






Log Attention – Assessing Software Releases with Attention-Based Log Anomaly Detection

Sohail Munir¹ , Hamid Ali² , and Jahangeer Qureshi² 

¹ Dubai Digital Authority, Dubai, UAE
lsohail.munir@digitaldubai.ae

² Xeric.ai, Dubai, UAE

Abstract. A Software Engineering Manager (EM) has to cater to the demand for higher reliability and resilience in Production while simultaneously addressing the evolution of software architecture from monolithic applications to multi-cloud distributed microservices. Pre-release functional testing is no longer sufficient to eliminate faults as more and more issues are generated at runtime, which is challenging to diagnose due to complex inter-service dependencies and dynamic late binding of services. Bugs in Production are known to propagate across software components and become critical as they go undetected.

This paper introduces LogAttention, a methodology based on analysis of runtime logs that provides actionable insights to the EM to identify faults and preempt failure in Production. LogAttention is a Log Anomaly Detection (LAD) technique that uses Attention-based Transformer Models to identify Anomalous Log Messages. LogAttention assigns a quality score to the software release in Production and presents remarkable logs to the EM to analyze, predict, and preempt failure. This paper presents empirical evidence showing that LogAttention outperforms existing LAD techniques to identify anomalous log messages and ensure that the detected log anomalies are reliable indicators of the health of a software release.

Keywords: AIOps · Attention models · Log analysis · Log anomaly detection · Log attention · Software release quality

1 Introduction

Software today has evolved from large monolithic applications deployed on-premises at enterprise data centers to multi-cloud, cloud-native distributed applications based on Microservice Architecture (MSA) [1]. The Software Engineering Manager (EM) in a Software Product Organization (SPO) has the responsibility to build and support software applications that should be scalable, resilient, and robust while adapting to the dynamic and evolving user requirements. However, at the same time, MSA brings the complexity of having hundreds or thousands of interdependent microservices which become cumbersome to manage in production.

An EM can use traces, metrics, or logs to detect bugs in Production [2, 3]. In most cases, logs are the only available data that record software runtime information; however,

a log-based approach must deal with the enormous volume of log data averaging gigabytes of data per hour for a typical commercial cloud application. Unlike pre-release testing, an EM cannot directly look for bugs in a running production environment. Instead, she relies on advanced techniques to look for the effects of bugs. She could use Log Anomaly Detection (LAD) to find anomalous logs and expect that some bug has caused the anomalies. Anomalous system behavior could mean functional inconsistencies, performance degradation, system compromise, or a change in log patterns and would indicate that some possible bugs exist, and the EM should proactively discover these bugs and address them.

This paper presents a methodology for assessing software release quality and identifying bugs in Production, based only on runtime logs. The proposed model uses LAD based on Attention-based Transformer Models to present remarkable logs to the EM along with a numeric score depicting the health of the software release. The research evaluates the proposed LogAttention model against three log datasets which include two different releases SE-A and SE-C of a software product SE having more than 50,000 daily users and one release DN-C of a software product DN having more than 100,000 monthly active users and more than 500,000 total users. The results demonstrate the superior performance of LogAttention over the current techniques.

2 Methodology

The paper proposes LogAttention, a methodology for assessing the quality of a software release in Production based on runtime logs. The proposed methodology uses Attention-based transformer models to classify log messages as Remarkable. A log message classified as remarkable would mean that the model predicts it as anomalous and representing a bug in Production.

