SHORT-PAPER

Grammar-Based Anomaly Detection of Microservice Systems Execution Traces

ANDREA D'ANGELO, University of L'Aquila, L'Aquila, AQ, Italy

GIORDANO D'ALOISIO, University of L'Aquila, L'Aquila, AQ, Italy

Published: 07 May 2024

Citation in BibTeX format

ICPE '24: 15th ACM/SPEC International Conference on Performance Engineering
May 7 - 11, 2024
London, United Kingdom

Conference Sponsors:
SIGMETRICS
SIGSOFT

Grammar-Based Anomaly Detection of Microservice Systems Execution Traces

Andrea D'Angelo
andrea.dangelo6@graduate.univaq.it
DISIM Department
University of L'Aquila
L'Aquila, Italy

Giordano d'Aloisio
giordano.daloisio@graduate.univaq.it
DISIM Department
University of L'Aquila
L'Aquila, Italy

ABSTRACT

Microservice architectures are a widely adopted architectural pattern for large-scale applications. Given the large adoption of these systems, several works have been proposed to detect performance anomalies starting from analysing the execution traces. However, most of the proposed approaches rely on machine learning (ML) algorithms to detect anomalies. While ML methods may be *effective* in detecting anomalies, the training and deployment of these systems as been shown to be less *efficient* in terms of time, computational resources, and energy required.

In this paper, we propose a novel approach based on Context-free grammar for anomaly detection of microservice systems execution traces. We employ the SAX encoding to transform execution traces into strings. Then, we select strings encoding anomalies, and for each possible anomaly, we build a Context-free grammar using the Sequitur grammar induction algorithm. We test our approach on two real-world datasets and compare it with a Logistic Regression classifier. We show how our approach is more effective in terms of training time of ∼15 seconds with a minimum loss in effectiveness of ∼5% compared to the Logistic Regression baseline.

CCS CONCEPTS

• **Software and its engineering → Software performance**; *Formal language definitions*; • **Theory of computation → Grammars and context-free languages**.

KEYWORDS

Anomaly Detection,Execution Traces,Context-free Grammar,Micro Service System

ACM Reference Format:
Andrea D'Angelo and Giordano d'Aloisio. 2024. Grammar-Based Anomaly Detection of Microservice Systems Execution Traces. In *Companion of the 15th ACM/SPEC International Conference on Performance Engineering (ICPE '24 Companion), May 7–11, 2024, London, United Kingdom.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3629527.3651844

1 INTRODUCTION

Microservice architecture is nowadays one of the most adopted architectural patterns to develop large-scale systems (like Netflix, Amazon, Facebook, and others) [1]. In general, a microservice system can be represented as a network of individually deployed systems, each one devoted to a single specific task (i.e., a *microservice*). By interacting with each other, the different microservices allow the completion of more complex tasks for the end user. Given the wide adoption of this architectural style, several studies have been conducted to tackle performance anomalies of these kinds of systems. In particular, anomaly detection of microservice systems is a widely addressed topic in the literature [26]. However, most of the methods proposed employ machine learning (ML) algorithms to address this task. While the adoption of these approaches can be *effective* in terms of anomaly detection, the training and deploying of ML methods is generally not *efficient* in terms of required training time, computational resources, and energy consumption [11, 18, 21].

In this work, we move towards a more energy-efficient anomaly detection in microservice systems execution traces by presenting an innovative approach based on formal Context-Free grammar. We employ SAX encoding [25] to transform execution traces into strings, and then infer a grammar from the set of strings that encode anomalies. The constructed grammar functions as an anomaly detector, enabling the encoding and membership check of any new measurement. We present a first implementation using Sequitur [22] for grammar induction. We compare the training time and effectiveness scores of our approach against Logistic Regression. Results show that the grammar-based approach achieves comparable effectiveness while requiring significantly less time for training.

