# LOG ANOMALY DETECTION ON EUXFEL NODES

A. Sulc*, A. Eichler, T. Wilksen, DESY, Hamburg, Germany

*Abstract*

This article introduces a method to detect anomalies in the log data generated by control system nodes at the European XFEL accelerator. The primary aim of this proposed method is to provide operators a comprehensive understanding of the availability, status, and problems specific to each node. This information is vital for ensuring the smooth operation. The sequential nature of logs and the absence of a rich text corpus that is specific to our nodes poses significant limitations for traditional and learning-based approaches for anomaly detection. To overcome this limitation, we propose a method that uses word embedding and models individual nodes as a sequence of these vectors that commonly co-occur, using a Hidden Markov Model (HMM). We score individual log entries by computing a probability ratio between the probability of the full log sequence including the new entry and the probability of just the previous log entries, without the new entry. This ratio indicates how probable the sequence becomes when the new entry is added. The proposed approach can detect anomalies by scoring and ranking log entries from EuXFEL nodes where entries that receive high scores are potential anomalies that do not fit the routine of the node. This method provides a warning system to alert operators about these irregular log events that may indicate issues.

## INTRODUCTION

The stability and reliability of the European XFEL facility are essential for a successful operation. To facilitate this, a network of watchdog nodes is continuously monitoring the health state of the facility's essential components. These nodes, numbering in the hundreds, act as monitoring technology, ensuring the proper functionality of crucial European XFEL accelerator elements. Within their logs lie valuable information about the health state that can signal any potential problems with specific components or parts that could impact the entire facility. Automating the costly task of monitoring these lengthy and often redundant logs becomes especially important in guaranteeing the optimal performance of all nodes. The logs contain a wealth of information concerning the system's status, encompassing error messages, anomalies, and other factors that could affect the system or its associated components. By exploiting language embedding and anomaly detection techniques on these logs, we can efficiently identify and address issues or errors at the earliest possible stage when they occur in logs. This proactive approach empowers us to pinpoint potential problems before they escalate, enabling prompt measures to be taken to resolve ongoing issues. Furthermore, it facilitates timely intervention and the implementation of preventive measures to mitigate potential problems from arising. Monitoring the logs of the watchdog nodes by textual analysis of their logs not only provides an automated means of comprehending the European XFEL accelerator system conditions but also enables early detection and resolution of issues that would otherwise only gain significance in the event of a specific node failure.

The structure of the paper is the following: First, we summarize the related work in log anomaly detection. In the next section, we show four main steps of our approach with important justifications and examples. Lastly, we show several examples and sketch a potential future work in this field.

## RELATED WORK

A common approach to detecting anomalies in logs is to manually define rule-based systems. For example, Cinque et al. [1] and Yen et al. [2] have developed rule-based methods that scan logs for predefined patterns indicative of anomalies. However, these approaches rely heavily on expert knowledge to construct effective rules, which can be labor-intensive. To overcome this limitation, more automated techniques have emerged leveraging machine learning to discover anomalies.

With the increasing popularity of machine learning (ML) models, deep learning-based approaches gave the potential to perform a thorough log analysis under the presence of a large log corpus, often also accompanied by laboriously made labels. Long-term-short-term (LSTM) recurrent neural networks [3–5] turned out to be popular for log-anomaly detection due to its ability to handle sequential data. Recently transformers [6] were deployed in training to detect anomalies in logs [7]. In [7] they used a BERT [8] model for log-anomaly detection. However, their reliance on large training datasets and millions of parameters can limit their applicability in resource-constrained scenarios like ours. For a more comprehensive survey of ML log analysis, see [9].

Bertero et al. [10] propose an approach that treats logs as natural text and leverages word vector `Word2Vec` representations [11, 12] to perform automated word embedding. This technique maps words to a vector space, enabling the use of off-the-shelf classifiers for anomaly detection. A major drawback is that their approach still relies on manual labeling to train the classifier, which can be prohibitively expensive in our scenario. Additionally, they treat each log entry independently, ignoring the sequential nature of consecutive log message relationships. To mitigate the need for labeled data, other works like [13, 14] have explored unsupervised learning techniques. These methods apply text mining to logs and employ clustering approaches to identify anomalies without relying on manual labels. However,