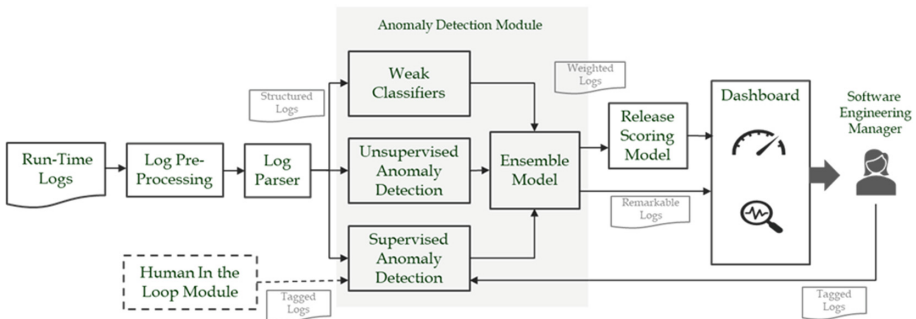


Fig. 1. LogAttention model

The EM can view the remarkable log messages, analyze the log's sequence, and determine if it represents a bug that she should address. The objective is for LogAttention to provide a consistently accurate prediction and a very low detection prevalence for the recommendations to be meaningful for the EM. Figure 1 illustrates the LogAttention methodology explained in the subsequent sections.

The LogAttention model presented in this paper comprises multiple modules. These modules can be categorized based on their role in data understanding, preparation, and analysis. This order also reflects the progression of our methodology. Each section explains our approach to data understanding, preparation, and analysis, along with the description of corresponding models and functions.

2.1 Log Preprocessing

In LogAttention, there is a Log Pre-processing module that cleanses the logs to parse them efficiently. The preprocessing function is fine-tuned to each new production environment to cleanse the log files and automatically extract a log format as an input to log parsers. This results in a “burn-in” period which is essential to determine the most appropriate log parser by testing and evaluating results on the preprocessed log data.

Logs contain an abundant amount of information regarding software activity, e.g., events, parameters, execution details. While there is no fixed structure, a log message typically consists of a variable part (log parameter) and a constant part (log event). We preprocess the log files using the following preprocessing functions:

1. In the scenario where the date is not provided in the logs, a function is used to find the date in the log file name.
2. Verify if a given string is the beginning of a logline.
3. Clean a log file by appending log traces and outputs into a singular line.
4. Remove any log headings for the log file to remove any initial redundant log strings.
5. Extract the log format by identifying Date, Time, Log Level, and Content Token positions.

However, after these preprocessing functions have been applied the logs are still not in a state where they can be processed efficiently. To parse these logs effectively log parsers are essential. After an exhaustive literature review for recent advances in Log Parsing, the Log Parsers that we found of particular interest were DRAIN [4], a Log Structure Heuristics-based Log Parser, and NuLog [5] a neural network-based Log Parser.

These parsers demonstrated impressive performance, yet both seemed to function with varying efficiency based on the amount of data present for parsing. While NuLog outperforms DRAIN on large log files, when insufficient data is available, DRAIN has been observed to be a better choice. Nevertheless, this uncertainty makes a burn-in period essential to parse any new log datasets properly. An implementation of the technique should offer the EM to choose between NuLog, DRAIN, or other viable log parsers.

2.2 Log Anomaly Detection Module

Once the structured logs are extracted, log data is input to the Log Anomaly Detection (LAD) Module. The Machine Learning based LAD Module consists of the Weak Classification (WC) Module, the Supervised Anomaly Detection (SAD) Module, and the Unsupervised Anomaly Detection (USAD) Module.

Weak Classification (WC) Module – WC is based on heuristics from the analysis of different log datasets. Their composition and distribution of anomalies in these datasets help reach immediate approximation. This module classifies each log message as either Normal or Remarkable (i.e., possibly anomalous). This module allows the system to classify logs without any training data despite being less accurate. Based on heuristic measures, this module guides the system into a relevant solution space that further allows the curation of an initial dataset for labeling saving computing overhead. The WC module classified data uses three distinct labeling functions as follows:

Log Level Classifier – A log message contains an associated log level that gives a rough guide to the message’s importance and urgency, like INFO, WARNING, or ERROR. Log Level-based classification function utilizes this log level to classify the incoming log messages, and any level other than the “INFO” level is classified as “Remarkable.”

Double σ Classifier – A standard log message contains a timestamp that denotes the exact time of the event that has triggered the log. Calculating the differential between the timestamp of the current log and that of the previous log helps estimate the event’s execution time that triggered the previous log. This allows the function to calculate the mean execution time for every unique log message. If the execution time of a log message exceeds by two standard deviations, it indicates an abnormal instance of that log message. The EM may manually adjust the standard deviation limits beyond which an event will be classified/labeled as remarkable. This customization helps the EM adjust the system based on historical performance/operational context, thus contributing to early classification accuracy.

