01 Jan 2023

LightESD: Fully-Automated and Lightweight Anomaly Detection Framework for Edge Computing

Ronit Das

Tie (Tony) T. Luo
Missouri University of Science and Technology, tluo@mst.edu

Recommended Citation

R. Das and T. T. Luo, "LightESD: Fully-Automated and Lightweight Anomaly Detection Framework for Edge Computing," *Proceedings - IEEE International Conference on Edge Computing*, pp. 150 - 158, Institute of Electrical and Electronics Engineers, Jan 2023.
The definitive version is available at https://doi.org/10.1109/EDGE60047.2023.00032

2023 IEEE International Conference on Edge Computing and Communications (EDGE) | 979-8-3503-0483-1/23/$31.00 ©2023 IEEE | DOI: 10.1109/EDGE60047.2023.00032

LightESD: Fully-Automated and Lightweight Anomaly Detection Framework for Edge Computing

Ronit Das
Department of Computer Engineering
Missouri University of Science and Technology
Rolla, USA
rdkz8@umsystem.edu

Tie Luo*
Department of Computer Science
Missouri University of Science and Technology
Rolla, USA
tluo@mst.edu

***Abstract*—Anomaly detection is widely used in a broad range of domains from cybersecurity to manufacturing, finance, and so on. Deep learning based anomaly detection has recently drawn much attention because of its superior capability of recognizing complex data patterns and identifying outliers accurately. However, deep learning models are typically iteratively optimized in a central server with input data gathered from edge devices, and such data transfer between edge devices and the central server impose substantial overhead on the network and incur additional latency and energy consumption. To overcome this problem, we propose a fully-automated, lightweight, statistical learning based anomaly detection framework called LightESD. It is an on-device learning method without the need for data transfer between edge and server, and is extremely lightweight that most low-end edge devices can easily afford with negligible delay, CPU/memory utilization, and power consumption. Yet, it achieves highly competitive detection accuracy. Another salient feature is that it can auto-adapt to probably any dataset without manually setting or configuring model parameters or hyperparameters, which is a drawback of most existing methods. We focus on time series data due to its pervasiveness in edge applications such as IoT. Our evaluation demonstrates that LightESD outperforms other SOTA methods on detection accuracy, efficiency, and resource consumption. Additionally, its fully automated feature gives it another competitive advantage in terms of practical usability and generalizability.**

***Index Terms*—Extreme studentized deviate, anomaly detection, on-device learning, periodicity detection, edge computing**

I. Introduction

The inception of research in outlier detection can be dated to as early as 1852 when Benjamin Peirce came up with *Peirce's Criterion* [1] to detect and remove outliers from numerical data. Since then, research in this area has grown remarkably and has now become ubiquitous in almost every domain, such as cybersecurity, transportation, manufacturing, finance, and computer networks.

With the rapid development of edge computing, a large number of applications that require real-time response have been moving to edge devices. The concern pertaining to cyber attacks, fault diagnosis, and other similar data analytics has also urged development of anomaly detection for edge computing. Traditional machine learning-based methods [2], [3] have seen a recent trend of being replaced by deep learning-based methods [4]–[7], due to the latter's state-of-the-art (SOTA) performance in detection accuracy. However, deep learning models typically require intensive training and a large amount of data; as a result, a central server or cloud is often deployed which collects data from edge devices and then performs model training [6]. This entails data transfer over the network and adds substantial network traffic and overhead, as well as incurs large delay. An alternative is to train the model offline at a central server using all the historical data, and then deploy the model at edge devices for inference only. However, this approach is not able to keep up with new data and can lead to the problem of *concept drift* [8].

In this paper, we propose a fully-automated, lightweight, statistical learning-based anomaly detection framework called LightESD, for detecting anomalies directly at the edge site. It is extremely lightweight with little resource consumption and little training overhead that almost all edge devices can afford (we have quantified these in our evaluation). A salient feature of LightESD is that it is a *weight-free*, unsupervised model, meaning that it stores no weights. This enables it to *auto-adapt to any data* to learn the underlying patterns on the fly in a fully unsupervised manner, without the need for manual *pre-processing or hyperparameter-tuning* to "match" any specific dataset. Thus, it is much favorable for practical adoption, and has good generalizability over different data.

Another important feature of LightESD is that it is *non-parametric*, meaning that it does not make any distributional or functional-form assumptions of the observed (original) data, whereas many other statistic approaches do. This also contributes to its good generalizability. Note that *being non-parametric is not equivalent to being weight-free*; for example, SVM [9], [10] with RBF-kernel is non-parametric but has weights.[1]

This paper focuses on time series which pervades in numerous application domains (such as Edge) that have temporal properties. The main contributions of this paper are:

*Corresponding author.

[1]Support vector coefficients (which are dual coefficients) are learnt from training data and stored in the memory for making predictions. Hence whenever we have another dataset, the model needs to be retrained and the coefficients (weights) need to be replaced. However, our proposed approach auto-adapts to any underlying data to make predictions on the fly, without explicit training or the need for storing any coefficients/weights.

2767-9918/23/$31.00 ©2023 IEEE
DOI 10.1109/EDGE60047.2023.00032

1) We propose LightESD, an anomaly detection framework for time series, that realizes on-device learning and suits deployment on edge devices. LightESD is weight-free, non-parametric, unsupervised, and can auto-adapt to perhaps any univariate time series regardless of the underlying distribution, seasonality, or trend, without manual pre-processing or hyperparameter-tuning.
2) We propose a new evaluation metric, *ADCompScore*, that allows for comparison of edge anomaly detection models in a holistic manner. To the best of our knowledge, this is the first attempt to develop a new metric to understand not only the anomaly detection performance but also the computational power and resource usage, using a single numeric value which allows for quick decision-making regarding the feasibility of deploying anomaly detection algorithms on edge devices.
3) We evaluate LightESD using both synthetic and real-world datasets and demonstrate its superior overall performance compared to SOTA methods, in terms of both anomaly detection and feasibility for on-device training and edge deployment.

The rest of this paper is organized as follows. Section II discuss related work as the background. Section III presents the proposed LightESD approach. Section IV describes our experiments and comprehensive evaluation of the proposed method in comparison with other methods. Section V concludes with future directions.