* antonin.sulc@desy.de

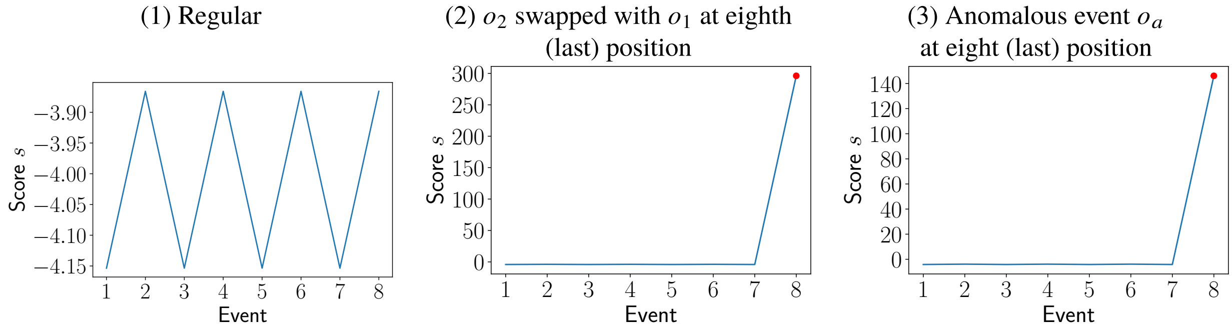


Figure 1: To demonstrate the anomaly detection capabilities of our HMM model, we examine three cases on a synthetic log sequence containing observable events $o_1$, $o_2$, and an anomalous event $o_a$. The HMM parameters were estimated using the sequence excluding the last entry (the minimum required to avoid overfitting). First (left), a sequence with a repetitive $o_1$, $o_2$ pattern where the HMM likelihood score $s$ fluctuates around very low values, as expected. Second (center), swapping $o_2$ and $o_1$ in the last position disrupts the pattern, increasing the score $s$ noticeably. Finally, inserting the anomalous $o_a$ event at the end causes a substantial $s$ spike, clearly detecting the improbable observation (right). This shows that the model detects anomalies from both unlikely or novel log messages and unexpected sequencing of normal events. Small disruptions in patterns or particularly improbable observations increase the model's likelihood scores.

they still consider logs in isolation rather than leveraging contextual information across log sequences.

In this work, we propose an alternative approach to detect anomalies without any labels or extensive user intervention. Our method is designed to adapt to novel log messages while also capturing the sequential nature of log analysis, overcoming the limitations of prior techniques. Specifically, we faced challenges from the limited diversity of log entries, which does not provide sufficient training data for standard ML models. Inspired by [10], we employ word vector embeddings to represent log entries in a high-dimensional space, mitigating data scarcity. However, instead of relying on supervised classifiers, we take a sequential modeling approach by treating logs from each source as temporal event streams. Our key insight is to focus on modeling patterns of occurrences within these streams, rather than just individual log entries. To achieve this, we introduce an unsupervised technique based on Hidden Markov Models operating on the embedded log sequences. By learning sequential regularities, anomalies can be detected from deviations in context rather than content. This probabilistic approach requires estimating only a minimal number of parameters, enabling robust detection even with limited training data.

## METHOD

In this section, we explain our proposed approach for scoring individual log entries to detect anomalies. The approach involves four main steps. First, we perform pre-processing on the raw log entries to replace redundant patterns and minimize the effect of unique token sparsity. Pre-processing transforms the text into consistent tokenized forms. Next, we generate embeddings for each log entry using `Word2Vec`. We calculate a mean vector of the word vectors for all terms in the entry. This provides a dense numeric representation capturing the contextual meaning of individual words in the log entry. Third, we fit an HMM model on sequences of these log entry embeddings from past observed logs. The HMM learns a probability distribution over likely sequences of log entries. Finally, we score new log entries by computing their probability under the trained HMM. Low probability entries deviating from the learned sequential patterns are identified as anomalies. The key advantage of our approach is that it relies solely on sequence modeling of log embeddings, without needing content analysis or keyword matching rules.

### *Preprocessing and Tokenization*

In this section, we detail the preprocessing steps applied to the raw log text before analysis.

First, we separated the log entries by identifying timestamp delimiters and newline characters in the raw messages. This extracted the individual log entries. Next, we tokenized the log entries using the NLTK tokenizer [15], splitting them into individual tokens. The following transformations are then applied to each token:

1. Special characters are removed, except for numeric, alphabetic, and forward slash (`/`) characters.
2. Tokens potentially containing server or device names are replaced with placeholders, including those starting with `xfel` or ending in `svr` or `server`.
3. Numeric tokens are replaced with placeholders like `$nz` for non-zero numbers and `$zero` for zeros.
4. Entire log entry is converted to lowercase.
5. English stop words [15] are removed.

Preprocessing significantly reduced sparseness in the log entries by converting them into consistent tokenized forms. The key steps of entry extraction, tokenization, entity masking, and stop word removal help prevent overfitting minor textual variations. This enables more robust sequence modeling in later stages.

