Detecting Outliers in CI/CD Pipeline Logs using Latent Dirichlet Allocation

Daniel Atzberger[1] [a], Tim Cech[2] [b], Willy Scheibel[1] [c], Rico Richter[2] [d], and Jürgen Döllner[1] [e]
[1] *University of Potsdam, Digital Engineering Faculty, Hasso Plattner Institute, Germany*
[2] *University of Potsdam, Digital Engineering Faculty, Germany*
{*daniel.atzberger, willy.scheibel*}*@hpi.uni-potsdam.de*
{*tcech, rico.richter.1, doellner*}*@uni-potsdam.de*

Keywords: Log Analysis, Anomaly Detection, Event-Streaming, Latent Dirichlet Allocation

Abstract: Continuous Integration and Continuous Delivery are best practices used in the context of DevOps. By using automated pipelines for building and testing small software changes, possible risks are intended to be detected early. Those pipelines continuously generate log events that are collected in semi-structured log files. In practice, these log files can amass 100 000 events and more. However, the relevant sections in these log files must be manually tagged by the user. This paper presents an online learning approach for detecting relevant log events using Latent Dirichlet Allocation. After grouping a fixed number of log events in a document, our approach prunes the vocabulary to eliminate words without semantic meaning. A sequence of documents is then described as a discrete sequence by applying Latent Dirichlet Allocation, which allows the detection of outliers within the sequence. By integrating the latent variables of the model, our approach provides an explanation of its prediction. Our experiments show that our approach is sensitive to the choice of its hyperparameters in terms of the number and choice of detected anomalies.

1 INTRODUCTION

Continuous Integration (CI) is a set of best practices in agile software development, in which developers frequently push changes to the source code to the project's main branch, which is then built and tested by an automated processing pipeline (Fitzgerald and Stol, 2017). Continuous Delivery (CD) is built on CI, i.e., "Continuous Delivery is a software development discipline where you build software in such a way that the software can be released to production at any time"[1], and therefore allows a user or customer to interact with the software early. As a result, CD improves the overall "execution efficiency, cross-team communication, product-market fit, agility, and organizational transparency"[2] through iterative feedback loops between the users and the engineering team. In Continuous Deployment (CDE), an extension of CD, the builds that successfully pass the tests are automatically deployed in an actual production environment (Shahin et al., 2017).

CI/CD pipelines rely on various coding, building, testing, deployment, and monitoring tools. Each step in the CI/CD pipeline provides a status message, a so-called log event, to inform the developers about the results of the step, e.g., whether a test was successful or failed. The individual log events are collected in a single log file in a semi-structured form during the execution of the pipeline. In the case of complex pipelines, which can take several days to execute, the log files can have a length of several 100 000 lines. In order to get relevant feedback faster, it would be desirable to stop the pipeline execution early after the first anomaly and detect the relevant sections in the log file that correlate with the problematic behavior. Rule-based methods that search for keywords are not applicable in our use case because the vocabulary depends heavily on the team and the tools they use. Modern approaches relying on Machine Learning achieve good results but provide no explanation to the user, thus complicating their use in practice (Zhao et al., 2021).

We present an unsupervised approach that detects

[a] https://orcid.org/0000-0002-5409-7843
[b] https://orcid.org/0000-0001-8688-2419
[c] https://orcid.org/0000-0002-7885-9857
[d] https://orcid.org/0000-0001-5523-3694
[e] https://orcid.org/0000-0002-8981-8583
[1] martinfowler.com/bliki/ContinuousDelivery.html
[2] atlassian.com/continuous-delivery/principles

anomalies in individual log events. For this purpose, our algorithm groups a fixed number of log events, i.e., lines in the log file, into single documents and cleans them from words with no semantic meaning, e.g., stopwords. The corpus generated in this way is used for training a Latent Dirichlet Allocation (LDA) model, which results in a description of the log files as a discrete sequence. By detecting anomalies in the sequence, our algorithm detects suspicious entries. It furthermore explains its result by taking the topic word distributions into account. We showed that our approach can detect relevant sections within large log files through interviews with domain experts. Furthermore, we investigated the influence of the hyperparameters on the results.