II. Related Work

Statistical methods for anomaly detection. These have been the go-to approaches for a long time, such as ARIMA and linear regression. While they require minimal effort for the model to learn from data, they hinge on the *normality* assumption that the underlying data must conform to a *Gaussian* or *Gaussian-like* distribution [11], which often does not hold in real-world data or cannot characterize multi-modal distribution encountered in some datasets [12]. This makes such models, in their standalone form, unsuitable for detecting anomalies in real-world data.

Machine Learning and Deep Learning based anomaly detection. Some of the commonly used machine learning-based methods include distance based techniques [13], density-based techniques [2], tree-based techniques [3], Bayesian networks [14], and clustering techniques [15]. Most machine learning-based anomaly detection methods have been superseded by deep learning-based approaches owing to the latter's much better anomaly detection performance. Indeed, recent advances in deep learning (DL) have created a hype of using DL in nearly all tasks including anomaly detection. Unsupervised DL architectures include those based on deviation networks [16], adversarial learning such as f-AnoGAN [17] and TadGAN [5], an other variants. While deep learning-based methods can achieve good performance, they often come with a large memory and computational footprint, which can pose a bottleneck for deployment on edge devices.

Anomaly Detection for the Edge. While there exist several papers that discuss the hardware-implementation of neural network-based anomaly detection approaches like ANNet [6], LightLog [4], and others, all of them adopt the approach of training a model at a central server and then deploying ("implementing") it on the device hardware. On the other hand, ONLAD [18] proposes to develop an OS-ELM [19] based approach to detect anomalies, which can be directly trained and deployed at an edge site. Since [18] attempts to solve the same challenge as we do, we compare it with our proposed approach in Section IV-B.

Some of the recent works, like that of Bayesian Random Vector Functional Link AutoEncoder with Expectation Propagation (EPBRVFL-AE) [20], take a distributed approach to training their proposed anomaly detection schemes. While taking a distributed or federated learning-based approach might seem intuitive, such a direction can increase the overall communication overhead in a network, thereby degrading the network's efficiency. Moreover, such approaches still require a central server and a communication network which are not always available especially in rural or challenging environments, which is where our work fits in.

III. The LightESD Framework

LightESD stands for *Lightweight Extreme Studentized Deviate* test. In a nutshell, it works as follows. A time series Y_t can be decomposed additively as

$$Y_t = T_t + \sum_{i=1}^{k} S_t^i + R_t \tag{1}$$

where T_t, S_t^i and R_t represent the *trend*, the i-th *seasonal*, and the *residual* components, respectively, and there are a total of $k \geq 1$ seasonal components. LightESD first detects all the different seasonal periodicities in the input time series, i.e., to determine S_t^i, where it uses *Welch's Periodogram Method* with a PSD-locating technique improved by us (Section III-A). Second, it extracts the trend component T_t using *RobustTrend* and *FastRobust-STL* decomposition methods, and then removes both the detected trend and seasonality components, to extract the residual R_t (Section III-B). Third, LightESD detects anomalies based on the residual—which is a statistically correct (because the residuals follow a normal or approximately-normal distribution) and much more reliable method than detecting based on original signals—using a generalized ESD test with our improvement in robustness (Section III-C). Note that, our choice of the above methods is based on careful contemplation and trial experiments. We explain our choice below when we describe those methods.

A. Improved Periodicity Detection

Time series data often exhibit a recurring pattern at regular time intervals like weekly, monthly, or yearly, which is called the *seasonality* of a time series. We detect seasonality by first computing a *periodogram*, which is an estimate of the power spectral density (PSD) of a signal, and then analyzing the

Algorithm 1: Improved Period Detection

Input: Y: Time series
Output: prd: Array of detected periods

for $i = 1$ *to* 100 **do**
  $Y' \leftarrow \texttt{permutation}(Y)$
  $freq, pow \leftarrow \texttt{Welch}(Y')$
  $p_{max} \leftarrow \texttt{max}(pow)$
  $max\_power.\texttt{add}(p_{max})$
$max\_power \leftarrow \texttt{sort}(max\_power, \texttt{ascending} = \texttt{true})$
$index \leftarrow 0.99 \times \texttt{len}(max\_power)$
$thresh \leftarrow max\_power[index]$
$freq, pow \leftarrow \texttt{Welch}(Y)$
$prd \leftarrow -1$, $temp\_psd \leftarrow -1$
for $j = 1$ *to* $\texttt{len}(pow) - 1$ **do**
  if *(*$pow[j] > thresh$*) and (*$pow[j] > pow[j-1]$*) and (*$pow[j] > pow[j+1]$*)* **then**
    if *(*$pow[j] > temp\_psd$*)* **then**
      $prd.\texttt{add}(\left\lfloor \frac{1}{freq[j]} \right\rfloor)$
      $temp\_psd \leftarrow pow[j]$
if *(*$prd == -1$*)* **then**
  $prd \leftarrow 1$ `// Nonseasonal`
return prd

periodogram to find out the dominant frequencies that generate the highest PSD estimates above a designated threshold. There are both parametric and non-parametric methods to estimate PSD, where parametric methods assume an underlying data distribution which may not hold. Therefore, we take a non-parametric approach.

1) Welch's Periodogram Method: Among the few non-parametric methods that can compute the Periodogram for a time series, we choose Welch's method [21] as it outputs the most reliable PSD estimates, even from noisy data. This method first splits the signal into K data segments of length L, which are overlapped by D points (D/L ranges from 0 to 0.5). This can reduce the effect of noise on PSD estimation, unlike *Bartlett's method* which has no overlapping. Second, each of the K overlapping segments is applied a time window that is either quadratic ($1 - t^2, t \in [-1, 1]$) or triangular ($1 - |t|, t \in [-1, 1]$). Third, a periodogram is generated for each window by first computing the discrete Fourier transform (DFT) [22] and then the squared magnitude of the DFT output. Finally, the individual periodograms are averaged to reduce the variance of individual power measurements.