### *Embedding*

In our approach, we use `Word2Vec` [11, 12] to represent log entries numerically in a $N$-dimensional space. `Word2Vec` is based on the idea that words appearing in similar contexts likely have similar meanings. It trains a shallow neural network to reconstruct word contexts, learning embeddings that capture semantic relationships from the surrounding words. We employ continuous bag-of-words (CBOW) introduced in [12] to train the `Word2Vec`. CBOW uses the context to predict a target word omitted in the input, alternatively, a skip-gram training can be used, which does the reverse. The linear `Word2Vec` mapping learns vectors where similarity in embedding space correlates to semantic similarity. For log analysis, `Word2Vec` can learn relationships between terms that often co-occur, capturing the context. An important capability is that arithmetic operations can be performed on the embedded vectors. For example, adding the embeddings for `disk` and `space` yields vectors close to related terms like `available` and `lack`. Furthermore, combining `linux` and `mac` embeddings produces vectors near other operating system terms like `windows` and `os`, see Fig. 2. The additive property is important for representing multi-word log entries by taking the mean of the token embeddings. While more complex pooling techniques exist [16, 17], mean pooling proved sufficient for our needs.

### *Anomaly Detection with HMM*

We borrow the notation from [19]. Consider a set $\{q_1, \ldots q_N\}$ of hidden states, and a sequence of observations $(o_1, \ldots o_T)$, each one drawn from a vocabulary $V$. We make two assumptions: first, that a state $q_i$ depends only on the previous state $q_{i-1}$, i.e. $p\left(q_i|q_1, \ldots q_{i-1}\right) = p\left(q_i|q_{i-1}\right)$ (first-order Markov assumption), second a probability of $o_i$ depends only on state that produced the observation $q_i$ and not on any other states or observations, i.e. $p\left(o_i|q_1, \ldots q_i \ldots q_T, o_1, \ldots, o_i, \ldots, o_T\right) = p\left(o_i|q_i\right)$. The above-stated assumptions can be represented via hidden Markov Models (HMM).

In our model, the observations are vector representations of log entries, obtained through preprocessing, tokenization, and embedding into an $N$-dimensional space. The hidden states represent the unknown underlying state of the system generating the logs.

Given a sequence of observed log vectors $(o_1, \ldots, o_{i-1})$, our goal is to estimate the probability of a new vector $o_i$ and compare how probable his occurrence is considering previously observed vectors $(o_1, \ldots, o_{i-1})$

$$\begin{aligned} s_i &= \log \frac{p_\theta(o_1, \ldots, o_{i-1})}{p_\theta(o_1, \ldots, o_i)} \\ &= \log p_\theta\left(o_1, \ldots, o_{i-1}\right) - \log p_\theta\left(o_1, \ldots, o_i\right). \end{aligned} \tag{1}$$

The score $s_i$ is quantifying the anomaly level $s_i$ of the new entry $o_i$ based on parameters $\theta$. HMM parameters $\theta$ can be estimated from inputs before $o_i$, i.e.$(o_1, \ldots, o_{i-1})$ or a sub-sequence (e.g. sliding window). We discuss estimation in the following section.

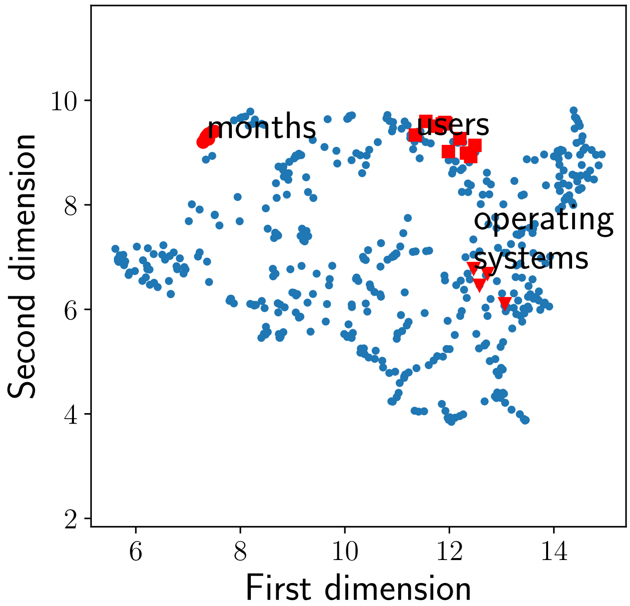


Figure 2: Figure shows a 2D UMAP [18] projection of the 16-dimensional `Word2Vec` embeddings for all words in the entire vocabulary. In the projection, semantically similar words within distinguished categories have embedding vectors clustered closely together, distinguished by the red. They indicate clusters of three-letter month names (except `feb` which is absent in vocabulary), user names, and operating systems (i.e. `windows`, `linux`, `mac`, `os`). The proximity of point clusters with similar meanings shows that the `Word2Vec` has learned latent semantic relationships and mapped similar words to nearby embedding locations.