Error Lexicon Classifier – Each log entry contains a message that holds information regarding the event that triggered it. This message comprises alphanumeric content that helps understand the nature of the event. By using a lexicon of common error terminologies which occur in log messages, the Error Lexicon-based classification function classifies logs with mentions of erroneous tokens as “Remarkable.”

This function’s efficiency depends on the accuracy of the lexicon dictionary being used as a reference. To create a thorough and efficient lexicon, this research followed a novel approach using issues from GitHub Activity Data, containing the keywords related to faults like Errors and Exceptions. The Error Lexicon is created by scraping over one million issues of top-rated GitHub repositories, of which around 50,000 issues are related to either an exception or an error.

Unsupervised Anomaly Detection (USAD) Module – USAD refers to the practice of detecting statistical outliers from a dataset with no reliance on labels. This methodology is especially beneficial for the classification of a dataset with no prior training. Additionally, the possible biases in a human-labeled training dataset are also nullified, thus allowing the detection of anomalies that are not explicitly sought out by human users. USAD has two steps, learning representative embeddings from the datasets and classifying the logs as either Normal or Anomalous.

Embeddings – Word Embeddings or Word vectorization is a methodology in Natural Language Processing (NLP) that maps words or phrases from vocabulary to a corresponding vector of real numbers. This mapping allows downstream functions to solve

language-related problems such as word predictions, word similarities, and semantics. Log messages contain strings of information in an alphanumeric format, and any analysis of this data, including LAD, requires the conversion of this information into the downstream-functions-ready format.

Attention-based Transformer Models – Attention in neural networks is a mechanism that a model can learn to make predictions by focusing on (attending to) a small sub-set of data. To predict or infer one element (sequence), the model estimates using the attention vector how strongly it is correlated with (“attends to”) other elements (sequences) and takes the sum of their values weighted by the attention vector as the approximation of the target. Self-Attention is a category of Attention models that “attends to” different positions in the same input sequence [6].

Transformer is an architecture based on Self-Attention models that transform one sequence into another with the help of the Encoder and Decoder [7]. Transformers do not require sequential data to be processed in order, allowing massive parallelization during training, enabling training on very large datasets very fast.

Language Modeling – Language modeling is a technique that builds language models to help predict a sequence of recognized words and phonemes that are used for real-world problems relating to Natural Language Processing (NLP) using statistical techniques, neural networks, and deep learning methods [8] specially Attention-based Transformer Models. Specifically, word embedding is adopted to use a real vector representing each word in the project vector space based on their usage, allowing words with a similar meaning to have an equal representation while still retaining their contextual information.

Training a Transformer model for use in a particular NLP task is simple. Start with a randomly initialized Transformer model, put together a huge dataset containing text in the language or languages of interest, pre-train the Transformer on the huge dataset, and fine-tune the pre-trained Transformer on the particular task in question, using the task-specific dataset. The advantage of this method is that only labeled data is needed for the final step of fine-tuning. Language models are categorized as:

1. Causal Language Models (CLM): In CLM, the model tries to predict or generate the next word(s) given a sequence of words.
2. Masked Language Modeling (MLM): In MLM, a certain proportion (token) of the text sequence, which may be a complete word or a part of a word, or a text sequence, is masked, and the model then predicts the token that is masked. The masked token may then be replaced with an actual mask or replaced with another random token from the vocabulary.

Recent research has introduced many high-performing Transformer models that could be used for USAD. This research analyzes multiple light-weight transformer models such as ELECTRA [9], DistilBERT [10], and GloVe [11], as well as traditional machine learning models and tested them on datasets SE-A and DN-C against multiple classifiers. The results indicate that the language models allowed far better representations to be learned for the data and provided better results. Figure 2 shows Macro-F₁ scores on SE-A and DN-C datasets, respectively, in a completely unsupervised setting,

using different classifiers and embeddings. As evident from Fig. 2, DistilBERT gives the best Macro-F₁ scores on both datasets using Self Organising Maps (SOM) [12] and Isolation Forest (IF) [13] classifiers. DistilBERT is therefore selected as the Transformer model of choice, and SOM and IF as the classifiers of choice for the USAD module.

