Concurrent Deep Distributed Anomaly Detection Model for Embedded Systems

Mellitus O. Ezeme, *Member, IEEE,* Qusay Mahmoud, *Senior Member, IEEE,* and Akramul Azim, *Senior Member, IEEE*

***Abstract*—System logs and traces contain information that reflects the state of the System and provides a comprehensive insight into the workings of the embedded system. Moreover, logging of traces as a tool for monitoring the operation of an embedded system is recommended by most safety standard organizations. However, because the data can be overwhelmingly huge within a short space of time, the use of models that do not rely only on known signatures for online anomaly detection becomes difficult to use due to the challenge of processing such an enormous amount of data at runtime. Hence, the resort to the use of signature-based tools by most practitioners as an embedded system is inherently limited in memory and computation power.**

In this paper, we introduce a distributed anomaly detection model that smartly identifies the requirements of an application and uses the information to intelligently provide simultaneous anomaly detection for multiple profiles by leveraging the cloud computing platform as an additional computational resource when needed. Intelligent decisions about *which* part of the application runs *where* is a function of both the embedded process *requirements* and resource *availablity* in the embedded platform. Our anomaly detection model is a multi-stage process which makes use of transfer learning by using the knowledge gained from the unsupervised deep learning part to the supervised clustering portion of the model. Long Short-Term Memory (LSTM) networks fitted with a recursive context input and a context-aware attention mechanism provide a learning process that traverses horizontally (present and past inputs) and vertically (present inputs). We test the performance of the anomaly detection model with publicly available datasets of multiple processes running concurrently and our model demonstrates both effectiveness and robustness in detecting the anomalous sequences while satisfying the timing and other constraints.

***Index Terms*—Anomaly detection, embedded system, distributed system, deep learning**

I. Introduction

From the Internet of Things (IoT) to the Internet of Everything (IoE), the common denominator is *connectivity* driven by ubiquitous embedded systems. The goal of these devices and infrastructures is to build a cyber-physical framework that predicts and automates the mundane, enabling people to concentrate on more productive and creative things [1]. This digital infrastructure puts sensors on everything (animate and inanimate), connects the sensors, think about what the *things* are saying, analyzes the data generated by the *things* to create a knowledge-base, and finally makes predictions or takes action based on that information [2]. In this way, it produces a new value in the form of time which is the positive aspect. On the downside, this increased connectivity era implies that an automated system handles both our safety and non-safety critical data and actions, and a breach in the intended behavior of the connected devices could prove catastrophic as they manage our daily activities from medicine to autonomous vehicles avionics and power systems. Hence, the need for an online anomaly model that can be integrated into these devices to monitor the conformity of the behavior of the devices with the prescribed operational standards. When the behavior of the devices and applications are well-characterized, then the state transition checks espoused by the authors in [3], [4] can be applied to detect the anomalies. However, there are two challenges associated with the tools in this data and connectivity explosion age; **a)** it is daunting to define all the possible states of the tuples associated with a particular process running in the embedded system because of the increasing complexity of tasks performed by these processes. **b)** the increasing complexity of the functions [2] performed by these cyber-physical embedded systems demands a high level of dynamism which makes the use of static state transition analysis untenable. Therefore, we propose a dynamic anomaly detection model that exploits the temporal information in the system call execution sequences to detect anomalies.

Our model processes the traces as sequential tuples and takes into account the temporal drift by building a profile that can create both short and long-term contexts of the sequences. The use of the LSTM cells and context-based attention layer provides medium to short-term contexts while the incorporation of the feedback input from the attention layer creates a long-range dependency between the present and past inputs to the model. This dynamic approach enables us to target both identified and unseen anomalies and deepens the understanding of the execution contexts of the kernel events *horizontally* and *vertically*. This tool employs the concept of transfer learning as we start off with an unsupervised model architecture and use the knowledge base from the unsupervised learning phase to create a fast supervised classifier to detect when an anomaly has occurred. The unsupervised phase of the model uses only the traces from the *normal* operating states while the supervised part makes use of a fraction of anomalous data and the normal profile data. Some statistical methods that use exponential weighted moving average [5] or cumulative sum [6] can be useful, but their effectiveness is limited to the traces where there is no randomness. Trace tuples from the kernel layer containing interrupts make predictability with [5], [6] ineffective. Also, anomaly detection models based on