Our work has the following structure: In Section 2, we present existing approaches for detecting anomalies in log files. In Section 3, we present details about LDA required for our approach, which is presented in Section 4. Our conducted experiments, their results, and threats to validity are presented in Section 5. We conclude this work in Section 6.

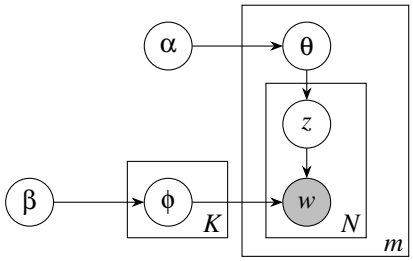
2 RELATED WORK

Numerous approaches for detecting log anomalies, i.e., relevant parts in semi-structured log files, have been proposed (Wittkopp et al., 2022). Supervised and unsupervised techniques have been developed, both usually following the same four steps (Le and Zhang, 2022): (1) log parsing, (2) log grouping, (3) the formalization of log events by semantic vectors, and (4) the application of a Deep Learning (DL) model for detecting outliers in sequences of vectors or discrete values. In our considerations, we present modern supervised and unsupervised algorithms for detecting log anomalies and ideas for explaining the results. For a detailed survey on automated log analysis, we refer to (He et al., 2021).

Supervised Approaches. Zhang et al. proposed a log anomaly detection algorithm that has shown promising results in dealing with unstable log data, i.e., changing logging statements and processing noise (Zhang et al., 2019). Their approach *LogRobust* derives a vector representation for a log event using pre-trained word embeddings and a subsequent tf-idf aggregation scheme. Subsequently, log anomalies are detected using an attention-based Bi-LSTM network. Yang et al. applied a similar approach for creating a semantic vector embedding for log files (Yang et al., 2021). Their approach *PLELog* further incorporates knowledge about historical log files that are marked as normal to estimate the class label of a set of historical log files. The labeled data is then used to train an attention-based GRU neural network for detecting outliers. The approach presented by Lu et al. relies on sequences of so-called log keys, i.e., frequently occurring constants in log files recognized by a log parser, and specific vector embeddings trained for such log keys (Lu et al., 2018). The authors developed two supervised DL methods, one relying on a Convolutional Neural Network (CNN) and the other on a Multi-layer Perceptron (MLP). The approach *LogBERT* presented by Guo et al., also starts with a description of logs derived from log keys (Guo et al., 2021). Using Bidirectional Encoder Representation from Transformers (BERT), their approach aims to learn patterns of normal log sequences and detects suspicious log sequences that deviate from normal ones. Shao et al. further developed *LogBERT* by utilizing a DL model. A comparison of different supervised DL approaches for log anomaly detection was presented by Le et al. (Le and Zhang, 2022).

Unsupervised Approaches. Meng et al. proposed *LogAnomaly*, an unsupervised approach for modeling a stream of log data, where each log event is described using a vector space embedding derived from *Word2Vec* (Meng et al., 2019). Their approach consists of a DL component trained offline and an online outlier detection algorithm. A similar concept was proposed by Chen et al., whose approach also consists of an offline training mode and an online outlier detection algorithm (Chen et al., 2020). Furthermore, their approach combines labels from a source system and a target system using a transfer learning algorithm to reduce the noise in log sequences. In current work, Li et al. showed that the results of unsupervised DL approaches can further be improved by employing Part-of-Speech and Named Entity Recognition into account (Li et al., 2022).

Explainability. In all the approaches presented so far, the DL component was used as a black-box, i.e., the user cannot explain why a log snipped is marked as an outlier. However, explaining anomalies is highly desirable in practice (Zhao et al., 2021). Brown et al. presented an unsupervised learning method for anomaly detection that achieves the quality of a deep learning model and further explains the decision-making basis (Brown et al., 2018). Specifically, their approach uses an Recurrent Neural Network (RNN) based on the Long Short Term Memory (LSTM) units to build a statistical model that learns vocabulary distribution within logs. At the same time, an attention score is computed to indicate the presence of an out-



LDA

Figure 1: Graphical model underlying LDA in plate notation. The shaded circles denote observed variables.

lier. Our method takes up the idea of explainability and allows the user to conclude on the level of words within a log. To the best of our knowledge, we are the first to present a log anomaly detection algorithm that provides explainability in this detail.