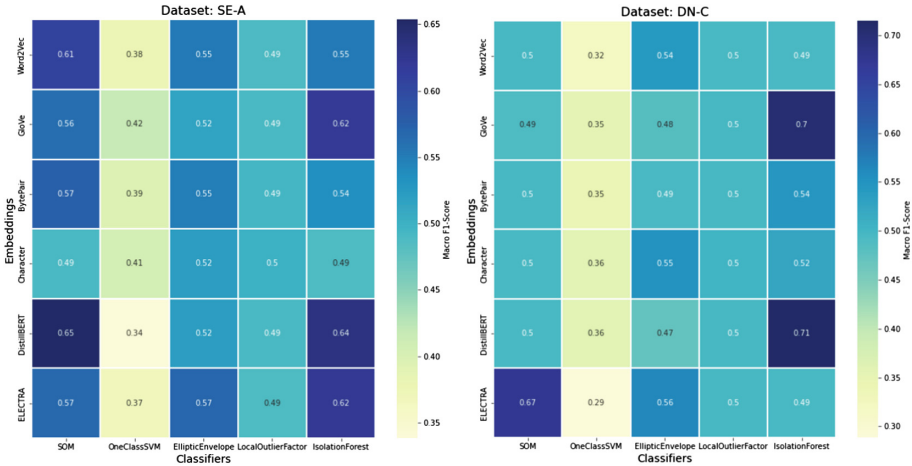


Fig. 2. Model selection of USAD (SE-A and DN-C Datasets)

Supervised Anomaly Detection (SAD) Module – SAD uses active learning to detect anomalies in a supervised paradigm. It uses transfer learning (learning of pre-trained models) and a novel data curation technique for data labeling by the Human-in-the-Loop (HIL), which can be Domain Experts(s), the software developers, or the EM.

The transfer learning and data curation aspects of SAD depicted in Fig. 3 below are as follows:

Pre-trained Model – The pre-trained model is trained on a large dataset comprising real log data, and the weights of this pre-trained model are then applied to the SAD module. When the embedded data is input to the module, it can provide better outputs in lesser iterations based on transfer learning.

Human in the Loop – An important component of training a supervised model is the labeled dataset. This labeling is done by a HIL, who is typically a software engineer from the agile development team who assigns a priority score (i.e., High, Medium, or Low) to each data point (i.e., a log event). While this helps train the supervised model, it also helps the model understand the software engineering team’s priority based on these labels. This customization is essential in helping the model focus on specific aspects the developers might find significant and learn the preferences of different developers in the process while still allowing the system to maintain its search for statistically significant logs. Another significant input comes from the EM, which rates the anomaly detection system’s output, serving as a labeling feedback system to the SAD module. This labeling

serves as an important input for the model because eventually, the model assists the EM in making decisions relating to software release. Since labeling large swathes of data is a tedious task with significant budgetary and computing overheads, the dataset to be labeled needs to be curated smartly. The model leverages the predictions from the Weak Classification (WC) module and the Unsupervised Anomaly Detection (USAD) module to achieve this. In cases of availability of pre-trained models, the Supervised Anomaly Detection (SAD) module predictions are also utilized. It may be noted that the HIL step is needed only once every major release.