Mellitus O. Ezeme, Qusay Mahmoud and Akramul Azim are with the Department of Electrical, Computer and Software Engineering, University of Ontario Institute of Technology, Oshawa ON, L1H 7K4 Canada.
E-mail: {mellitus.ezeme,qusay.mahmoud,akramul.azim}@uoit.net

entropy like [7] does not fit this kind of analysis because it relies on volume to detect an anomaly rather than the temporal information conveyed by the tuples. Also, the variants of the vector space models used in [8], [9], [**?**] cannot be used for online anomaly detection because it performs classification based on a large buffer of the traces.

To facilitate the capture of long and short-term temporal dependencies, we use a stack of the Long Short-Term Memory (LSTM) [10] which its variants have been used in various tasks to demonstrate its effectiveness in capturing complex non-linear relationships that exist in a sequence. The use of hierarchical LSTM learns the temporal relationships amongst events while the attention layer determines in small details, the impact of each feature in one another. This strategy helps to filter out the effect of the randomness prevalent in system logs. Our anomaly model uses a context-aware variant of the attention mechanism which does three functions; **a)** within tuples in a window under consideration, it improves the target tuple prediction accuracy by diminishing the effect of unessential source inputs for any target output. **b)** it narrows the dimension of the hidden state output of the LSTM and introduces flexibility in handling the size of the output vector. **c)** between predictions, the attention layer controls the influence of the previous output on the next target by forming a component of the context vector that controls the alignment of the next prediction. And this helps to answer the *why* question in the computation of the prediction by giving us a view of the *features* that influence each output as demonstrated in [11]. The logic in our reasoning is that just like natural language, each tuple has different contexts based on usage and the effect of the present tuple V_t on next prediction V_{t+1} should be derived from a deeper and longer context than just the present tuple V_t and current attention vector Z_t because concurrently running tasks can make the order of the traces complex to analyze. Therefore, in comparison with other design architectures, our design varies on how the context vector is constructed and used in both the attention layer and next target prediction. And this is one of the principal technical contributions of the work.

The primary target of the work is system logs from operating systems like kernel traces or system calls, but we will also test with other types of log files to demonstrate the versatility of the model. To handle bias in the model, in each layer of the system which generates the logs, we train the model only with attributes that cut across the different formats emitted by the different kinds of operating systems for the layer under consideration. Example of the characteristics of system calls sequence is open $\longrightarrow$ mmap $\longrightarrow$ read $\longrightarrow$ close and these are used as features to train the model and get the classifier for this layer. Therefore, we state the problem as thus: *given logs obtained during the normal operation of a system, is it possible to create a model that uses the information solely from normal behavior to characterize the standard and deviant behaviors in the system logs?*

To answer the question posed above, we propose the model designed using LSTM networks with attention to detect anomalies in log sequences. In summary, our contributions are: **a)** we present a deep context-aware architecture for anomaly detection in semi-structured sequences with bias to system logs. **b)** we demonstrate the significance of using the attention layer to provide rich meanings that help to identify the nonlinear high dimensional associations inherent in system logs sequences.

To ease the comprehension of the work, we have divided the rest of the work into the following sections; Section II briefly highlights the related work in this domain. Section III discusses the technical details of our model while Section IV details our tests as well as discussion of the results. In Section V, we conclude the work with an insight into our future research directions.

II. Related Work

In both intrusion and anomaly detection systems, two strategies have been adopted [12]. The first method depends on the signatures of known anomalies to construct a signature database. Moreover, filtering of incoming signatures is done by allowing only messages which do not match any pattern in the database to go through. Authors in [13] detail this approach very well. The obvious limitation of this method is that *zero-day* vulnerabilities cannot be detected as it only searches for known signatures. On the other hand, Authors in [8], [14], [9], [15] use the second approach which involves the construction of a model to target both seen and unseen anomalies. The core principle of these approaches consists of the extraction of features from the operational profile of the system to construct models which can differentiate normal and anomalous behavior. While it is versatile regarding its threat target coverage, its performance comes at the cost of having a higher false positive than the signature-based methods.