3 LATENT DIRICHLET ALLOCATION

LDA is a probabilistic topic model proposed by Blei et al., which is often used for capturing the semantic structure within a collection of documents, a so-called corpus (Blei et al., 2003). Given a corpus $\mathcal{C}$, i.e., a set of documents $\{d_1,\ldots,d_m\}$ that share a vocabulary $\mathcal{V} = \{w_1,\ldots,w_n\}$, LDA extracts semantic clusters in its vocabulary, so-called *topics* $\phi_1,\ldots\phi_K$, as multinomial distributions over the vocabulary $\mathcal{V}$. Additionally, each document is described as a mixture of the extracted topics, describing its semantic composition. Given the number of topics K specified as a hyperparameter together with Dirichlet priors $\alpha = (\alpha_1,\ldots,\alpha_K)$, where $\alpha_i > 0$ for all $1 \leq i \leq K$ and $\beta = (\beta_1,\ldots,\beta_N)$, where $\beta_j > 0$ for all $1 \leq j \leq N$ that capture the document-topic-distributions and the topic-term-distribution respectively, LDA defines a fully generative model for the creation of a corpus. The plate notation of the generative process is shown in Figure 1. The generative process operates as follows:

1. For each document d_i, $1 \leq i \leq m$, choose a distribution over topics $\theta_i \sim \text{Dirichlet}(\alpha)$
2. For each word w_j, $1 \leq j \leq N$, in d_i, $1 \leq i \leq m$,
   (a) Choose a topic $z_j \sim \text{Multinomial}(\theta_i)$
   (b) Choose the word w_j according to the probability $p(w_j|z_j,\beta)$

During the training phase LDA infers the topics $\phi_1,\ldots,\phi_K$ and the document-topic-distributions $\theta_1,\ldots,\theta_m$ for the given corpus. As exact inference is intractable, approximation techniques must be applied, e.g., Variational Bayes (Blei et al., 2003), its online version (Hoffman et al., 2010), or (collapsed) Gibbs Sampling (Griffiths and Steyvers, 2004). A survey of different approaches for utilizing LDA for mining software repositories can be found in (Chen et al., 2016).

4 APPROACH

Our approach aims to detect relevant sections within a log file by utilizing LDA for describing a log file as a discrete sequence. Log anomalies are detected using an outlier detection algorithm by associating relevant log sections with conspicuous subsequences. Furthermore, our approach supports the user in explaining the results by viewing the latent variables of the LDA model. Figure 2 shows the concept of our approach.

4.1 Corpus Creation

Log events are preprocessed immediately after they are created to remove words that do not carry semantic meaning. For this purpose, the following steps are performed:

1. Removal of general log-data specific phrases, e.g., `ERROR_ON_WARNING`,
2. Removal of symbols, e.g., parentheses, time stamps, file paths, and number literals,
3. Lower-casing all words,
4. Removal of stop words, and
5. Lemmatization of the vocabulary to reduce the size of the vocabulary.

A fixed number of preprocessed log events are merged into a document and stored in the Bag of Words (BOW) form. The documents generated in this way form the corpus for subsequent analysis by LDA. During the execution of the pipeline, the corpus is permanently extended by new documents.

4.2 Sequence Modelling

Since not the entire corpus is available at the beginning, we apply the online version of Variational Bayes for training the LDA model. As a result, each document is described by a K-dimensional vector whose components represent the expression in the respective topic. By choosing a symmetric Dirichlet prior α with $\alpha_1,\ldots,\alpha_K = 1/K$, the inference algorithm favors topic distributions with strong expressions in a few topics. We, therefore, identify each document with