A key requirement in log anomaly detection is handling novel entries [5]. Although our method cannot fully generalize to completely new logs, it focuses more on sequence modeling than individual semantics. This enables detecting anomalies based on contextual irregularities and variations rather than content. We observed that anomalies manifest more as unusual sequences than specific terms. By scoring based on sequence likelihood rather than keyword rules, even new log messages can be assigned anomaly scores using their contextual deviation. This differentiates our technique from [3–5, 7, 10], which relies on supervised classification. Instead, we take an unsupervised sequential approach to assess entries based on context rather than predefined labels. However, false alarms may still occur if natural fluctuations also deviate from learned patterns, as we show in Fig. 8.

**Parameter Estimation** It is important to clarify the specific log sequence segments used for estimating the HMM parameters $\theta$ in our experiments.

In the real-world results presented (Figures 5, 6, 7, and 8), parameter estimation consistently leveraged all previous log messages, keeping test sequences separate. The final few messages were excluded and treated as a test set.

This strategy demonstrated satisfactory efficiency given the hardware, as the Baum-Welch algorithm for HMM training scales linearly with sequence length.

However, some stations had tens of thousands of messages, potentially causing unreasonable computational time growth. We explored two non-overlapping strategies to mitigate this:

1. Using a sliding window subset of the sequence, as shown in Fig. 1. Since the HMM has few parameters, it remains stable even on shorter training sequences.

2. Initializing with parameters from the previous iteration, rather than full retraining. This avoids deviating far from a previous parameter estimate.

The sliding window approach dynamically focuses on recent local context, while parameter reuse leverages past parameters as context and can increase stability. Both maintain reasonable training times but have trade-offs we intend to evaluate further in future work. Full sequence training sufficed initially, but scaling necessitates more efficient training strategies. The sliding window and reuse techniques present two promising directions for larger-scale logs.

Unlike the minimal example in Fig. 1 where the anomaly was excluded from training, Fig. 3 shows the performance when the full log sequence including the anomaly is used for HMM estimation. Despite the anomalous entries being present during estimation, the model still detects the disruption in the learned sequence patterns, as evidenced by the spike in anomaly scores. This highlights an important capability of our approach - the ability to identify anomalies even when trained on logs containing anomalies. By relying on sequential deviations rather than content matching, the presence of irregular entries in the training logs does not prevent detecting more such deviations at test time. This enables post-mortem analysis scenarios where clean training data is not available. Even if the training logs contain some anomalies, new anomalies can still be flagged based on their contextual irregularity. The model detects breaks in sequential patterns irrespective of whether anomalous logs were seen during training.

### *An Example - Sequence Anomaly Detection*

To demonstrate how HMM can detect different types of anomalies, consider a simple HMM with two hidden states $q_1, q_2$. We examine two scenarios with different observable outputs:

- The observable vocabulary contains two common outputs $o_1, o_2$ along with one anomalous event $o_a$

- The observable vocabulary only contains standard outputs $o_1, o_2$. However, one of these common outputs appears in an unexpected sequence position.

In the first scenario, the anomalous output $o_a$ will decrease the likelihood and thus increase the anomaly score when it appears, allowing it to be flagged as anomalous. In the second scenario, although the observed output is familiar, its occurrence in an unlikely position based on the learned sequence dynamics will also increase the anomaly score.

To demonstrate, we created a minimal 8-event example where disrupting the pattern impacts the scores, see Fig. 1. Critically, the HMM parameters are estimated excluding the last (anomalous) event, and this approach succeeds even if the anomalous sequence was included in parameter estimation, as Fig. 3 shows. This example shows that the model detects anomalies by identifying disruptions in expected patterns, even with limited and corrupted input data.

We show that HMM can detect anomalies either due to unlikely/novel log messages themselves or due to standard messages appearing in surprising positions that break the expected sequencing patterns. The HMM assigns lower scores when observations diverge from the learned distributions. The examples in Fig. 1 and Fig. 3 also demonstrate that finding parameters of HMM requires only a few events and inputs can even be corrupted with noise.

