implementation use almost the same amount of memory to store the constants and variables. The recurrent solution requires additional 8 bytes compared to the straightforward implementation used to store the previous mean (4 B) and standard deviation (4 B). Each sensor reading is represented as single precision floating point value and takes 4 bytes of memory and we use an array to store the most recent sensor readings. Therefore, the maximum size of the window computed according to Equation 10 is $N_{max}^{(I)} = 12206$ for the iterative solution and $N_{max}^{(R)} = 12204$ for the recurrent implementation.

## 5 Conclusions

In this paper, we propose a methodology of tailoring simple data pre-processing models targeting to run at the level of smart sensors or resource-constrained networked devices in order to perform data filtering and pre-labeling before sending it to more computationally expensive ML models. We illustrate the tailoring process using two basic statistical-based anomaly detection techniques, demonstrating the impact of data structures and reusing previous computations on the performance of the algorithm in order to ensure meeting the time constraints of an online setting. The performance improvements emerging from using proper data structures at the cost of additional memory are presented in the case of *Interquartile Range* method, while the enhancements of recurrent computations are illustrated by means of *Z-Score* outlier detection technique. We experimentally evaluate the proposed improvements in a real online setting which uses a humidity and temperature digital sensor connected to a MCU. Our results show that the time complexity of the such algorithms can be significantly lowered by using proper data structures and reusing already computed results. Moreover, we perform a quantitative analysis to determine how many sensor readings can be stored on the device depending on the outlier detection method implementation.

This research provides a new perspective on anomaly detection performed at the level smart sensors as an emerging need of providing a first step of pre-processing and filtering observations before feeding them to a more sophisticated and computationally demanding ML models. In future work, the proposed method of customizing anomaly detection methods will be further expanded and optimized for different sensor data to handle data and device heterogeneity and to ensure online outlier detection on more complex or aggregated sensor data.

## Acknowledgments

This research received funding from the European Union's Horizon Europe research and innovation program under the Grant Agreements number 101136216 (Hedge-IoT). Views and opinions expressed are, however, those of the author(s) only and do not necessarily reflect those of the European Union or the European Climate, Infrastructure, and Environment Executive Agency. Neither the European Union nor the granting authority can be held responsible for them.

## References

[1] Varun Chandola, Arindam Banerjee, and Vipin Kumar. 2009. Anomaly detection: A survey. *ACM Comput. Surv.* 41, 3, Article 15 (jul 2009), 58 pages. https://doi.org/10.1145/1541880.1541882

[2] Laura Erhan, M Ndubuaku, Mario Di Mauro, Wei Song, Min Chen, Giancarlo Fortino, Ovidiu Bagdasar, and Antonio Liotta. 2021. Smart anomaly detection in sensor systems: A multi-perspective review. *Information Fusion* 67 (2021), 64–79. https://doi.org/10.1016/j.inffus.2020.10.001

[3] Federico Giannoni, Marco Mancini, and Federico Marinelli. 2018. Anomaly detection models for IoT time series data. *arXiv preprint arXiv:1812.00890* (2018).

[4] Yi Li, Zhangbing Zhou, Xiao Xue, Deng Zhao, and Patrick C. K. Hung. 2023. Accurate Anomaly Detection With Energy Efficiency in IoT–Edge–Cloud Collaborative Networks. *IEEE Internet of Things Journal* 10, 19 (2023), 16959–16974. https://doi.org/10.1109/JIOT.2023.3273542

[5] Mao V. Ngo, Hakima Chaouchi, Tie Luo, and Tony Q. S. Quek. 2020. Adaptive Anomaly Detection for IoT Data in Hierarchical Edge Computing. arXiv:2001.03314 [cs.LG] https://arxiv.org/abs/2001.03314

[6] Paul D. Rosero-Montalvo, Zsolt István, Pınar Tözün, and Wilmar Hernandez. 2023. Hybrid Anomaly Detection Model on Trusted IoT Devices. *IEEE Internet of Things Journal* 10, 12 (2023), 10959–10969. https://doi.org/10.1109/JIOT.2023.3243037

[7] Sebastian Schmidl, Phillip Wenig, and Thorsten Papenbrock. 2022. Anomaly detection in time series: a comprehensive evaluation. *Proc. VLDB Endow.* 15, 9 (may 2022), 1779–1797. https://doi.org/10.14778/3538598.3538602

[8] Arnaldo Sgueglia, Andrea Di Sorbo, Corrado Aaron Visaggio, and Gerardo Canfora. 2022. A systematic literature review of IoT time series anomaly detection solutions. *Future Generation Computer Systems* 134 (2022), 170–186. https://doi.org/10.1016/j.future.2022.04.005

[9] Sergio Trilles, Sahibzada Saadoon Hammad, and Ditsuhi Iskandaryan. 2024. Anomaly detection based on Artificial Intelligence of Things: A Systematic Literature Mapping. *Internet of Things* 25 (2024), 101063. https://doi.org/10.1016/j.iot.2024.101063

[10] Barry Payne Welford. 1962. Note on a method for calculating corrected sums of squares and products. *Technometrics* 4, 3 (1962), 419–420.

[11] Yang Zhang, Nirvana Meratnia, and Paul Havinga. 2010. Outlier Detection Techniques for Wireless Sensor Networks: A Survey. *IEEE Communications Surveys & Tutorials* 12, 2 (2010), 159–170. https://doi.org/10.1109/SURV.2010.021510.00088