In [8], a vector space model is used with dendrograms to construct a threshold to distinguish different operational states from the normal states with excellent results but it is solely an offline based classifier, and scalability will be complicated because it consumes the whole observed sequences before making a decision. The authors of [14] built an anomaly detection framework called *Deeplog* using two layers of LSTM networks and a workflow construction approach for each key in the log for diagnostics. This *Deeplog* model uses same LSTM cells like ours, but there is no notion of attention layer in the framework. Also, the authors of [7] used the statistical metric of entropy to implement an anomaly detection model for network logs but this type of anomaly model is best suited for cases where the volume of logs determine if an anomaly has occurred like denial of service attack. In [16], a real-time systems anomaly detection model is designed using the principle of inter-arrival curves to detect anomalous traces in a log sequence. However, this inter-arrival curve-based model works offline because it requires a large number of logs before it computes the curves. Reference [9] designed a vector space model to mine console logs for anomalies and [4] used an optimization method of minimum debugging frontier sets to create a model for detection of errors/faults in software execution.

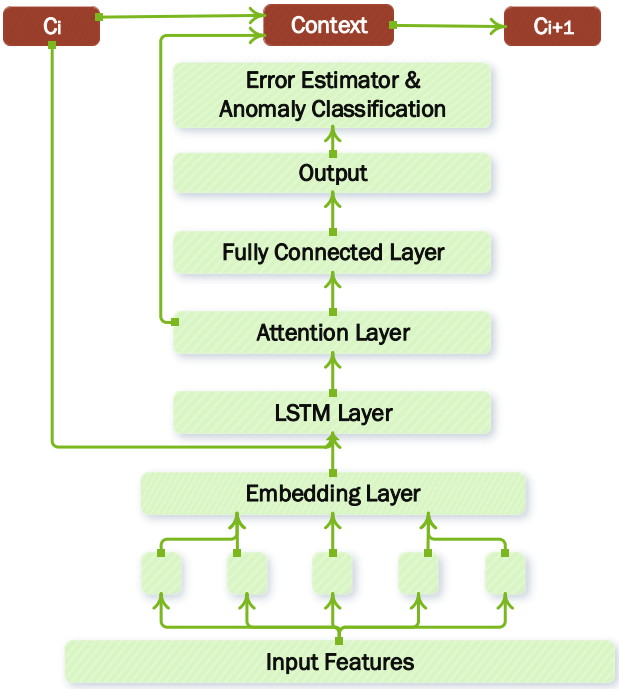

Fig. 1. Architecture of the Recursive Deep Context Anomaly Detection Model

III. Model Outlines

In the following subsections, we describe the working details of our model design and the reasoning behind the architectural decisions we made. The simplified diagram of Fig. 1 shows the complete model design for one timestep. The *unrolling* of the network computes the prediction for subsequent timesteps.

A. Log Preprocessing

The logs are in the form of alpha-numeric tuples, and we perform some preprocessing before being loaded them into the network. The processing includes extracting the features of interest. As we mentioned in Section I, for trace logs in the same layer, we only process the attributes which are common across the trace irrespective of the system to remove any system-induced bias on the classifier. In kernel, an example event stream of THRUNNING $\longrightarrow$ THREADY $\longrightarrow$ THRECEIVE $\longrightarrow$ THREPLY has *timestamps, PID, TID, NID*, etc. which are common across the event stream. Therefore, we formulate the characteristics of interest using these features which exist across kernel event streams irrespective of platform. We will describe specific features of interest in Section IV as we deal with each of the datasets used in the experiments.

B. Embedding Layer

This component creates a non-sparse vector representation of the features instead of the *onehot* vector encoding of the tuples, which makes the neural learning process difficult. We create a set of the features in each scenario, and this forms our vocabulary. The *word2vec* method discussed in [17] is used to create the word to vector representations. This block eliminates sparsity and improves the extraction of the nonlinear temporal relationships inherent in the sequence of the observations. The input to this block is a scalar integer v and the output is a vector $\vec{u} \in \mathbb{R}^n$ where n represents the number of dimensions the scaler v is projected to. Finally, we reverse the source sequence to create short-term dependencies in the learning process as discussed in [18].