The remainder of the paper is structured as follows: in Section 2 we discuss some related works; Section 3 presents in detail the proposed approach; Section 4 shows the experimental evaluation we conducted to assess the *efficiency* (in terms of required time) and *effectiveness* of our approach; finally Section 5 presents some future works and concludes the paper.

2 RELATED WORKS

The problem of performance analysis, and in particular, anomaly detection in the performances of microservice systems, has been widely studied by the literature [26].

Among all the proposed approaches, we observe how most of them employ ML techniques. For instance, Bensal *et al.* proposed DeCaf, a system for automated diagnosis and triaging of KPI issues using service logs [4]. The proposed approach uses machine learning along with pattern mining to help service owners automatically root cause and triage performance issues. Similarly, Du *et al.* presented a system for anomaly detection in the performances of container-based microservice systems [10]. The proposed approach

ICPE '24 Companion, May 7–11, 2024, London, United Kingdom
© 2024 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-0445-1/24/05
https://doi.org/10.1145/3629527.3651844

consists of a monitoring module that collects the performance data of containers, a data processing module based on machine learning models and a fault injection module integrated for training these models. An approach employing several ML algorithms is the one proposed by Jin *et al.* [14]. The authors perform two different anomaly detection analyses in their work: invocation chain anomaly analysis based on robust principal component analysis and a single indicator anomaly detection algorithm. The single indicator anomaly detection algorithm comprises an Isolation Forest algorithm, a One-Class Support Vector Machine algorithm, a Local Outlier Factor algorithm, and the 3σ principle. Finally, Wu *et al.* employed a Deep Learning model for performance diagnosis in cloud-based microservice systems [29]. All the described approaches employ ML techniques, which may be effective but also require high computational resources and time for their training [21]. Moreover, ML models are often black-box or challenging to interpret [8].

A different approach for anomaly detection in microservice systems is the one proposed by Traini *et al.* [28]. In their work, the authors proposed a search-based approach for diagnosing performance issues in service-based systems. In our work, we move towards the same direction of not employing ML-based techniques to identify anomalies in the performances of microservice systems. In particular, we first transform each execution trace (i.e., a sequence of response times of remote calls to different microservices) into a string using the Symbolic Aggregate Approximation (SAX) encoding [25], which is a well-established technique for anomaly detection [6, 13]. Next, we employ a context-free-grammar-based approach to identify if the string representation of the execution trace contains an anomaly or not.

3 METHODOLOGY

In this section, we formally define our methodology for grammar-based anomaly detection. Figure 1 depicts the methodology in all its components. We first detail the methodology for Grammar Construction, then move onto the process of Membership Testing.

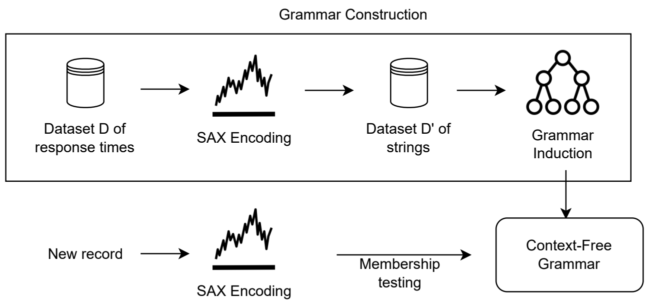

Figure 1: The proposed methodology involves a one-time grammar construction process. Then, each new record undergoes membership checking against the grammar.

3.1 Grammar Construction

We first focus on the one-time process of Grammar Construction. Starting from dataset D of response times, we define the SAX Encoding as the function $SAX : \mathbb{R}^n \to \Sigma^n$, where Σ represents a predetermined set of characters. SAX takes a set of real numbers i as input and maps them to a string of characters s such that $|i| = length(s)$. By encoding all the response times in dataset D with SAX encoding, we obtain a dataset D' of labelled strings. From D' we select the execution traces (i.e., set of response times) showing anomalies. Then, for each anomaly category A, we apply a Grammar induction algorithm to the execution traces having that specific anomaly A, to obtain a Context-Free Grammar $G = (V, T, P, S)$, where V is the set of non-terminal symbols, T is the set of terminal symbols, P is the set of productions and S is the starting symbol. The grammar G must have the following properties:

(1) $T = \Sigma$
(2) $L(G) = \{SAX(i) \mid i \text{ is labeled as an anomaly}\}$

The first property ensures that the set of terminal symbols recognized by the grammar is the same as the set of characters obtained via SAX encoding. For this reason, the domain-specific choice of Σ for SAX Encoding plays an important role in the resulting grammar. A Σ encompassing numerous characters enhances the precision of anomaly detection, yet it may lead to an unwarranted increase in the size of the grammar. Conversely, a Σ comprising only a few characters may not adequately discern subtle anomalies.

The second property fixes the language generated from G as the set of strings obtained via Sax Encoding and labelled as anomalies in the starting dataset D. Note that the second property does not imply that only those strings must be employed to build the grammar. Several grammar induction algorithms also consider negative examples as an aid to build the resulting grammar.

The grammar G was chosen to be Context-Free as it is able to represent intrinsic links inside of the string that is useful in our context. In instances where a microservice relies on several others, anomalies in their response times may be interconnected. It is crucial that the chosen grammar can accurately deduce that an elevated response time for one microservice could be contingent on another. Achieving this level of inference is not feasible with Regular Grammars or Regex, making the use of a Context-Free Grammar essential.

3.2 Membership Testing

Once the grammar G is built from the grammar induction algorithm, it effectively functions as a model for anomaly detection. We can now test new sets of real numbers for anomalies. When a new execution trace j arrives, it must be processed with the same SAX function used for grammar construction. Then:

- if $SAX(j) \in L(G)$, j contains an anomaly.
- if $SAX(j) \notin L(G)$, there is no anomaly.

The process of understanding if a string is part of the language generated by a grammar is known as Membership checking. Several Python libraries provide convenient methods for this task (e.g., NLTK [5]).

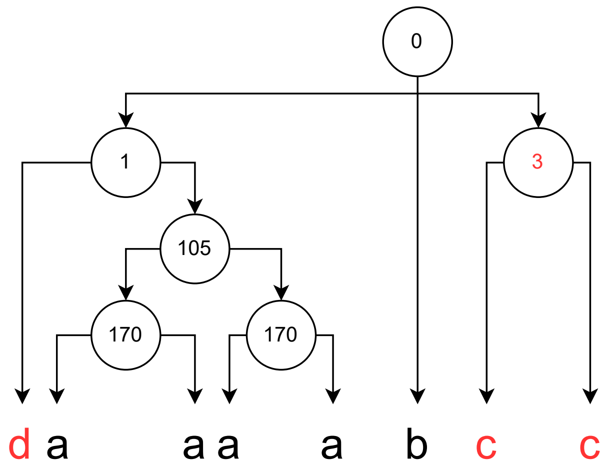

Figure 2: Parse tree generated when checking the membership of a string that includes an anomaly in our grammar. The final two microservices exhibit values above the typical range, impacting the overall response time (initial character).

```
0 -> 1 'b' 3
1 -> 'd' 105
105 -> 170 170
170 -> 'a' 'a'
170 -> 'a' 'a'
3 -> 'c' 'c'
```

Listing 1: Grammar productions involved in the membership checking of string "daaaabcc".

Whenever a string appears to be part of a grammar, the membership check also produces a parse tree of productions starting from the initial symbol of the grammar S to the given string. For instance, listing 1 depicts the necessary productions to derivate the string "daaaabcc" (which contains an anomaly) from the starting symbol of our grammar. The derivation can also be represented in a visually intuitive way by generating a parse tree. For instance, Figure 2 portrays a parse tree generated when the aforementioned string is processed by our grammar. The parse tree provides a visual representation that aids in understanding the precise locations of anomalies within the string. By examining the tree structure, we can easily pinpoint the specific steps and grammar productions leading to the anomalous elements.