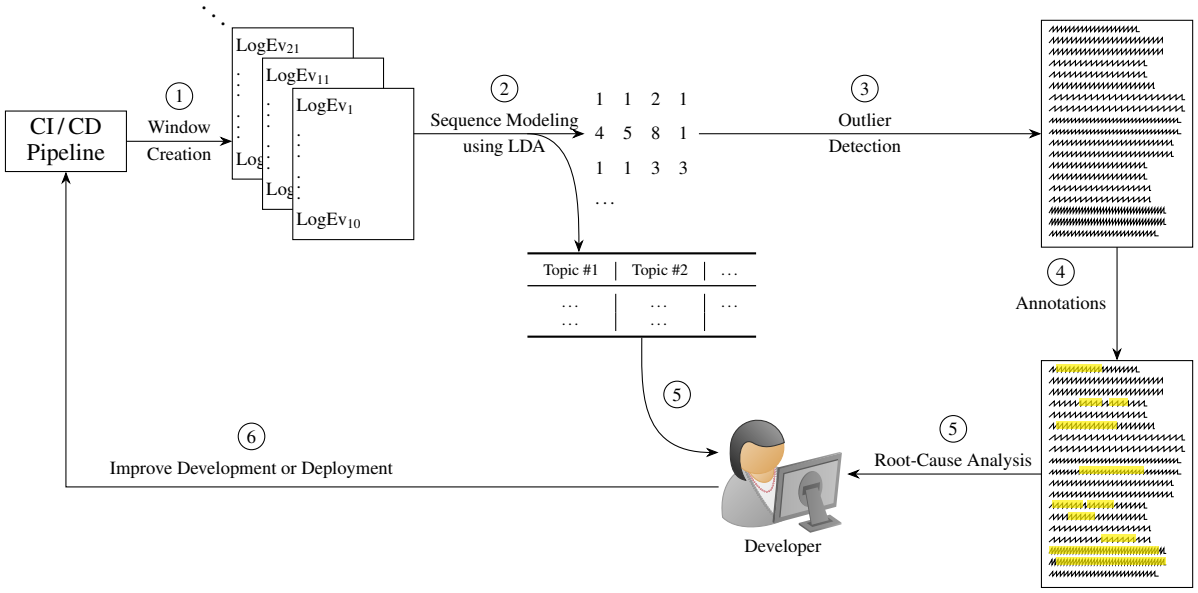


Figure 2: Process Overview of our approach. The log events from a CI/CD pipeline are pruned and grouped into documents. A discrete sequence is generated using LDA, and outliers are detected. The resulting topics and the topic-term assignments in the conspicuous document are presented to the developer.

the number $1,\ldots,K$ of its dominant topic as proposed by Panichela et al. (Panichella et al., 2013). This way, we obtain a discrete sequence with values in $1,\ldots,K$, created during the pipeline execution.

Our implementation is based on the *gensim* library[3], which is actively maintained and widely used in the industry. We further apply the *NLTK*[4] for preprocessing the data.

4.3 Outlier Detection

A discrete sequence $\mathcal{S}$ is an ordered list $(s_1,\ldots,s_N)$ of elements from a finite set. Different definitions for anomalies in the context of discrete sequences exist (Chandola et al., 2012):

1. An entire sequence $\mathcal{S} = (s_1,\ldots,s_N)$ is anomalous if it is significantly different from a set of other sequences.
2. A subsequence $\mathcal{S}_i = (s_i,\ldots,s_{i+k-1})$ within the sequence $\mathcal{S}$ is anomalous if it is significantly different from the other subsequences within $\mathcal{S}$ of length k. Here the window size k is a hyperparameter of the model and needs to be set by the user.
3. An element s_i, $1 \leq i \leq N$, within $\mathcal{S}$ is anomalous if it significantly differs from an element expected at position i.

[3]https://radimrehurek.com/gensim/
[4]https://www.nltk.org/

Our application scenario relates to the second case. The first case is not applicable in our scenario as our approach is intended to detect anomalies as they occur and not after the entire pipeline has finished. The third definition does not consider the sequence structure, which has shown relevant information during our experiments.

Numerous online outlier detection algorithms for subsequences were proposed (Chandola et al., 2012). They all involve the same two steps: (1) the calculation of an anomaly score for the respective subsequence and (2) the comparison of the anomaly score with a specified threshold λ. The subsequence is marked as an outlier if the anomaly score exceeds the threshold. To ensure the explainability of our approach, we decided to use a trivial anomaly score. The inverse frequency of the subsequence in the set of previous subsequences gives our anomaly score. By choosing the threshold $0.5 < \lambda < 1$, a subsequence is marked as an outlier if and only if it occurs for the first time. In our experiments, we always look at windows comprising four documents. To avoid too many outliers at the beginning of a new run, we assume a minimum number of 1,000 documents before first comparing the anomaly score with the threshold.