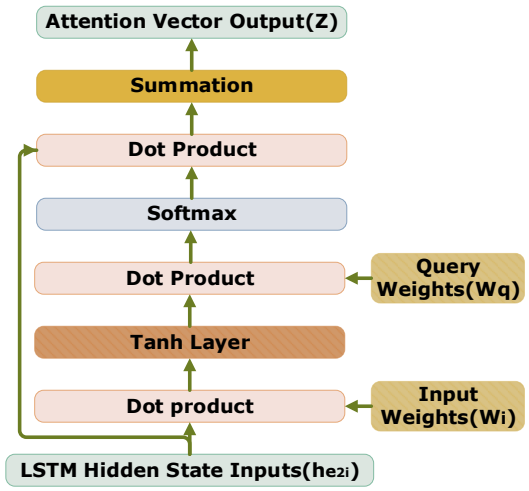

Fig. 2. Context-aware Attention Network for the Model

C. LSTM Layer

Our LSTM layer is Bidirectional stack of two LSTM networks to improve the discerning of nonlinear temporal relationships of the tuples. The authors of [19] have shown the importance of hierarchy in decoding relationships amongst tuples. We feed the embedding layer vector $\vec{u} \in \mathbb{R}^n$ to the first layer of the LSTM layers and feed a reversed input to the second layer of the LSTM. The outputs from the two layer are concatenated and fed to the attention layer during training. Different kinds of recurrent networks can be used at this layer but we use the LSTM cells described in [10] to create our LSTM Layer.

$$h_{e_1 i} = k\left(x'_i, h_{e_1 i-1}\right) \tag{1}$$

$$h_{e_2 i} = k\left(h_{e_2 i-1}, h_{e_1 i}\right) \tag{2}$$

D. Attention

Attention layers come in broadly two flavors: *soft* and *hard* attention. The soft-attention uses weighted outputs of the input to *attend* while hard-attention randomly selects a subset of the input to *attend*. Each has its advantage and disadvantages, but we focus on the soft-attention method in this work. We sacrifice the efficiency of computation by using the weighted sum of all source inputs, as this helps the model to learn efficiently using backpropagation with gradient descent during training. Differing from [20] that uses memory to create context, we add a query weight W_q that is learned

during training to ensure that each tuple is not attended to by just its occurrence in the present input sequence only but also based on its context throughout the training period. This query performs the similar role as the term-frequency inverse document frequency used to weigh the occurrence of tuples in the vector space model.

Fig. 2 shows the connections of the attention layer block of Fig. 1. The inputs the this layer are the input weights W_i and the second layer encoder outputs h_{e2i}. We pass the encoder output via a $\tanh$ layer after being scaled by the input weights W_i and the input bias vector b_i to generate correlation vectors m_i given in Equation 3.

$$m_i = \tanh(W_i \cdot h_{e_2 i} + b_i) \tag{3}$$

This correlation vector (Equation 3) represents the effect of each input based on the present. Hence, we multiply it with the query vector W_q which has the global knowledge of each input tuple in the present input sequence to provide deep horizontally spanning inputs for the inference process as shown in Equation 4. This vector is then passed through a softmax layer to generate s_i in Equation 5. This normalized values is scaled by the input vectors $h_{e_2 i}$ and summed to generate the attention vector Z in Equation 6.

$$a_i = W_q \cdot m_i + b_q \tag{4}$$

$$s_i = \left(\frac{e^{a_i}}{\sum_j e^{a_j}} \right)_i \tag{5}$$

$$Z = \sum_{i=1}^{n} s_i \times h_{e_2 i} \tag{6}$$

E. Fully Connected Layer

Our significant architectural change from the previous *encoder-decoder* frameworks is on what constitutes the input to the decoder. In most architectures that are recursive including that of [21], the past output target V_{t-1} only is the context which forms part of the input into the decoder for predicting V_t. In our framework, we reason that while such V_{t-1} only as the context may be sufficient for natural languages which are more structured, logs which are semi-structured require more horizontal traversing contexts during the decoding process to improve the accuracy of the prediction. Hence, our decision to use the Z_{t-1} and V_{t-1} as context for decoding V_t. Because the problem is formulated as a sequence-to-sequence problem, we posit that while V_{t-1} provides short term contexts in the decoder, the Z_{t-1} provides the much needed long term context required for accurate decoding of the output. In this design, h_{di-1}, V_{i-1} provide the context about decoded outputs while Z, Z_{i-1} provide the contexts about current and past attended inputs. For a sequence reconstruction task like ours, the conditional probability is given in Equation 7 where k is a nonlinear function.

$$P\left(\vec{V}\right) = \prod_{i=1}^{I} k\left(h_{dt-1}, V_{t-1}, Z, Z_{t-1}\right) \tag{7}$$

The decoder output of Equation 7 is then passed through a fully-connected layer which produces the predicted output V_t in the t timestep.