4 EVALUATION

In this section, we describe the experimental evaluation conducted to assess the *efficiency* and *effectiveness* of our approach. In particular, we aim to answer the following two research questions (RQ):

RQ1. How much time does the proposed approach require to construct a grammar compared to the training time of standard ML methods?

RQ2. How effective is the proposed approach in detecting anomalies compared to standard ML methods?

In both experiments, we employ a Logistic Regression (LogReg) classifier [19] as a baseline. We have chosen this method among the possible classification approaches because it natively supports multi-class classification (i.e., classification problems where the number of possible values of the label is higher than two [2]) and because it usually achieves a good trade-off between prediction's effectiveness and training time compared to other classification models [17]. We adopted the implementation provided by the `scikit-learn` Python library [23].

Concerning our approach, we employ the Sequitur algorithm [22] for the grammar induction phase shown in figure 1. Sequitur is an algorithm that allows the generation of context-free grammars starting from a sequence of strings by replacing repeated phrases with a grammatical rule that generates the phrase. It repeats this process recursively until all the strings are examined.

As a use case, we employ the dataset provided by Traini *et al.* in [28][1]. The dataset comprises 560 CSV files containing pre-processed execution traces (i.e., series of response times of remote procedure calls to different microservices) with injected anomalies originating from two open-source microservices systems: Train-Ticket [16] and E-Shopper[2]. Each scenario features two possible anomalies, identified by the `anomaly` column.

In the following, we describe how we addressed **RQ1**. Next, we detail the answer to **RQ2**. The complete replication package of the experiment is available in Zenodo [9].

4.1 Addressing RQ1

The first research question focuses on the amount of time required by the proposed approach to generate context-free grammars compared to the amount of time needed for the LogReg classifier to train on a specific dataset. To answer this question, we computed the grammar construction and training times twenty times to avoid possible measurement biases. It is worth noticing how the grammar construction phase encompasses both the SAX encoding of the dataset and the grammar induction, as shown in figure 1. This experiment has been executed on a DELL XPS 13 2019 with an Intel i7 processor, 16 GB of RAM and Windows 11 operating system.

Table 1: Comparison of means and standard deviation of Logistic Regression training and Grammar Construction times in seconds.

	E-Shopper	Train-Ticket
Grammar	5.848 ± 0.506	7.724 ± 0.309
LogReg	18.833 ± 2.933	21.556 ± 2.863

Table 1 presents the mean and standard deviation of LogReg training time and grammar construction time for both E-Shopper and Train-Ticket datasets. As can be seen, our approach requires ~13 seconds less to construct a grammar compared to the training time of the LogReg classifier for the E-Shopper use case and ~14 less for the Train-Ticket use case. In addition, we note how the

[1]https://github.com/SpencerLabAQ/icpe-data-challenge-delag
[2]https://github.com/SEALABQualityGroup/E-Shopper

grammar construction time has less variability compared to the training time of the model.

Answer to RQ1
Our proposed approach requires a time to construct a grammar that is ~13 seconds lower compared to the training time of a LogReg classifier for the E-Shopper use case and ~14 seconds lower for the Train-Ticket use case. Moreover, the time required to build the grammar is almost constant over different runs.

4.2 Addressing RQ2

The second research question focuses on the effectiveness of our proposed approach in detecting anomalies compared to a LogReg classifier. To answer this question, we used our approach to detect the anomalies on both E-Shopper and Train Ticket datasets and compared their effectiveness with the LogReg baseline. More in detail, for each dataset, we perform a train-test split and use 80% of the data to build the grammar/train the LogReg model, and we predict anomalies on the remaining 20%. We employ accuracy [24], precision, and recall [7] scores as effectiveness metrics. Concerning the setting of hyper-parameters, for LogReg, we used the default ones provided by the `scikit-learn` library. Instead, the only hyper-parameter required by our approach is the number of bins used by the SAX encoder algorithm [25], which we set to 5. We tested different values of this parameter and found that 5 achieves the highest effectiveness in both use cases.