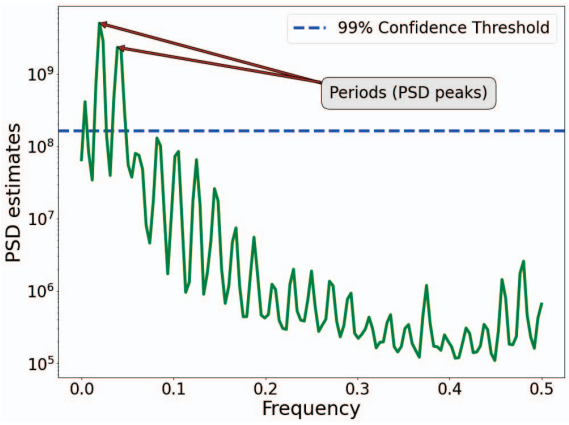
2) Improved Periodicity Detection: From the obtained periodogram, we need to find the significant PSD peaks which correspond to the different periods (seasonalities), if they exist. Although Welch's method reduces noise to some extent via overlapping segments, it is fairly primitive and the remaining noise in time series still creates many spurious PSD peaks in the resulting periodogram. Therefore, we improve a periodicity detection approach *AutoPeriod* [23] to filter out those spurious peaks. Specifically, we first permute the original time series for 100 times and compute the periodogram for each permuted sequence using the Welch's method. In each iteration, the random permutation destroys all the temporal correlations as well as the second and higher-order central moments of the original time series, and thereby convert the time series into pure noise. Among all the PSD values generated from each noise series using the *Welch's* method, we take the maximum PSD value and add it into a vector. After that, the vector containing the 100 PSD values is ordered in an ascending manner, and the PSD value at the 99th index is selected as the threshold value (basically it calculates the 99-th percentile). The reason for choosing this particular value as the threshold is that 99% of the PSD values come from the permuted version of the original time series (noise), which always lie below the 99-th percentile value (also known as the threshold PSD value). So any PSD estimate lying above this threshold is significant, at 99% confidence, while those lying below are just PSD estimate from pure noise which we disregard.

Our contribution to this period detection algorithm is that, unlike [23] which takes *all* the PSD values and has a high space complexity of $\mathcal{O}(N)$, our proposed periodicity detection method selects significant peaks only (since a peak represents a cyclic event) which has a space complexity of merely $\mathcal{O}(1)$. In other words, as the length of the time series increases, the space consumed by [23] increases approximately proportionally as it considers all the significant values whereas our method remains constant. For time complexity, both [23] and our approach are at the same level $\mathcal{O}(N \log N)$, due to the usage of Fast Fourier Transform.

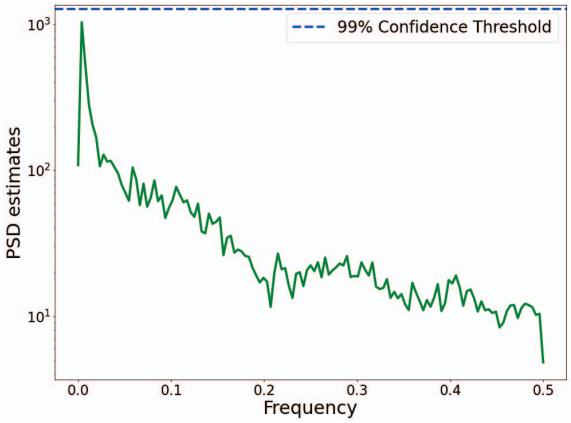
The above procedure is presented in Algorithm 1 and the outcome is illustrated in Fig. 1 using the Numenta Anomaly Benchmark (NAB) dataset [24]. We have considered other approaches that detect periodicities, such as *AutoAI-TS* [25], where the latter uses *pre-defined* schemes only (like hourly / daily / weekly periodicities) to detect periodicities (seasonalities) in the time domain (whereas we detect in the frequency domain). This has the limitation of not being able to detect periods that are *not* in the manually pre-defined set.

B. Residuals Extraction

To extract the residuals R_t, we use *RobustTrend* [26] if the time series is nonseasonal (aperiodic) where our Algorithm 1 will return $prd = 1$. Otherwise, if the series is seasonal, we use *FastRobust-STL* [27] to extract both the seasonality and the residuals. There are four reasons why we choose *RobustTrend* [26] for decomposing aperiodic time series and why *FastRobust-STL* [27] for seasonal series: (1) these two methods can model both abrupt and slow-changing trend components, (2) they are robust to outliers and noise, (3) *FastRobust-STL* is able to handle multiple seasonalities and can decompose much faster than other comparable methods, and (4) both approaches are free from assumption of any particular distribution and are non-parametric.



(a) Seasonal periods are correctly detected.



(b) Non-seasonality also detected (not fooled by noise).

Fig. 1: Illustration of our periodicity detection result on NAB [24]: (a) NYC Taxi dataset (seasonal), (b) UPS Tweets dataset (nonseasonal).

1) Nonseasonal Series: In this case, we use *RobustTrend* to de-trend and extract residuals. In order to extract the trend from the time series in a reliable way, it is important to mitigate the negative impact of noise and outliers. This is achieved by minimizing the *Huber Loss* of the residual signal, combined with first and second order difference regularization, as follows:

$$\arg\min_{\mathbf{t}} h_\gamma(\mathbf{y}-\mathbf{t}) + \lambda_1 ||\mathbf{D}_{(1)}\mathbf{t}||_1 + \lambda_2 ||\mathbf{D}_{(2)}\mathbf{t}||_1 \qquad (2)$$

where $h_\gamma(.)$ is the *Huber Loss*, $\mathbf{D}_{(1)}$ is the first-order difference matrix, $\mathbf{D}_{(2)}$ is the second-order difference matrix, and λ_1 and λ_2 control the amount of the regularization.

The optimization problem (2) can be solved using *Alternate Direction Method of Multipliers* (ADMM) based on *Majorization-Maximization* [28], to estimate the trend $\mathbf{t}^* = T_t$. Then the residuals are extracted by $R_t = Y_t - T_t$.

2) Seasonal Series: In this case, we use *FastRobust-STL* [27] which first extracts the trend using a robust sparse model. In order for accurate estimation of the trend component, first we need to remove noise and the adverse effect of the different seasonalities (note that this is different from estimating the different seasonal components), as well as to take into account possible outliers. To remove noise, a *bilateral filter* [29] is used to denoise the time series. To remove the influence of seasonal components on trend extraction, a seasonal differencing operation is carried out, which refers to the difference between the outputs of the bilateral filter for every timestep t and that for $t-T$, where T is the largest seasonal period (i.e., $T = \max(prd)$) in the time series.

Then, the trend is extracted by formulating an objective function that minimizes the least absolute deviation (LAD) of the smoothed (de-noised) and seasonal-differenced signal, in order to make the trend extraction robust to large outliers; the objective is further combined with L_1-regularizations to capture both abrupt and slow trend changes.