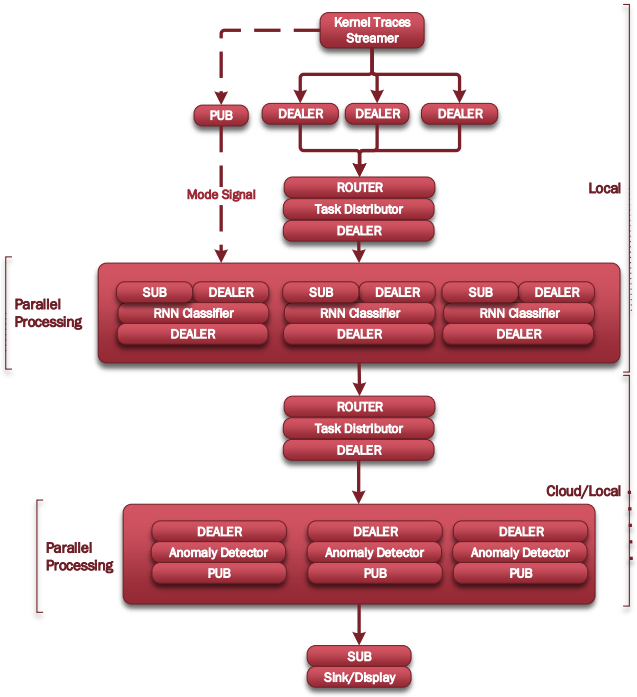

Fig. 3. Deep Anomaly Model Showing both Concurrency and Decomposition

F. Anomaly Detection

Given $\vec{x} \in \mathbb{Z}^p$ which serves as the input and target sequences, we aim to regenerate the $\vec{x}$ at the output by creating $\vec{V} \in \mathbb{Z}^p$. Therefore, the perfect result is when $\vec{x}_k \equiv \vec{V}_k$ but this is hardly feasible because of the high randomness caused by interrupts and other events in the log traces. Hence, when we perform $f : \vec{x} \mapsto \vec{V}$ given $\vec{x}$ as the ground truths, the deviation $\vec{d} = \left| \vec{x} - \vec{V} \right|$ is the difference between the ground truth and the predicted value for the given sequence. This deviation becomes the prediction error values which we process further to decide if an anomaly has occurred or not. In our design, we use only the logs/traces from normal operating conditions of the system for training. To determine if an anomaly has occurred, we deviate from the approaches of [22], [23] by avoiding the use of threshold as that would form a hyperparameter which may be difficult to tune by users of the model. Rather, we use the prediction errors to model a Gaussian probability distribution and use confidence interval (CI) to determine if an anomaly has occurred. This CI method makes it easier for a non-professional to use and vary the *CI* according to the safety level of an application.

G. Distributed Model

Our model provides concurrency for multiple processes, and we distribute it between the embedded platform and a server. There are multiple ways of achieving parallelism, but we settled for the use of *ZeroMQ* [24] as it helps to reduce the number of low-level use of locks, mutexes, and semaphores to provide conflict-free synchronization with no data corruption. *Dealer, Router, Sub, Pub* shown in Figure 3 are communication

sockets and some of them are capable of asynchronous duplex communication.

IV. Experiments and Results

V. Conclusions and Future Work

References

[1] M. K. Weldon, *The future X network: a Bell Labs perspective*. CRC press, 2016.

[2] M. O. Ezeme, "A multi-domain co-simulator for smart grid: Modeling interactions in power, control and communications," Ph.D. dissertation, University of Toronto (Canada), 2015.

[3] W. N. Sumner and X. Zhang, "Comparative causality: Explaining the differences between executions," in *Proceedings of the 2013 International Conference on Software Engineering*. IEEE Press, 2013, pp. 272–281.