4.4 Explaining Results

After the detection of an outlier, the corresponding log sections, i.e., the k elements of the subsequence,

Table 1: Three extracted topics from our dataset. For each topic, the ten most probable terms are displayed.

Topic #1	Topic #2	Topic #3
state	data	external
change	directory	storage
stage	local	label
warning	input	cache
event	class	load
equal	search	click
mask	missing	path
road	member	warn
result	interface	error
archive	dependency	result

are displayed to the user to derive the root cause for a conspicuous pipeline. Thereby, the reason behind this needs to be explained to the user by two additional outputs.

On the one hand, the underlying concept can usually be derived from the most probable words of a topic. Table 1 shows the top 10 words of three exemplary topics extracted from the dataset used in our experiments. The first topic concerns the abstract topic *Status Reporting*, the second *Data Inclusion*, and the third one *Removed External Storage*. A topic's interpretability can be further increased using a weighting function (Sievert and Shirley, 2014).

The generative process underlying LDA makes it clear that each word is derived from precisely one topic. The latent variable z describes this assignment. Our method marks all words derived from the dominant topic in the given section to assist users further.

5 EVALUATION

Our dataset comprises log events originating from two CI/CD pipelines. The dataset from project one consists of approximately 3.6 million log events, and the dataset from project two consists of approximately 9.9 million. Usually, anomaly detection methods are evaluated on labeled log files. In our case, such data was unavailable; only individual examples could be discussed with domain experts. Therefore, we focus our considerations on two questions that investigate the influence of the two hyperparameters of the method, i.e., the number of topics K and the number s of log events grouped in one document.

5.1 Impact of the Hyperparameters on the Number of Anomalies

Our first experiment investigates the influence of the choice of K and s on the number of anomalies detected. For this purpose, we record the proportion of documents marked as anomalies in the total number of documents for each combination of $s, K \in \{10, 20, 30, 40, 50, 60, 70, 80, 90, 100\}$. We refer to this metric as η in the following. The results for project one are shown in Table 2, and the results for project two are shown in Table 3.

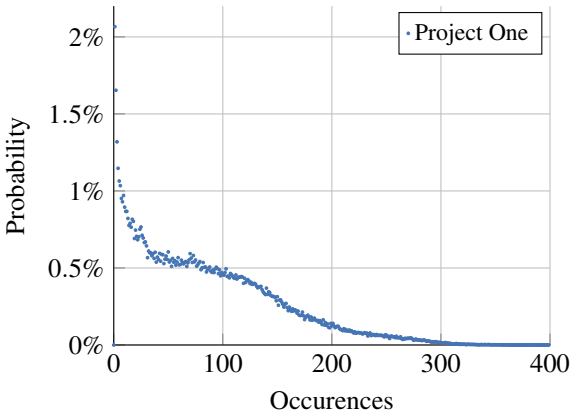


Figure 3: Results of our second experiment on the first project.

Both projects show that when the parameter s is fixed, the metric η tends to grow with increasing K, especially for small values of s. This observation can be explained by the fact that the logs are described as a discrete sequence with values in $\{1, ..., K\}$, and as K grows, there are more possibilities for different subsequences.

With fixed topic number K, the metric η tends to increase with increasing window size s, but not strictly monotonically. For example, η reaches its maximum at $K = 70$ in the first project at $s = 70$. Both projects also show that η seems to reach its maximum in the investigated parameter range. It is striking that η runs through a significantly larger range of values in project one than in project two, which leads to the assumption that η shows a strong dependence on the data set in addition to the hyperparameters.

5.2 Impact of the Hyperparameters on the Detected Anomalies

Our second experiment investigates whether different parameter configurations also lead to different detected anomalies. For this purpose, we count for each log event of both projects the frequency of how often it was marked as an outlier. With 100 possible combinations for s and K and a fixed subsequence length of 4, a log event can be maximally marked as an anomaly 400 times. From the count, we then derive for $1 \leq l \leq 400$ how many log events were marked as outliers exactly l times. The normalized frequencies are shown in Figure 3 and Figure 4.