Then, after extracting and removing the trend, in order to estimate the multiple seasonal components, an improved version [27] of a non-local seasonal filter [30] is used to extract the different seasonal components present in the time series.

Finally, the residual component is extracted by subtracting the trend and the multiple seasonal components from the original time series, as $R_t = Y_t - T_t - \sum_{i=1}^{k} S_t^i$.

C. Anomaly Detection based on Residuals

The rationale that we perform anomaly detection based on residuals instead of original data, is that such residuals as we extracted above, are unimodal and follow an approximately Gaussian distribution. This stems from the fact that trend-adjusted series (time series with its trend removed) are *stationary* and have been empirically shown to follow an approximately-Gaussian distribution [12]. This is an important property that makes our anomaly detection much more accurate than other anomaly detection methods when the original data distribution is not Gaussian-like or unimodal. For example, using any real-world data (which may have multiple mode(s)) directly for detecting anomalies, can result in many of the anomalies not being detected at all [12].

A classical approach to outlier detection is to conduct a *hypothesis test* at a certain significance level to decide whether to reject the null hypothesis (a data point in question is *not* an outlier). Some of the most common statistical methods including Pierce's Criterion [31] and the Tietjen Moore Test [32] take this approach. However, these methods have various drawbacks such as being not scalable to large datasets or requiring prior knowledge of the exact number of outliers in the data. In LightESD, we propose a method based on an improvement to the generalized Extreme Studentized Deviate (ESD) test [33].

1) Generalized Extreme Studentized Deviate Test: The ESD test [33] enhances the standard Grubb's Test [34] in that ESD can find up to a user-specified maximum, say a_{max}, of outliers in a series of data points, whereas Grubb's Test can only find a single outlier. However, both ESD and Grubb's Test assume normality of the input data which often does not hold, while our method does not make that assumption. ESD runs a_{max}

iterations and, in each iteration, it tests if a single outlier exists by comparing a test statistic R with a critical value λ. The test statistic R is defined as

$$R = \max_i \left(\frac{|Y_i - \mu|}{\sigma} \right), \; i = 1, ..., n \tag{3}$$

where μ and σ are the mean and standard deviation, respectively, of the input data Y of length n. The critical value λ is defined with respect to a significance level α, as

$$\lambda = \frac{t_{n-l-2,p} \times (n-l-1)}{\sqrt{(n-l) \times (t^2_{n-l-2,p} + n - l - 2)}} \tag{4}$$

where $p = 1 - \frac{\alpha}{2} \times (n-l)$, l is the iteration index ranging from 0 to $a_{max} - 1$, and n is the current length of the series. The value of $t_{(.,.)}$ can be looked up in the *two-tailed T-Distribution table*. If $R > |\lambda|$, we reject the null hypothesis that Y_{i^*} is not an outlier, where i^* is the index i that yields R as in (3). Then we remove this outlier Y_{i^*} from Y, decrement n by 1, and move to the next iteration (increment l by 1). Otherwise, no outlier is detected in this iteration and the algorithm continues to the next iteration until l reaches a_{max}.

2) Improvement to ESD: Besides that we do not make the normality assumption of observed data that ESD makes when it operates on the observed data in a standalone form, we also identify that ESD test is vulnerable to deviant points: the test statistic R used by ESD in (3) will break down when the series Y contains a small fraction of, or even just a single, very large outlier. To address this issue, we redefine the test statistic R as

$$R_{robust} = \max_i \left(\frac{|Y_i - \text{median}(Y)|}{S(Y)} \right), \; i = 1, .., n \tag{5}$$

$$S(Y) = \text{median}_i \left(\text{median} \, |Y_i - Y| \right). \tag{6}$$

That is, we replace μ and σ as in (3) with median and a *robust estimator* S [35], respectively. This is important and implies that the *finite sample breakdown point* [36] will now change from $1/(n+1)$ to $\lfloor \frac{n}{2} \rfloor / n$. What this means is that our method is able to tolerate 50%, in the asymptotic case, of all the values to be arbitrarily large. Therefore, our method is far more robust than the original ESD since outliers can never exceeds 50% (otherwise they would not be called "outliers").

Moreover, it is also worth noting that we choose the robust statistic S over *Median Absolute Deviation* (MAD) as used by [12], even though both can achieve asymptotic 50% toleration of outliers. The rationale is that the statistic S does not make the assumption of *symmetric distribution* that MAD requires [37], and S is more efficient than MAD when dealing with Gaussian distributions [35].

D. Putting All Together

The entire anomaly detection process is summarized in Algorithm 2. At Line 6, a_{max} is set to 10% of the total number of data points since a real-world dataset typically contains less then 4-5% of anomalies. Importantly, note that this value is highly *insensitive* to different datasets since it is an *upper bound* to the number of anomalies rather than the exact quantity. The runtime complexity of Algorithm 2 is $\mathcal{O}(N \log N)$, owing to Algorithm 1. The space complexity of Algorithm 2 is $\mathcal{O}(N)$, since in the worst case scenario, the detected anomalous time indices can grow as a fraction of the length of the entire time series being analyzed.

Algorithm 2: LightESD: The complete procedure

Input: Y: Time Series
Output: $anomalies$: the corresponding indices
Initialize: $outlier\_index \leftarrow []$
1 $period \leftarrow$ `ImprovedPeriodDetection`(Y)
  `// Algorithm 1`
2 **if** $period == 1$ **then**
3 $residual \leftarrow Y -$ `RobustTrend`(Y)
4 **else**
5 $residual \leftarrow$ `FastRobustSTL`$(Y, period)$
6 $a_{max} \leftarrow 0.1 \times$ `len`(Y)
7 $outliers \leftarrow$ `ImprovedESD`$(\alpha = 0.05, residual, a_{max})$
8 $outlier\_index \leftarrow outliers.index$
9 **if** *(*$outliers[1] ==$ `True`*) and (*$outliers[2] ==$ `False`*)* **then**
10 $outliers[1] =$ `False`
11 $outlier\_index$.`pop`(1)
12 **if** *(*$outliers[n] ==$ `True`*) and (*$outliers[n-1] ==$ `False`*)* **then**
13 $outliers[n] =$ `False`
14 $outlier\_index$.`pop`(n)
15 $anomalies \leftarrow outlier\_index$
16 **return** $anomalies$

IV. Performance Evaluation

A. Experiment Setup

We used a single board computer, Hardkernel Odroid XU4 (OS: Ubuntu 22.04, CPU: Samsung Exynos 5422 Cortex A15 2.0 GHz and Cortex-A7 Octacore CPU, RAM: 2 GB LPDDR3 PoP stacked), as our experimental machine to emulate an edge device, that is capable of on-device training and deployment of anomaly detection models. See Fig. 2 for the setup.