[4] F. Li, Z. Li, W. Huo, and X. Feng, "Locating software faults based on minimum debugging frontier set," *IEEE Transactions on Software Engineering*, vol. 43, no. 8, pp. 760–776, 2017.

[5] N. Ye, S. Vilbert, and Q. Chen, "Computer intrusion detection through ewma for autocorrelated and uncorrelated data," *IEEE transactions on reliability*, vol. 52, no. 1, pp. 75–82, 2003.

[6] V. A. Siris and F. Papagalou, "Application of anomaly detection algorithms for detecting syn flooding attacks," in *Global Telecommunications Conference, 2004. GLOBECOM'04. IEEE*, vol. 4. IEEE, 2004, pp. 2050–2054.

[7] Y. Gu, A. McCallum, and D. Towsley, "Detecting anomalies in network traffic using maximum entropy estimation," in *Proceedings of the 5th ACM SIGCOMM conference on Internet Measurement*. USENIX Association, 2005, pp. 32–32.

[8] M. Ezeme, A. Azim, and Q. H. Mahmoud, "An imputation-based augmented anomaly detection from large traces of operating system events," in *Proceedings of the Fourth IEEE/ACM International Conference on Big Data Computing, Applications and Technologies*. ACM, 2017, pp. 43–52.

[9] W. Xu, L. Huang, A. Fox, D. Patterson, and M. Jordan, "Largescale system problem detection by mining console logs," *Proceedings of SOSP'09*, 2009.

[10] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[11] W. Chan, N. Jaitly, Q. Le, and O. Vinyals, "Listen, attend and spell: A neural network for large vocabulary conversational speech recognition," in *Acoustics, Speech and Signal Processing (ICASSP), 2016 IEEE International Conference on*. IEEE, 2016, pp. 4960–4964.

[12] V. Chandola, A. Banerjee, and V. Kumar, "Anomaly detection: A survey," *ACM computing surveys (CSUR)*, vol. 41, no. 3, p. 15, 2009.

[13] P. Garcia-Teodoro, J. Diaz-Verdejo, G. Maciá-Fernández, and E. Vázquez, "Anomaly-based network intrusion detection: Techniques, systems and challenges," *computers & security*, vol. 28, no. 1-2, pp. 18–28, 2009.

[14] M. Du, F. Li, G. Zheng, and V. Srikumar, "Deeplog: Anomaly detection and diagnosis from system logs through deep learning," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2017, pp. 1285–1298.

[15] X. Yu, P. Joshi, J. Xu, G. Jin, H. Zhang, and G. Jiang, "Cloudseer: Workflow monitoring of cloud infrastructures via interleaved logs," in *ACM SIGPLAN Notices*, vol. 51, no. 4. ACM, 2016, pp. 489–502.

[16] M. Salem, M. Crowley, and S. Fischmeister, "Anomaly detection using inter-arrival curves for real-time systems," in *Real-Time Systems (ECRTS), 2016 28th Euromicro Conference on*. IEEE, 2016, pp. 97–106.

[17] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," *In Proceedings of Workshop at ICLR*, 2013.

[18] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," in *Advances in neural information processing systems*, 2014, pp. 3104–3112.

[19] Z. Yang, D. Yang, C. Dyer, X. He, A. Smola, and E. Hovy, "Hierarchical attention networks for document classification," in *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, 2016, pp. 1480–1489.

[20] G. Salton, R. Ross, and J. Kelleher, "Attentive language models," in *Proceedings of the Eighth International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, vol. 1, 2017, pp. 441–450.

[21] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," *arXiv preprint arXiv:1409.0473*, 2014.

[22] P. Malhotra, A. Ramakrishnan, G. Anand, L. Vig, P. Agarwal, and G. Shroff, "Lstm-based encoder-decoder for multi-sensor anomaly detection," *arXiv preprint arXiv:1607.00148*, 2016.

[23] P. Malhotra, L. Vig, G. Shroff, and P. Agarwal, "Long short term memory networks for anomaly detection in time series," in *Proceedings*. Presses universitaires de Louvain, 2015, p. 89.

[24] F. Akgul, *ZeroMQ*. Packt Publishing, 2013.