Table 2: Results for the first project in the first experiment.

s \ K	10	20	30	40	50	60	70	80	90	100
10	1.0	3.2	4.4	5.9	6.2	7.3	8.8	8.6	8.7	9.8
20	1.7	4.2	8.1	8.0	9.8	10.1	12.9	13.8	15.5	13.2
30	1.7	6.1	9.0	11.5	12.4	17.6	15.4	14.5	18.0	16.2
40	2.0	7.6	10.8	14.0	14.6	16.1	15.2	16.6	18.2	18.7
50	3.7	8.3	11.1	13.1	14.3	18.7	20.6	16.9	20.0	19.7
60	2.4	9.4	11.5	16.7	17.2	19.6	19.7	19.7	20.8	19.9
70	3.2	9.9	12.3	18.2	18.7	18.8	25.2	21.4	23.3	23.5
80	3.9	11.6	14.1	18.8	19.4	18.9	19.4	22.3	24.4	21.3
90	4.3	9.2	15.5	17.1	18.1	19.9	21.7	19.4	21.7	21.7
100	4.3	11.2	14.7	18.2	17.8	20.3	20.6	22.5	23.6	23.9

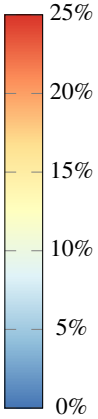
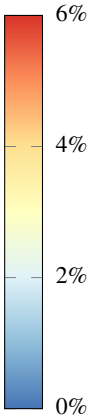


Table 3: Results for the second project in the first experiment.

s \ K	10	20	30	40	50	60	70	80	90	100
10	0.2	0.8	1.6	1.9	2.0	2.3	2.2	2.6	2.2	2.7
20	0.3	1.1	1.8	2.4	2.5	2.9	2.6	3.4	3.0	3.5
30	0.6	1.0	1.8	2.8	2.4	3.2	3.1	3.1	3.0	3.2
40	0.6	1.9	1.9	3.1	3.3	3.4	3.2	4.0	3.4	4.4
50	0.6	1.8	2.6	2.9	3.5	3.6	4.0	3.4	4.4	4.6
60	0.8	1.8	3.0	3.5	3.1	3.8	4.9	3.9	4.5	5.3
70	0.7	1.9	2.5	3.6	3.5	4.0	4.2	4.5	4.3	5.2
80	0.9	2.1	3.0	3.6	4.0	4.1	4.4	4.6	4.9	5.5
90	1.1	2.0	3.4	3.6	4.4	4.4	4.9	5.0	5.3	5.7
100	0.8	2.2	3.0	3.5	4.4	4.9	5.3	5.1	5.6	5.4



In both projects, the observation that lower occurrences are associated with higher probabilities holds. In both projects, the maximum probability occurs at $l = 1$ with 2,1 % in project one and 13,2 % in project two. Overall, different parameter configurations thus lead to enormously different results. However, this fact also depends strongly on the dataset. The maximum value of l is 362 in project one and 353 in project two.

5.3 Threats to Validity

The main threat to internal validity lies in the implementation of our approach. Our LDA model relies on the implementation provided by gensim, and it is still being determined to what extent different inference algorithms would affect the results. Previous experiments for bug triaging tasks have shown that the chosen LDA implementation and its hyperparameters can significantly impact the results (Atzberger et al., 2022). The main threat to generalization is that we reported our observations only from the conducted experiments on two datasets. However, it can already be seen from the two experiments that the results strongly depend on the respective characteristics of the data sets. Furthermore, it can be discussed whether the assumption that anomalies in discrete sequences accompany relevant sections is justified. For example, if only a few words occur within a document, they will have little influence on the dominant topic and, therefore will not be considered in the outlier detection.