Datasets. We evaluate our proposed LightESD framework on both synthetic and real-world datasets. We carefully synthesize two types of datasets that emulate different real-world univariate time series, as follows:

1) *Seasonal data (with both trend and seasonality)* ($\mathcal{STD}$):

$$Y_t = \kappa \cdot t^2 + \beta \cdot \sin(\frac{2\pi t}{30.5}) + \gamma \cdot \epsilon_t \tag{7}$$

where κ, β, γ are drawn randomly from [0.001,0.01], [1.3e+04, 1.5e+04], and [1.5e+03, 3.0e+03], respectively, and $\epsilon_t \sim N(0,1)$ is gaussian white noise. The period is 30.5 (days) to emulate a monthly seasonality.

2) *Random Walk (non-seasonal, no trend)* ($\mathcal{RW}$):

$$Y_t = Y_{t-1} + \epsilon_t \tag{8}$$

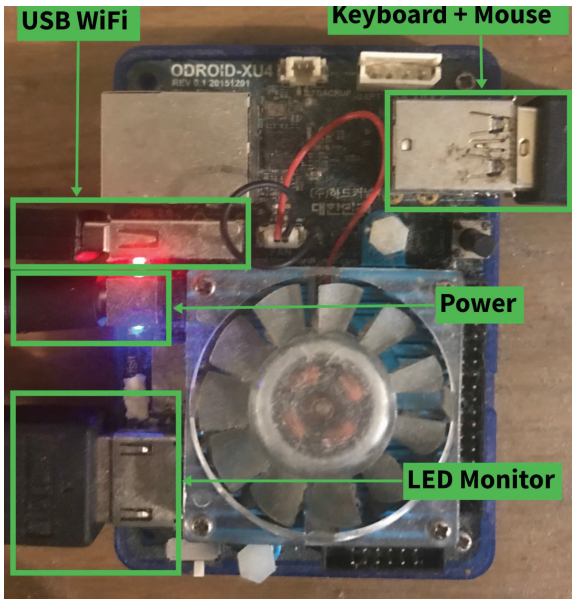


Fig. 2: The Odroid experimental setup including peripherals.

with $Y_0 = 1.0$ and $\epsilon_t \sim \mathcal{N}(0,1)$ is the gaussian white noise, where the current value of the series is only dependent on the previous timestep value. Note that random walk is different from Gaussian white noise.

TABLE I: Injected anomalies by type (percentages are w.r.t. the total no. of anomalies)

Dataset	#Anom.	Spikes	Dips	Coll. Anom.
$\mathcal{STD}$	7	43%	28.5%	28.5%
$\mathcal{RW}$	9	44%	44%	12%

Each synthetic dataset has 5,000 data points. The outliers are then injected with varying magnitudes and types as follows. The magnitudes randomly vary from 0.5σ to 6.0σ as in [12], where σ is the standard deviation of the dataset concerned. The types and their respective quantities are given in Table I. The positions of anomalies are random.

For **real-world datasets**, we use the Numenta Anomaly Benchmark dataset [24] (the *realKnownCause* category) and the Yahoo Anomaly Detection dataset [38] (the *A1 Benchmark* category). These univariate time series datasets incorporate complex dynamics which is hard to emulate using synthetic data, thereby allowing us to evaluate our model's overall performance more comprehensively.

Baselines. We compare our proposed model with SOTA neural network models dedicated to edge, as well as with popular time-series anomaly detectors. We categorize the baseline models into (a) *Neural Network-based approaches*, and (b) *Machine Learning-based approaches*, and describe them as follows:

a. *Neural Network-based approaches.* We consider two SOTA approaches that employ neural network-based detection architecture.

a.1) EPBRVFL-AE : *Bayesian Random Vector Functional Link AutoEncoder with Expectation Propagation* [20] is the latest model for detection of anomalies at the edge in a distributed manner and is a single layered neural network. We re-implemented EPBRVFL-AE [20] without the *expectation propagation* as originally proposed in [20], because we are only interested in detecting anomalies on a single edge device without inter-node communication overhead. From here onwards, we refer to this re-implemented EPBRVFL-AE simply as BRVFL-AE. All the hyperparameters are set as proposed in the paper [20].

a.2) ONLAD [18]: A single-layer feed-forward neural network model that was implemented directly on edge hardware for anomaly detection. We implement three different versions of ONLAD, each with $16, 64$, and 128 neurons in the hidden layer, respectively. The performance measures for ONLAD, as discussed in the later sections, are averaged over all the three models.

b. *Machine Learning-based approaches.* We consider three commonly used machine learning-based approaches and describe them briefly below.

b.1) ONE-CLASS SVM (OC-SVM): First introduced by [39] as a one-class classifier based on *support vector machine*, mostly used for *novelty detection*. Its robust performance of detecting *novelties* (anomalies), along with very easy-to-use approach has made it one of the top choices for anomaly detection.

b.2) LOCAL OUTLIER FACTOR (LOF): It is a density-based machine learning approach, developed by [2]. This approach is lightweight, along with good performance on detecting anomalies, thereby making it a suitable choice for many IoT/edge applications.

b.3) ISOLATION FOREST: This is a tree-based machine learning aproach, first proposed by [3]. A high anomaly detection performance, with very few parameters to save, makes it an overall excellent candidate for anomaly detection problems.

The hyperparameters for the machine learning-based approaches (OC-SVM, LOF, and IF) are set to their default values as by the SCIKIT-LEARN library.

While other SOTA deep learning-based anomaly detection approaches do exist, like ANNet [6] and LightLog [4], they are not developed to be trained directly at an edge site (device), due to their computational and space complexities. Hence, these models are not considered as baselines in our evaluation.

The baseline methods as well as our proposed LightESD are all implemented in PYTHON 3.10. Moreover, we compare the baselines against our proposed model at 95% (LightESD-1; $\alpha = 0.05$) and 99.9% (LightESD-2; $\alpha = 0.001$) confidence levels, where α is the significance level introduced in Section III-C1.

Evaluation Metrics. For a good understanding of model performance from different perspectives, we evaluate detection performance (Precision, Recall, and F_1-score), generality,

latency, resource utilization, and power consumption of all the different models.