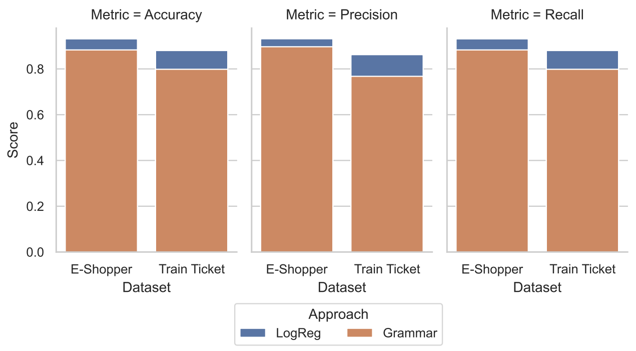

Figure 3: Comparison of accuracy, precision and recall scores of Logistic Regression and our grammar-based approach.

Figure 3 reports the accuracy, precision and recall scores for E-Shopper and Train Ticket use cases. The blue bar shows the results achieved by LogReg while the orange bar displays the results achieved by our grammar-based approach. As can be seen, our approach has an effectiveness that is almost comparable to the one achieved by the LogReg model, with a difference of, at most, 10% in the precision score for the Train Ticket dataset. However, we also observe how, in general, the LogReg classifier has a lower effectiveness in the Train Ticket dataset, meaning that the anomaly patterns are less evident in this use case.

Answer to RQ2
Our approach has an effectiveness that is almost comparable to the one achieved by a LogReg classifier, with a delta of at most 10% in a use case with less evident anomaly patterns.

4.3 Discussion

The performed experiments showed how our proposed approach achieves a higher efficiency in terms of time required to construct the grammar, with a little cost in terms of prediction effectiveness compared to a Logistic Regression classifier. However, it is worth noticing how the efficiency and effectiveness of our approach are also related to the algorithms employed for SAX encoding and grammar induction. Concerning the SAX encoding, in this preliminary work, we employ an implementation of the classical algorithm proposed in [25]. However, other versions of the algorithm have been proposed in the literature that may better detect the differences in anomaly execution traces [20, 27, 30].

The same can be said for the grammar induction algorithm. In this work, we employ the Sequitur algorithm which is one of the most adopted algorithms for grammar induction. However, the grammars generated by this algorithm are often too large and not optimal. Finding the minimum grammar representing a specific language is known to be an NP-complete problem [12]. Nevertheless, some works have been proposed in the last years that try to achieve this task [3, 15].

5 CONCLUSION AND FUTURE WORK

In this paper, we introduced an innovative method for anomaly detection utilizing Context-free Grammar, serving as an alternative to Machine Learning techniques. We formally outlined our methodology and introduced an initial implementation using a naive grammar induction algorithm. We then presented preliminary results, which demonstrated comparable effectiveness to Logistic Regression with minimal training time requirements. The presented methodology is adaptable and can be easily tuned based on domain-specific parameters. In addition, the grammar induction algorithm can be interchangeable based on specific needs. Future work avenues include exploring more sophisticated SAX encoding algorithms like the ones proposed in [20, 27, 30], along with an automatic approach for the selection of Σ. Next, we plan to investigate other grammar induction algorithms like ARVADA [15], which can produce more readable and shorter context-free grammars. Another possible approach would be the hand-crafting of regular expressions or grammars by domain experts. Our methodology can also be easily expanded to account for explainability needs. By constructing the parse tree of a given string, we can easily visualize where the anomaly happened and what microservices it encompassed.

ACKNOWLEDGMENTS

Part of the numerical simulations have been realized on the Linux HPC cluster Caliban of the High-Performance Computing Laboratory of the Department of Information Engineering, Computer Science and Mathematics (DISIM) at the University of L'Aquila.