### *Analysis - Word Embedding*

In this section, we will perform a more detailed analysis of the embeddings produced by our corpus to demonstrate robustness even though the log entries are not natural language. Processing logs with `Word2Vec` embedding presents an interesting language task because the corpus contains only a few words (475 unique tokens) and after pre-processing and tokenization, there are less than 1000 unique log messages, see Fig. 4. The absence of diversity of messages and words also justifies why more parameter-rich approaches are unfeasible.

Fig. 2 shows an embedding with some points being distinguished to demonstrate that semantically similar words are embedded closely. Furthermore, in Fig. 4, we show embedding of averaged entry vectors to further underline the challenge in the lack of diversity of log entries, which form only a few packed clusters.

## IMPLEMENTATION

The code was implemented in Python 3.9. For embedding log messages with `Word2Vec`, the Gensim library was used [20]. For modeling the sequences, the Hmmlearn package was used [21].

## RESULTS

We selected four instances to show. Input logs and their computed scores are shown in Figures 5, 6, 7 and 8.

Fig. 5 shows a short part of the input log and results of the anomaly detection method working as anticipated - the sudden appearance of the error message `rpccheck nullproc error` at row 11 coincides with a sharp increase in anomaly scores, as it is indicated in the bottom left and right charts. This clear spike shows the method can detect anomalous events.

In Fig. 6, we again see the method detecting an anomaly as expected - the error message at row 5 leads to an abrupt

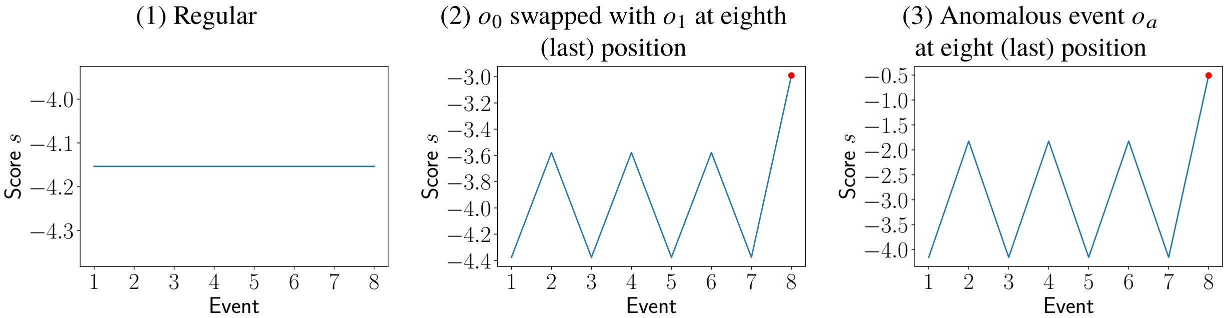


Figure 3: To demonstrate the anomaly detection capabilities of our HMM model and its capacity to detect anomalies of log entries which are also a part of parameter estimation, but are not so frequent we show results of anomaly detection of identical sequences like it Fig. 1, but we use the entire sequence for training. The results show that the model detects anomalies from both unlikely or novel log messages, even when the anomalous data are a part of the parameter estimation of the HMM.

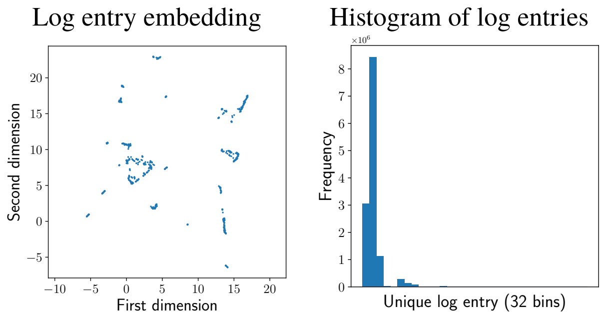


Figure 4: Distribution of unique log entries (sum token embeddings) embedded in 16 dimensions and then projected to 2 dimensions with UMAP [18]. The absence of uniform distribution of the embedded log messages shows that there are only very few clusters of messages.

escalation in scores in the bottom left chart. This confirms the method's ability to identify potential issues.

Fig. 7 showcases more nuance and complexity. Score spikes at rows 9, 11, and 14 reveal multiple potential anomalies according to the method. However, the spike at 14 stands out as most prominent when viewed in the context of the previous 50 entries in the bottom right chart. This example illustrates that while the method can flag multiple possibilities, further verification may be needed to determine the most significant anomaly.

Finally, Fig. 8 highlights some limitations and challenges. The high baseline of scores makes it harder to discern anomalies from typical background noise for this log. Also, frequent errors, even if minor, generate many potential false positives. Despite these difficulties, a slight increase in the score at row 17 still suggests the method can detect likely anomalies if we take a closer look at the results.