By generality, we mean the ability to maintain a consistent performance across different types of data. For this purpose, we use the *coefficient of variation* (CV) as the metric, which is defined as the standard deviation divided by the mean of a principal metric, chosen as the F_1-score in our case.

The latency refers to the time taken from the model under consideration receives an input sample till the anomaly score is generated.

The device resource utilization is measured using an open-source tool *s-tui* [40], where the CPU and memory utilization for the processes executing the code for the anomaly detection model(s), are measured separately.

The power consumption is measured using a wall ammeter, in Watt (W). The power plug of the device board is plugged into the ammeter, which in turn is plugged into the wall power socket. By power consumption, we mean the increase in power consumption (in %) when the code for a model is run against an established baseline (the idle state running power), with an LED monitor, keyboard, mouse, and a USB ethernet, connected to the device board, as seen in Fig. 2. The idle state running power of the experimental setup was found to be 29.8 Watt.

New metric. We also propose a new, comprehensive metric called *ADCompScore*, which summarizes all the above metrics into a single quantity, in order to provide a convenient measure of the overall performance of an anomaly detection model, for quick decision-making regarding the algorithm's deployability on an edge device. It is defined as

$$ADCS = \frac{1}{\sum w_*}\Big[w_f \cdot f + w_g \cdot (1-g) + w_l \cdot (1-l) + w_c \cdot (1-c) + w_r \cdot (1-r) + w_p \cdot (1-p)\Big] \quad (9)$$

where f is the F_1-score, g is the model's coefficient of variation (CV), l is the *min-max* normalized latency, c is the CPU utilization (%), r is the RAM utilization (%), p is the increase in power consumption (%), and $w_* = \{w_f, w_l, w_c, w_r, w_p, w_g\}$ are the weights associated with the corresponding performance measures. The *ADCompScore*, or *ADCS* in short, is in the range of $[0, 1]$.

Except for the F_1-score, all the above performance measures are desired to be as low as possible in magnitude. Hence, we use the complement of those performance measures in our *ADCompScore* metric definition. The weights can be specified to prioritize different performance measures that tailor to different applications or requirements. The *ADCompScore* metric is defined in this general manner such that it can be easily applied to a wide range of scenarios. In our experiment, we have chosen to simulate the scenario where the different performance metrics are weighted equally, as this allows us to better understand the contribution of each individual performance metric towards the overall score. However, the design of this new metric incorporates flexibility by allowing different weights to be assigned depending on the actual situation at hand.

B. Results

1) Quantitative Performance: The comparison of the proposed approach against the baseline models, on precision and recall, which are subsumed by F_1-score, is provided in Table II. As we can observe from Table II, LightESD-1 and LightESD-2 outperform all the other baseline approaches in terms of anomaly detection performance. The ability of LightESD to remove multiple seasonalities (if present) and the trend, results in a much better extraction of the residual component which in turn allows for superior anomaly detection. In a nutshell, LightESD implements a simple yet effective approach to detect anomalies. Note that in the "Generality" column, a larger mean and a smaller CV are desired. The BRVFL-AE [20] approach comes closest to our proposed model, in terms of the detection performance, followed by ONLAD [18]. One of the reasons for such performance of these neural network-based models is that, adding more number of hidden layers may have a positive impact on their detection performances, but increasing the number of hidden layers will add to the training complexity, along with a significant increase in resource utilization and power consumption, which are not desirable for edge applications.

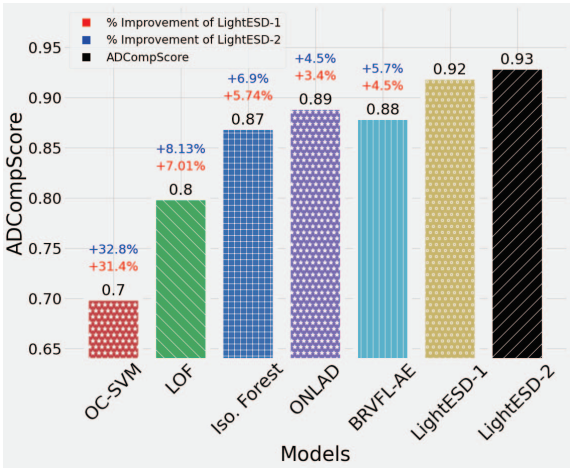


Fig. 3: *ADCompScore* of the different models.

Table III gives the performance on other metrics which are particularly relevant to edge computing. We see that the proposed approach, LightESD (both LightESD-1, and LightESD-2), performs well when compared with the other baseline methods. Note that LightESD does not need to store any weights/coefficients for the inference stage which is unlike the other methods, and this results in a low RAM utilization especially when compared to SOTA methods, as well as the lowest power consumption. SOTA approaches like ONLAD and BRVFL-AE, need to store their *weight matrices* and *bias vectors*, thereby adding extra overhead to memory utilization. However, observing Table III, we also see that both ONLAD and BRVFL-AE have better performance in

TABLE II: Detailed Performance Comparison on Detection

Models	$\mathcal{STD}$			$\mathcal{RW}$			NAB			Yahoo			**Generality**	
	Prec.	**Rec.**	**F1**	**Prec.**	**Rec.**	**F1**	**Prec.**	**Rec.**	**F1**	**Prec.**	**Rec.**	**F1**	**Mean**	**CV**
OC-SVM	0.51	0.71	0.59	0.5	0.72	0.59	0.50	0.64	0.56	0.52	0.69	0.59	0.58	0.03
LOF	0.62	0.65	0.63	0.58	**0.98**	0.73	0.52	0.7	0.6	0.61	0.77	0.68	0.66	0.1
Iso. For.	0.50	0.47	0.48	0.50	0.75	0.6	0.50	0.73	0.59	0.54	0.83	0.65	0.58	0.12
ONLAD	0.70	0.72	0.71	0.66	0.69	0.67	0.8	0.73	0.76	0.75	0.72	0.73	0.72	0.06
BRVFL-AE	0.67	0.76	0.71	0.7	0.88	0.78	0.79	0.8	0.79	0.8	0.73	0.76	0.76	0.05
LightESD-1	0.77	**0.82**	0.79	0.70	0.97	0.81	0.77	**0.83**	0.80	0.80	**0.88**	0.84	0.81	**0.02**
LightESD-2	**0.89**	0.78	**0.83**	**0.95**	0.97	**0.96**	**0.85**	**0.83**	**0.84**	**0.85**	0.87	**0.86**	**0.87**	0.06

TABLE III: Performance Comparison on Additional Factors on SBC (single-board computer).