REFERENCES

[1] Nuha Alshuqayran, Nour Ali, and Roger Evans. 2016. A systematic mapping study in microservice architecture. In *2016 IEEE 9th International Conference on Service-Oriented Computing and Applications (SOCA)*. IEEE, 44–51.

[2] Mohamed Aly. 2005. Survey on multiclass classification methods. *Neural Netw* 19, 1 (2005), 9. Publisher: Citeseer.

[3] Mohammad Rifat Arefin, Suraj Shetiya, Zili Wang, and Christoph Csallner. 2024. Fast Deterministic Black-box Context-free Grammar Inference. arXiv:2308.06163 [cs.SE]

[4] Chetan Bansal, Sundararajan Renganathan, Ashima Asudani, Olivier Midy, and Mathru Janakiraman. 2020. DeCaf: diagnosing and triaging performance issues in large-scale cloud services. In *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering: Software Engineering in Practice* (Seoul, South Korea) *(ICSE-SEIP '20)*. Association for Computing Machinery, New York, NY, USA, 201–210. https://doi.org/10.1145/3377813.3381353

[5] Steven Bird. 2006. NLTK: the natural language toolkit. In *Proceedings of the COLING/ACL 2006 Interactive Presentation Sessions*. 69–72.

[6] Konstantinos Bountrogiannis, George Tzagkarakis, and Panagiotis Tsakalides. 2021. Anomaly Detection for Symbolic Time Series Representations of Reduced Dimensionality. In *2020 28th European Signal Processing Conference (EUSIPCO)*. 2398–2402. https://doi.org/10.23919/Eusipco47968.2020.9287474

[7] Michael Buckland and Fredric Gey. 1994. The relationship between recall and precision. *Journal of the American society for information science* 45, 1 (1994), 12–19. Publisher: Wiley Online Library.

[8] Diogo V Carvalho, Eduardo M Pereira, and Jaime S Cardoso. 2019. Machine learning interpretability: A survey on methods and metrics. *Electronics* 8, 8 (2019), 832.

[9] Andrea D'Angelo and Giordano d'Aloisio. 2024. *Grammar-Based Anomaly Detection of Microservice Systems Execution Traces Replication Package*. https://doi.org/10.5281/zenodo.10806012

[10] Qingfeng Du, Tiandi Xie, and Yu He. 2018. Anomaly detection and diagnosis for container-based microservices with performance monitoring. In *Algorithms and Architectures for Parallel Processing: 18th International Conference, ICA3PP 2018, Guangzhou, China, November 15-17, 2018, Proceedings, Part IV 18*. Springer, 560–572.

[11] Raphael Fischer, Matthias Jakobs, Sascha Mücke, and Katharina Morik. 2022. A Unified Framework for Assessing Energy Efficiency of Machine Learning. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*. Springer, 39–54.

[12] E Mark Gold. 1978. Complexity of automaton identification from given data. *Information and Control* 37, 3 (1978), 302–320. https://doi.org/10.1016/S0019-9958(78)90562-4

[13] Fabio Guigou, Pierre Collet, and Pierre Parrend. 2019. SCHEDA: Lightweight euclidean-like heuristics for anomaly detection in periodic time series. *Applied Soft Computing* 82 (2019), 105594. https://doi.org/10.1016/j.asoc.2019.105594

[14] Mingxu Jin, Aoran Lv, Yuanpeng Zhu, Zijiang Wen, Yubin Zhong, Zexin Zhao, Jiang Wu, Hejie Li, Hanheng He, and Fengyi Chen. 2020. An Anomaly Detection Algorithm for Microservice Architecture Based on Robust Principal Component Analysis. *IEEE Access* 8 (2020), 226397–226408. https://doi.org/10.1109/ACCESS.2020.3044610

[15] Neil Kulkarni, Caroline Lemieux, and Koushik Sen. 2021. Learning Highly Recursive Input Grammars. In *2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. 456–467. https://doi.org/10.1109/ASE51524.2021.9678879

[16] Bowen Li, Xin Peng, Qilin Xiang, Hanzhang Wang, Tao Xie, Jun Sun, and Xuanzhe Liu. 2022. Enjoy your observability: an industrial survey of microservice tracing and analysis. *Empirical Software Engineering* 27 (2022), 1–28.