When errors produce clear spikes as in the first two examples in Fig. 5 and Fig. 6, the method reliably flags issues. With more nuanced cases as in Fig. 7, multiple possibilities may need further validation. For completeness, we also pointed out an example where the proposed approach does not work as reliably, shown in Fig. 8, but even with noisy baseline data, salient anomalies can emerge.

## FUTURE WORK

While this work demonstrates preliminary anomaly detection capabilities, there are several possibilities for improvement in the future. More advanced techniques like [22] could provide greater accuracy in identifying anomalies. Furthermore, increasing the log verbosity may lead to generating more data that could allow us to deploy more parameter-rich anomaly detection algorithms, as mentioned in related work.

Incorporating additional node data beyond just log messages holds promise for improving detection performance. Characteristics like CPU, memory, network, and disk usage contain valuable information but effectively combining such asynchronous numerical time series data with log messages poses modeling challenges. Developing algorithms to jointly analyze these diverse data sources represents the next milestone.

Furthermore, cybersecurity factors merit consideration given rising threats. Our knowledge of infrastructure specifics alongside network traffic flow logs could enable modeling and identifying security-related anomalies.

## CONCLUSION

This work presents a novel unsupervised approach for detecting anomalies in log data. By representing log entries with `Word2Vec` embeddings and modeling sequences as HMMs, the method identifies anomalies by calculating the likelihood of new log messages given history.

The results on real logs from European XFEL demonstrate the capability to flag potential issues via salient score spikes corresponding to errors or disruptions of typical patterns. The approach detects anomalies without requiring labeled data or extensive training and relies on modeling the behavior of the node log via HMM.

However, challenges remain in handling noise and minimizing false positives, as evidenced by certain noisy logs.

```
                                  ⋮
0    remoteerrors errorcount $nz
1    remoteerrors errorcount $nz
2    remoteerrors errorcount $nz
3    remoteerrors errorcount $nz
4    remoteerrors errorcount $nz toggled $nz times $nz min
5    remoteerrors errorcount $nz
6    remoteerrors errorcount $nz
7    remoteerrors errorcount $nz
8    remoteerrors errorcount $nz
9    remoteerrors errorcount $nz toggled $nz times $nz min
10   remoteerrors errorcount $nz
11   rpccheck nullproc error
12   rpccheck fails $nz kill $nz
13   getpid pid not match process name
14   no process try start
15   getpid pid not match process name
16   pid change $nz $nz
17   rpccheck nullproc error
18   rpccheck fails $nz kill $nz
19   rpccheck fails $nz kill $nz
                                  ⋮
```

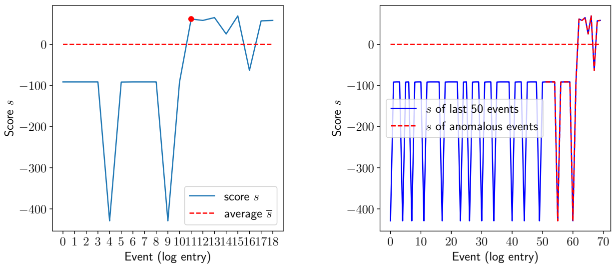


Figure 5: An anonymized instance of an anomalous log (upper section), with anomalous events becoming evident starting from row 11. This is marked by the sudden appearance of the error message `rpccheck nullproc error`, coinciding with a notable increase in anomaly scores (depicted in the bottom left and right figures). The more detailed score plot (bottom left) provides a close-up view of the scores, revealing a rapid increase beginning at the 11th log entry, indicative of a significant error. The score plot in the bottom right showcases scores for the last fifty log entries (in blue), with overlaid scores from just before the commencement of the anomalous event (represented by red dashed lines).

```
                                  ⋮
0    remoteerrors errorcount $nz toggled $nz times $nz min
1    remoteerrors errorcount $nz
2    remoteerrors errorcount $nz
3    remoteerrors errorcount $nz
4    remoteerrors errorcount $nz toggled $nz times $nz min
5    rpccheck clnt create error
6    remoteerrors errorcount $nz
7    rpccheck fails $nz kill $nz
8    pid change $nz $nz
9    rpccheck fails $nz kill $nz
10   pid change $nz $nz
11   rpccheck fails $nz kill $nz
12   no process try start
13   pid change $nz $nz
14   remoteerrors errorcount $nz
15   no process try start toggled $nz times $nz min
16   remoteerrors errorcount $nz
17   remoteerrors errorcount $nz toggled $nz times $nz min
18   remoteerrors errorcount $nz
19   remoteerrors errorcount $nz
20   remoteerrors errorcount $nz toggled $nz times $nz min
                                  ⋮
```