Models	Latency (s)	CPU Utilization (%)	RAM Utilization (%)	Increase in Power Consumption (%)
OC-SVM	1.34	8.35	3.87	23.8
LOF	0.4	3.6	0.01	14.7
Iso. For.	0.25	3.3	0.01	14.5
ONLAD	0.19	8.87	3.11	16.3
BRVFL-AE	0.21	10.96	5.33	17.7
LightESD-1	0.24	5.47	3.29	**14.3**
LightESD-2	0.24	5.47	3.29	**14.3**

terms of latency, and our proposed approach falls short against the SOTA methods possibly due to the iterative nature of the proposed LightESD algorithm. While ML-methods like the Local Outlier Factor and Isolation Forest have very competitive performances in terms of CPU and RAM utilization, they fall short in terms of anomaly detection performance.

As we have observed so far, a model cannot perform in a superior manner in *all* aspects. In order to have a holistic view of the overall performance of all the models, we utilize the *ADCompScore* evaluation metric as defined in Eq. (9). Fig. 3 provides the overall performance of the proposed LightESD approach for both LightESD-1 and LightESD-2, as well as the baseline methods, using our proposed *ADCompScore* evaluation metric, where all the weights w_* are set to 1. We can see that, although BRVFL-AE [20] has a lower latency (as seen in Table III) as well as a good detection performance, it falls short in terms of the *ADCompScore* due to its much higher device resource utilization. Along similar lines, machine learning baseline methods like Isolation Forest and the Local Outlier Factor have very competitive device resource utilization and low power consumption, but fall short in their detection performance. Fig. 3 also quantifies the percentage improvement of the proposed, LightESD, anomaly detection model with respect to all the baseline methods, in terms of the overall *ADCompScore*. The proposed LightESD approach is able to beat the SOTA models (BRVFL-AE and ONLAD) by a margin of $3-6\%$, and the other ML-based models by more significant margins (up to 33%). Taking into account all these factors besides the anomaly detection performance, which are of great importance to rapid, on-device edge learning and deployment, LightESD is able to exceed the overall performance of the SOTA edge anomaly detectors (like BRVFL-AE and ONLAD), as well as popularly used ML-based detectors, thereby making it a much desirable anomaly detection method for Edge AI applications.

2) Effect of Significance Level: As we observe in Table II, decreasing the significance level, α, from 0.05 to 0.001, significantly increases the **Precision** of LightESD, without notably impacting the **Recall**. In other words, with this change, the proposed model is able to reduce the number of *false positives* (a.k.a. *false alarms*) without negatively impacting the number of *false negatives* (i.e., *missed detection* of true anomalies), which is desirable for different kinds of problems.

V. Conclusion

This paper proposes an anomaly detection framework that is able to detect anomalies directly at the edge site, without the need for any training at a central server. Unlike many statistical and machine learning methods, it is non-parametric and does not require any assumption of the underlying distribution of input data. It is also weight-free and does not require separate training and validation phases, as the model auto-fits/adapts to the underlying data directly on the fly to identify outlier data points. With a focus on time series applications, the proposed approach, LightESD, can tackle different types of data including seasonal, non-seasonal, and random walks. Our comprehensive evaluation demonstrates that LightESD outperforms both SOTA and other popular anomaly detectors by clear margins, as well as mitigates false alarms. It also generalizes much better across different datasets both on synthetic and real-world data. Moreover, LightESD consumes very low power and CPU/memory resources, when compared to SOTA anomaly detection schemes for the edge. These attributes make LightESD a desirable choice for rapid deployment directly at the edge, and for producing near real-time detection performance due to the low latency.

One limitation of LightESD is that it requires a batch of data for learning the underlying patterns, which would face challenge when there is only a single training instance in some circumstances. Thus in future work, we plan to enhance LightESD from a *batch learning* setting to a pure *online learning* setting.

REFERENCES

[1] B. Peirce, "Criterion for the rejection of doubtful observations," *The Astronomical Journal*, vol. 2, pp. 161–163, 1852.

[2] M. M. Breunig, H.-P. Kriegel, R. T. Ng, and J. Sander, "Lof: identifying density-based local outliers," in *Proceedings of the 2000 ACM SIGMOD International Conference on Management of data*, 2000, pp. 93–104.

[3] F. T. Liu, K. M. Ting, and Z.-H. Zhou, "Isolation forest," in *IEEE International Conference on Data Mining*, 2008, pp. 413–422.

[4] Z. Wang, J. Tian, H. Fang, L. Chen, and J. Qin, "Lightlog: A lightweight temporal convolutional network for log anomaly detection on the edge," *Computer Networks*, vol. 203, p. 108616, 2022.

[5] A. Geiger, D. Liu, S. Alnegheimish, A. Cuesta-Infante, and K. Veeramachaneni, "Tadgan: Time series anomaly detection using generative adversarial networks," in *2020 IEEE International Conference on Big Data (Big Data)*. IEEE, 2020, pp. 33–43.

[6] G. Sivapalan, K. K. Nundy, S. Dev, B. Cardiff, and D. John, "Annet: A lightweight neural network for ecg anomaly detection in iot edge sensors," *IEEE Transactions on Biomedical Circuits and Systems*, vol. 16, no. 1, pp. 24–35, 2022.

[7] J. An and S. Cho, "Variational autoencoder based anomaly detection using reconstruction probability," *Special Lecture on IE*, vol. 2, no. 1, pp. 1–18, 2015.

[8] J. Lu, A. Liu, F. Dong, F. Gu, J. Gama, and G. Zhang, "Learning under concept drift: A review," *IEEE transactions on knowledge and data engineering*, vol. 31, no. 12, pp. 2346–2363, 2018.

[9] N. Cristianini, J. Shawe-Taylor *et al.*, *An introduction to support vector machines and other kernel-based learning methods*. Cambridge university press, 2000.

[10] C. Cortes and V. Vapnik, "Support-vector networks," *Machine learning*, vol. 20, pp. 273–297, 1995.

[11] V. Chandola, A. Banerjee, and V. Kumar, "Anomaly detection: A survey," *ACM computing surveys (CSUR)*, vol. 41, no. 3, pp. 1–58, 2009.