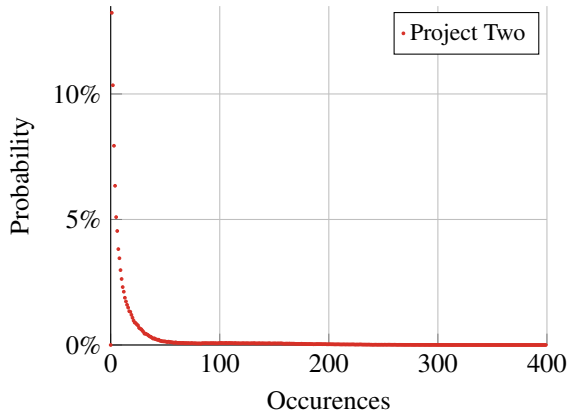


Figure 4: Results of our second experiment on the second project.

6 Conclusions and Future Work

The localization of anomalies in large log files is a question that developers often confront. However, this is enormously time-consuming and hardly possible with high volumes. Furthermore, an entire pipeline run can take several hours to days, so it would be desirable to detect relevant locations early on. Our work presents an approach for the processing and modeling of log events by LDA. Our approach detects anomalies in the resulting discrete sequence and localizes relevant log events. In addition, our method provides explanations through the topics and topic-word mappings, thereby concretely supporting the developer in root-cause analysis. The approach does not require domain knowledge about the individual specifics of the configurations of the pipelines. This is essential because pipeline configurations are volatile and can be changed by developers at any time. If the approach were based on knowledge about the tools being triggered during pipeline execution, the log analytics approach would have to be adapted. The approach detects outlier characteristics in real data logs while the log is consumed as a stream. Hence, the outlier detection takes place early and could serve as early feedback to the developers who have triggered the pipeline execution. Our experiments show that our method is susceptible to the chosen hyperparameters. Besides the frequency of occurring anomalies, the detected log events are also affected.

In future work, we plan to develop modifications to our approach that are more stable under changes in parameters. Furthermore, an evaluation on labeled data and a comparison withDL methods are highly relevant. In addition to the hit rate, a user study will be conducted to investigate the extent to which the results' explainability supports the root cause analysis.

ACKNOWLEDGEMENTS

This work was partially funded by the Federal Ministry of Education and Research, Germany through grant 01IS22062 ("AI research group FFS-AI"), and grant 01IS20088B ("KnowhowAnalyzer").

REFERENCES

Atzberger, D., Schneider, J., Scheibel, W., Limberger, D., Trapp, M., and Döllner, J. (2022). Mining developer expertise from bug tracking systems using the author-topic model. In *Proceedings of the 17th International Conference on Evaluation of Novel Approaches to Software Engineering*, ENASE '22, pages 107–118. INSTICC, SciTePress.

Blei, D., Ng, A., and Jordan, M. (2003). Latent Dirichlet Allocation. *Journal of Machine Learning Research*, 3:993–1022.

Brown, A., Tuor, A., Hutchinson, B., and Nichols, N. (2018). Recurrent neural network attention mechanisms for interpretable system log anomaly detection. In *Proceedings of the 1st Workshop on Machine Learning for Computing Systems*, MLCS '18, pages 1–8. ACM.

Chandola, V., Banerjee, A., and Kumar, V. (2012). Anomaly detection for discrete sequences: A survey. *IEEE Transactions on Knowledge and Data Engineering*, 24(5):823–839.

Chen, R., Zhang, S., Li, D., Zhang, Y., Guo, F., Meng, W., Pei, D., Zhang, Y., Chen, X., and Liu, Y. (2020). Logtransfer: Cross-system log anomaly detection for software systems with transfer learning. In *Proceedings of the 31st International Symposium on Software Reliability Engineering*, ISSRE '20, pages 37–47. IEEE.

Chen, T.-H., Thomas, S. W., and Hassan, A. E. (2016). A survey on the use of topic models when mining software repositories. *Empirical Software Engineering (Springer)*, 21(5):1843–1919.

Fitzgerald, B. and Stol, K.-J. (2017). Continuous software engineering: A roadmap and agenda. *Journal of Systems and Software (Elsevier)*, 123:176–189.

Griffiths, T. L. and Steyvers, M. (2004). Finding scientific topics. *Proceedings of the National Academy of Sciences*, 101(1):5228–5235.