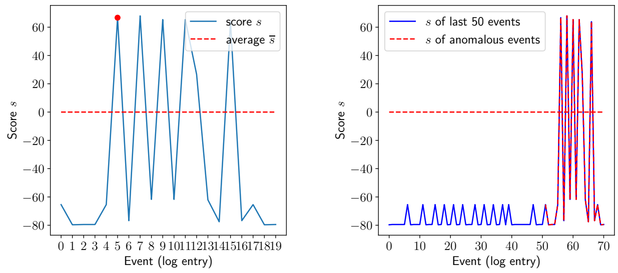


Figure 6: An anonymized example of an anomalous log (top section), with the emergence of anomalous events starting at row 5. The anomalous series of events starts with the occurrence of the message `rpccheck fails` *nzkill*`nz`. Consequently, an abrupt increase in anomaly scores is observed (visible in the bottom left). Notice the score plot in greater detail (bottom left) presents a sharp escalation in scores following the 5th log entry. Furthermore, despite log entry 6 bearing resemblance to entries 0-4, the error present in log entry 6 induces a slight increase in the score associated with log entry 5. The score chart (bottom right) provides an overview of scores for the preceding 50 log entries, prominently highlighting the location of the likely occurrence of the problem.

Code is available at https://github.com/sulcantonin/LOG_ICALEPCS23

## ACKNOWLEDGEMENTS


We acknowledge DESY (Hamburg, Germany), a member of the Helmholtz Association HGF, for its support in providing resources and infrastructure. Furthermore, we would like to thank all colleagues of the MCS and MSK groups as well as the European XFEL team and management for their contributions to this work and help in preparing this paper.


## REFERENCES


[1] M. Cinque, D. Cotroneo, and A. Pecchia, "Event logs for the analysis of software failures: A rule-based approach," *IEEE Transactions on Software Engineering*, vol. 39, no. 6, pp. 806–821, 2012.

[2] T.-F. Yen *et al.*, "Beehive: Large-scale log analysis for detecting suspicious activity in enterprise networks," in *Proceedings of the 29th annual computer security applications conference*, 2013, pp. 199–208.

[3] 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*, 2017, pp. 1285–1298.

[4] K. Zhang, J. Xu, M. R. Min, G. Jiang, K. Pelechrinis, and H. Zhang, "Automated it system failure prediction: A deep learning approach," in *2016 IEEE International Conference on Big Data (Big Data)*, IEEE, 2016, pp. 1291–1300.

[5] X. Zhang *et al.*, "Robust log-based anomaly detection on unstable log data," in *Proceedings of the 2019 27th ACM Joint

```
⋮
0   getpid no process
1   no process try start
2   getpid no process
3   getpid no process
4   no process try start
5   getpid no process
6   no process try start
7   no process try start
8   pid change $nz $nz
9   getpid pid not match process name
10  pid change $nz $nz
11  getpid pid not match process name
12  pid change $nz $nz
13  pid change $nz $nz
14  pid not match process name toggled $nz times $nz min
15  pid not match process name toggled $nz times $nz min
16  signal term received
17  terminating threads closing files
18  writer thread terminated
19  interrupt thread terminated
⋮
```

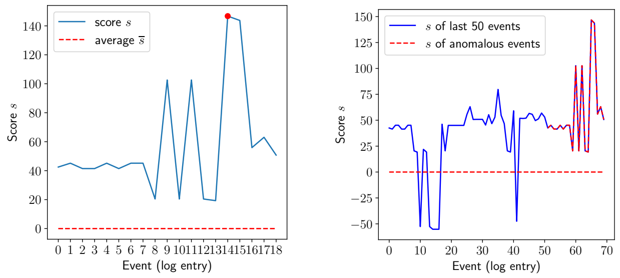


Figure 7: In this anonymized log, an anomaly appears to occur around row 14. At this point, the error message `pid not match process name toggled` appears repeatedly. This corresponds to a spike in the anomaly scores, as seen in the bottom left chart. However, similar spikes occur earlier at log entries 9 and 11, suggesting other potential anomalies. The detailed anomaly score chart shows a sharp rise after log entry 13. But notably, entries 9 and 11 also cause score increases, not just entry 14. When viewed in the context of the previous 50 log entries, the score spike at 14 is prominent, as illustrated in the bottom right chart. However, the earlier spikes indicate this may not be an isolated anomaly.

```
⋮
0   config file create error
1   config file create error
2   config file create error
3   remoteerrors errorcount $nz
4   config file create error
5   config file create error
6   config file create error
7   config file create error
8   rpccheck nullproc error
9   no process try start
10  pid change $nz $nz
11  getpid no process
12  no process try start
13  pid change $nz $nz
14  getpid no process
15  no process try start
16  pid change $nz $nz
17  rpccheck nullproc error toggled $nz times $nz min
18  rpccheck nullproc error
19  rpccheck fails $nz kill $nz
⋮
```