[12] J. Hochenbaum, O. S. Vallis, and A. Kejariwal, "Automatic anomaly detection in the cloud via statistical learning," *arXiv preprint arXiv:1704.07706*, 2017.

[13] R. Zhu, X. Ji, D. Yu, Z. Tan, L. Zhao, J. Li, and X. Xia, "Knn-based approximate outlier detection algorithm over iot streaming data," *IEEE Access*, vol. 8, pp. 42 749–42 759, 2020.

[14] S. Mascaro, A. E. Nicholso, and K. B. Korb, "Anomaly detection in vessel tracks using bayesian networks," *International Journal of Approximate Reasoning*, vol. 55, no. 1, pp. 84–98, 2014.

[15] G. Pu, L. Wang, J. Shen, and F. Dong, "A hybrid unsupervised clustering-based anomaly detection method," *Tsinghua Science and Technology*, vol. 26, no. 2, pp. 146–153, 2020.

[16] G. Pang, C. Shen, and A. van den Hengel, "Deep anomaly detection with deviation networks," in *Proceedings of the 25th ACM SIGKDD International Conference on knowledge discovery & data mining*, 2019, pp. 353–362.

[17] T. Schlegl, P. Seeböck, S. M. Waldstein, G. Langs, and U. Schmidt-Erfurth, "f-anogan: Fast unsupervised anomaly detection with generative adversarial networks," *Medical image analysis*, vol. 54, pp. 30–44, 2019.

[18] M. Tsukada, M. Kondo, and H. Matsutani, "A neural network-based on-device learning anomaly detector for edge devices," *IEEE Transactions on Computers*, vol. 69, no. 7, pp. 1027–1044, 2020.

[19] N.-Y. Liang, G.-B. Huang, P. Saratchandran, and N. Sundararajan, "A fast and accurate online sequential learning algorithm for feedforward networks," *IEEE Transactions on neural networks*, vol. 17, no. 6, pp. 1411–1423, 2006.

[20] M. Odiathevar, W. K. Seah, and M. Frean, "A bayesian approach to distributed anomaly detection in edge ai networks," *IEEE Transactions on Parallel and Distributed Systems*, vol. 33, no. 12, pp. 3306–3320, 2022.

[21] P. Welch, "The use of fast fourier transform for the estimation of power spectra: a method based on time averaging over short, modified periodograms," *IEEE Transactions on audio and electroacoustics*, vol. 15, no. 2, pp. 70–73, 1967.

[22] D. Kurtz, "An algorithm for significantly reducing the time necessary to compute a discrete fourier transform periodogram of unequally spaced data," *Monthly Notices of the Royal Astronomical Society*, vol. 213, no. 4, pp. 773–776, 1985.

[23] M. Vlachos, P. Yu, and V. Castelli, "On periodicity detection and structural periodic similarity," in *Proceedings of the 2005 SIAM International Conference on data mining*. SIAM, 2005, pp. 449–460.

[24] A. Lavin and S. Ahmad, "Evaluating real-time anomaly detection algorithms–the numenta anomaly benchmark," in *2015 IEEE 14th International Conference on machine learning and applications (ICMLA)*. IEEE, 2015, pp. 38–44.

[25] S. Y. Shah, D. Patel, L. Vu, X.-H. Dang, B. Chen, P. Kirchner, H. Samulowitz, D. Wood, G. Bramble, W. M. Gifford *et al.*, "Autoai-ts: Autoai for time series forecasting," in *Proceedings of the 2021 International Conference on Management of Data*, 2021, pp. 2584–2596.

[26] Q. Wen, J. Gao, X. Song, L. Sun, and J. Tan, "Robusttrend: a huber loss with a combined first and second order difference regularization for time series trend filtering," in *Proceedings of the 28th International Joint Conference on Artificial Intelligence*, 2019, pp. 3856–3862.

[27] Q. Wen, Z. Zhang, Y. Li, and L. Sun, "Fast robuststl: Efficient and robust seasonal-trend decomposition for time series with complex patterns," in *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2020, pp. 2203–2213.

[28] Y. Sun, P. Babu, and D. P. Palomar, "Majorization-minimization algorithms in signal processing, communications, and machine learning," *IEEE Transactions on Signal Processing*, vol. 65, no. 3, pp. 794–816, 2016.

[29] J. Thompson, "An empirical evaluation of denoising techniques for streaming data," Lawrence Livermore National Lab.(LLNL), Livermore, CA (United States), Tech. Rep., 2014.

[30] Q. Wen, J. Gao, X. Song, L. Sun, H. Xu, and S. Zhu, "Robuststl: A robust seasonal-trend decomposition algorithm for long time series," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, no. 01, 2019, pp. 5409–5416.

[31] S. M. Ross *et al.*, "Peirce's criterion for the elimination of suspect experimental data," *Journal of engineering technology*, vol. 20, no. 2, pp. 38–41, 2003.

[32] G. L. Tietjen and R. H. Moore, "Some grubbs-type statistics for the detection of several outliers," *Technometrics*, vol. 14, no. 3, pp. 583–597, 1972.

[33] B. Rosner, "Percentage points for a generalized esd many-outlier procedure," *Technometrics*, vol. 25, no. 2, pp. 165–172, 1983.

[34] F. E. Grubbs, "Sample criteria for testing outlying observations," *The Annals of Mathematical Statistics*, pp. 27–58, 1950.

[35] P. J. Rousseeuw and C. Croux, "Alternatives to the median absolute deviation," *Journal of the American Statistical association*, vol. 88, no. 424, pp. 1273–1283, 1993.

[36] B. Iglewicz and D. Hoaglin, "Volume 16: how to detect and handle outliers, the asqc basic references in quality control: statistical techniques, edward f. mykytka," Ph.D. dissertation, Ph. D., Editor, 1993.

[37] P. J. Huber, "Robust statistics," in *International encyclopedia of statistical science*. Springer, 2011, pp. 1248–1251.

[38] N. Laptev, S. Amizadeh, and Y. Billawala, "S5-a labeled anomaly detection dataset, version 1.0 (16m)," 2015.

[39] B. Schölkopf, R. C. Williamson, A. Smola, J. Shawe-Taylor, and J. Platt, "Support vector method for novelty detection," *Advances in neural information processing systems*, vol. 12, 1999.

[40] Manuskin, "GitHub - amanusk/s-tui: Terminal-based CPU stress and monitoring utility — github.com," https://github.com/amanusk/s-tui, 2017.