Guo, H., Yuan, S., and Wu, X. (2021). LogBERT: Log Anomaly Detection via BERT. In *In Proceedings of the International Joint Conference on Neural Networks*, IJCNN '21, pages 1–8. IEEE.

He, S., He, P., Chen, Z., Yang, T., Su, Y., and Lyu, M. R. (2021). A survey on automated log analysis for reliability engineering. *ACM Computing Surveys*, 54(6).

Hoffman, M., Bach, F., and Blei, D. (2010). Online learning for latent dirichlet allocation. *Advances in Neural Information Processing Systems*, 23:856–864.

Le, V.-H. and Zhang, H. (2022). Log-based anomaly detection with deep learning: How far are we? In *Proceedings of the 44th International Conference on*

Software Engineering, ICSE '22, pages 1356–1367. IEEE/ACM.

Li, Z., Zhang, J., Zhang, X., Lin, F., Wang, C., and Cai, X. (2022). Natural language processing-based model for log anomaly detection. In *Proceedings of the 2nd International Conference on Software Engineering and Artificial Intelligence*, SEAI '22, pages 129–134. IEEE.

Lu, S., Wei, X., Li, Y., and Wang, L. (2018). Detecting anomaly in big data system logs using convolutional neural network. In *Proceedings of the 16th International Conference on Dependable, Autonomic and Secure Computing, 16th International Conference on Pervasive Intelligence and Computing, 4th International Conference on Big Data Intelligence and Computing and Cyber Science and Technology Congress*, DASC/PiCom/DataCom/CyberSciTech '18, pages 151–158. IEEE.

Meng, W., Liu, Y., Zhu, Y., Zhang, S., Pei, D., Liu, Y., Chen, Y., Zhang, R., Tao, S., Sun, P., et al. (2019). LogAnomaly: Unsupervised detection of sequential and quantitative anomalies in unstructured logs. In *Proceedings of the International Joint Conference on Artificial Intelligence*, volume 19 of *IJCAI '19*, pages 4739–4745.

Panichella, A., Dit, B., Oliveto, R., Di Penta, M., Poshynanyk, D., and De Lucia, A. (2013). How to effectively use topic models for software engineering tasks? an approach based on genetic algorithms. In *Proceedings of the 35th International Conference on Software Engineering*, ICSE '13, pages 522–531. IEEE/ACM.

Shahin, M., Ali Babar, M., and Zhu, L. (2017). Continuous integration, delivery and deployment: A systematic review on approaches, tools, challenges and practices. *IEEE Access*, 5:3909–3943.

Sievert, C. and Shirley, K. (2014). LDAvis: A method for visualizing and interpreting topics. In *Proceedings of the Workshop on Interactive Language Learning, Visualization, and Interfaces*, pages 63–70. ACL.

Wittkopp, T., Wiesner, P., Scheinert, D., and Kao, O. (2022). A taxonomy of anomalies in log data. In *Proceedings of the 20th International Conference on Service-Oriented Computing*, ICSOC '22, pages 153–164. Springer.

Yang, L., Chen, J., Wang, Z., Wang, W., Jiang, J., Dong, X., and Zhang, W. (2021). Semi-supervised log-based anomaly detection via probabilistic label estimation. In *Proceedings of the 43rd International Conference on Software Engineering*, ICSE '21, pages 1448–1460. IEEE/ACM.

Zhang, X., Xu, Y., Lin, Q., Qiao, B., Zhang, H., Dang, Y., Xie, C., Yang, X., Cheng, Q., Li, Z., Chen, J., He, X., Yao, R., Lou, J.-G., Chintalapati, M., Shen, F., and Zhang, D. (2019). Robust log-based anomaly detection on unstable log data. In *Proceedings of the 27th Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ESEC/FSE '19, pages 807–817. ACM.

Zhao, N., Wang, H., Li, Z., Peng, X., Wang, G., Pan, Z., Wu, Y., Feng, Z., Wen, X., Zhang, W., Sui, K., and Pei, D. (2021). An empirical investigation of practical log anomaly detection for online service systems. In *Proceedings of the 29th Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ESEC/FSE '21, pages 1404–1415. ACM.