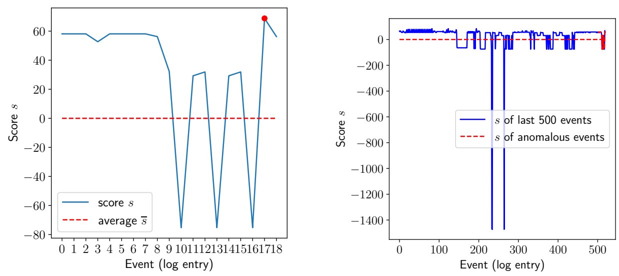


Figure 8: In this anonymized log, an anomaly appears at row 17, where the error message `rpccheck nullproc error toggled` first appears. This corresponds to a spike in the anomaly scores, as shown in the bottom left chart. Interestingly, earlier error messages like `config file create error` have relatively high anomaly scores as well. However, since these repeat multiple times, their scores are slightly lower than the message on row 17. The message `pid change $nz $nz` also appears repeatedly in different contexts. Typically, this message does not indicate anything unexpected, which is why its anomaly scores tend to be much lower on average. The spike at row 17 stands out as the most prominent anomaly in this log example. The bottom right chart shows the anomaly scores for the last 500 log messages from this node. It illustrates that this node tends to produce various (not necessarily) error messages fairly often, leading to generally high anomaly scores and many false positives with our method.


*Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2019, pp. 807–817.

[6] A. Vaswani *et al.*, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.

[7] H. Guo, S. Yuan, and X. Wu, "Logbert: Log anomaly detection via bert," in *2021 international joint conference on neural networks (IJCNN)*, IEEE, 2021, pp. 1–8.

[8] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805*, 2018.

[9] M. Landauer, S. Onder, F. Skopik, and M. Wurzenberger, "Deep learning for anomaly detection in log data: A survey," *Machine Learning with Applications*, vol. 12, p. 100 470, 2023.

[10] C. Bertero, M. Roy, C. Sauvanaud, and G. Trédan, "Experience report: Log mining using natural language processing and application to anomaly detection," in *2017 IEEE 28th International Symposium on Software Reliability Engineering (ISSRE)*, IEEE, 2017, pp. 351–360.

[11] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," *Advances in neural information processing systems*, vol. 26, 2013.

[12] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," *arXiv preprint arXiv:1301.3781*, 2013.

[13] J.-G. Lou, Q. Fu, S. Yang, Y. Xu, and J. Li, "Mining invariants

from console logs for system problem detection.," in *USENIX annual technical conference*, 2010, pp. 1–14.

[14] W. Xu, L. Huang, A. Fox, D. Patterson, and M. Jordan, "Online system problem detection by mining patterns of console logs," in *2009 ninth IEEE international conference on data mining*, IEEE, 2009, pp. 588–597.

[15] S. Bird, E. Klein, and E. Loper, *Natural language processing with Python: analyzing text with the natural language toolkit*. " O'Reilly Media, Inc.", 2009.

[16] T. Gao, X. Yao, and D. Chen, "Simcse: Simple contrastive learning of sentence embeddings," *arXiv preprint arXiv:2104.08821*, 2021.

[17] N. Reimers and I. Gurevych, "Sentence-bert: Sentence embeddings using siamese bert-networks," *arXiv preprint arXiv:1908.10084*, 2019.

[18] L. McInnes, J. Healy, and J. Melville, "Umap: Uniform manifold approximation and projection for dimension reduction," *arXiv preprint arXiv:1802.03426*, 2018.

[19] J. Daniel, M. James H, *et al.*, *Speech and language processing: An introduction to natural language processing, computational linguistics, and speech recognition*. prentice hall, 2007.

[20] R. Řehůřek and P. Sojka, "Software Framework for Topic Modelling with Large Corpora," English, in *Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*, `http://is.muni.cz/publication/884893/en`, Valletta, Malta: ELRA, May 2010, pp. 45–50.

[21] *Hmmlearn python package*, `https://maxbachmann.github.io/RapidFuzz/`, Accessed: 2023-08-15.

[22] N. Görnitz, M. Braun, and M. Kloft, "Hidden markov anomaly detection," in *International conference on machine learning*, PMLR, 2015, pp. 1833–1842.