[17] Tjen-Sien Lim, Wei-Yin Loh, and Yu-Shan Shih. 2000. A Comparison of Prediction Accuracy, Complexity, and Training Time of Thirty-Three Old and New Classification Algorithms. *Machine Learning* 40, 3 (Sept. 2000), 203–228. https://doi.org/10.1023/A:1007608224229

[18] Francesca Marzi, Giordano d'Aloisio, Antinisca Di Marco, and Giovanni Stilo. 2023. Towards a Prediction of Machine Learning Training Time to Support Continuous Learning Systems Development. *arXiv preprint arXiv:2309.11226* (2023).

[19] Scott Menard. 2002. *Applied logistic regression analysis*. Vol. 106. Sage.

[20] Muhammad Marwan Muhammad Fuad. 2012. Genetic algorithms-based symbolic aggregate approximation. In *Data Warehousing and Knowledge Discovery: 14th International Conference, DaWaK 2012, Vienna, Austria, September 3-6, 2012. Proceedings 14*. Springer, 105–116.

[21] Nadia Nahar, Haoran Zhang, Grace Lewis, Shurui Zhou, and Christian Kästner. 2023. A Meta-Summary of Challenges in Building Products with ML Components – Collecting Experiences from 4758+ Practitioners. https://doi.org/10.48550/arXiv.2304.00078 arXiv:2304.00078 [cs].

[22] Craig G Nevill-Manning and Ian H Witten. 1997. Identifying hierarchical structure in sequences: A linear-time algorithm. *Journal of Artificial Intelligence Research* 7 (1997), 67–82.

[23] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. 2011. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research* 12 (2011), 2825–2830.

[24] G.H. Rosenfield and K. Fitzpatrick-Lins. 1986. A coefficient of agreement as a measure of thematic classification accuracy. *Photogrammetric Engineering and Remote Sensing* 52, 2 (1986), 223–227. http://pubs.er.usgs.gov/publication/70014667

[25] Hagit Shatkay and Stanley B Zdonik. 1996. Approximate queries and representations for large data sequences. In *Proceedings of the Twelfth International Conference on Data Engineering*. IEEE, 536–545.

[26] Jacopo Soldani and Antonio Brogi. 2022. Anomaly Detection and Failure Root Cause Analysis in (Micro) Service-Based Cloud Applications: A Survey. *ACM Comput. Surv.* 55, 3, Article 59 (feb 2022), 39 pages. https://doi.org/10.1145/3501297

[27] Youqiang Sun, Jiuyong Li, Jixue Liu, Bingyu Sun, and Christopher Chow. 2014. An improvement of symbolic aggregate approximation distance measure for time series. *Neurocomputing* 138 (2014), 189–198.

[28] Luca Traini and Vittorio Cortellessa. 2023. DeLag: Using Multi-Objective Optimization to Enhance the Detection of Latency Degradation Patterns in Service-Based Systems. *IEEE Transactions on Software Engineering* 49, 6 (2023), 3554–3580. https://doi.org/10.1109/TSE.2023.3266041

[29] Li Wu, Jasmin Bogatinovski, Sasho Nedelkoski, Johan Tordsson, and Odej Kao. 2020. Performance diagnosis in cloud microservices using deep learning. In *International Conference on Service-Oriented Computing*. Springer, 85–96.

[30] Yufeng Yu, Yuelong Zhu, Dingsheng Wan, Huan Liu, and Qun Zhao. 2019. A novel symbolic aggregate approximation for time series. In *Proceedings of the 13th International Conference on Ubiquitous Information Management and Communication (IMCOM) 2019 13*. Springer, 805–822.