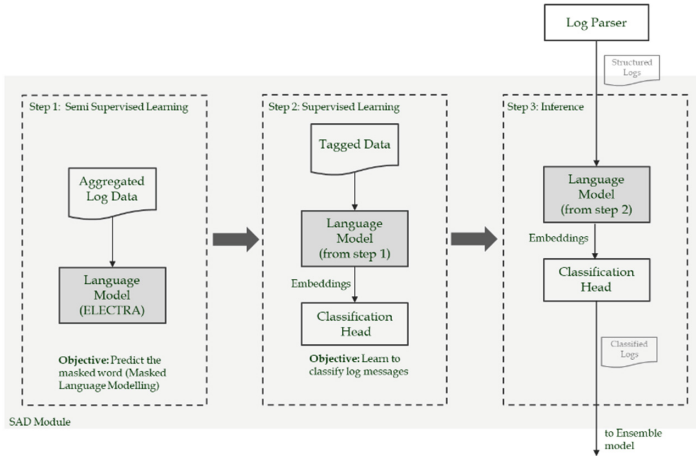


Fig. 3. Supervised anomaly detection module

The paper proposes a novel approach for sampling the data, **MCR Sampling**, to make the size of the dataset tagged by the HIL manageable. The dataset comprises sampling from the following three techniques in equal proportions, **M**arginal Sampling, **C**ertainty Sampling, and **R**andom Sampling. Kazerouni, et al. [14] have proposed Marginal Sampling (Exploitation) and Random Sampling (Exploration) for Active Learning from a skewed dataset. LogAttention HIL model proposes MCR Sampling, which introduces Certainty Sampling to ensure the model also performs well in a completely new environment.

1. **Marginal Sampling** – Marginal sampling involves the data samples for which the model is least certain about the classification, i.e., Remarkable or not. Selecting this type of most informative and uncertain data samples for which the model has less confidence in predicting, i.e., prediction on the margins, improves the model's discriminative ability the most. These samples are close to the decision boundary where uncertainty is the highest, and hence Marginal Sampling is used as an Exploitation method.
2. **Certainty Sampling** – While Marginal sampling helps the model where the model is unsure about the correctness of its labeling, Certainty sampling, on the other hand,

helps the model consolidating its labeling, which the model is fully certain about and, in a way, tests the current state of the model. These samples are usually away from the decision boundary, and the prediction values are over 0.9.

- 3. Random Sampling – Random sampling involves the selection of random data samples as an exploratory method.

Training the Model – LogAttention firstly trains a Transformer Model with a language modeling head to learn the structure of a log file, referred to as the Self-Supervised learning step. In the next step, the Fine-Tuning step, the language modeling head is replaced with a classification head composed of a simple, fully connected layer and trained using the labels generated from the human experts. Once done, the model infers new logs using the Transformer Model with the fine-tuned classification head to attribute the correct label to each data instance.

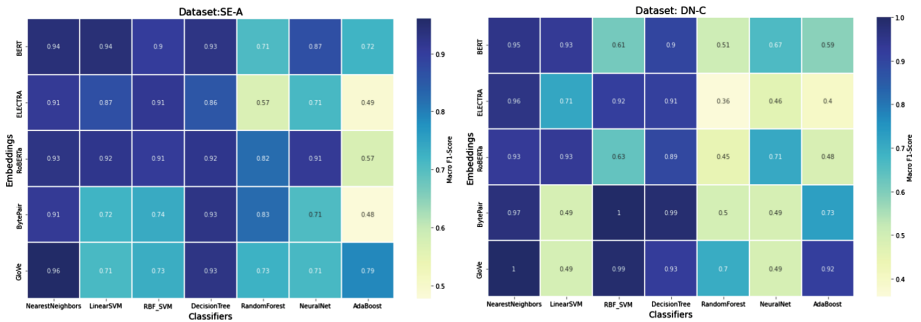


Fig. 4. Model selection of SAD (SE-A and DN-C Datasets)

Model Selection Journey. Like the USAD module, we select a Transformer model for the SAD to benefit from the relatively small amount of data required to train the model and its capacity for transfer learning. As shown in Fig. 4, the transformer models are the most reliable source of embeddings. Additionally, transformer model-based embeddings allow the possibility of zero-shot performance and transfer learning. Given the very slight differences in performance between ELECTRA and RoBERTa [15], we selected ELECTRA as our transformer for classification purposes because it is faster to train than any other model evaluated.

2.3 Ensemble Model

The output from all three ML modules, i.e., WC, USAD, and SAD, is input to the Ensemble model where each classifier is assigned an appropriate weight given its performance on the labeled set of structured log data. The ensemble model outputs a single prediction vector embodying all the best aspects of the previous modules indicating which logs are remarkable. The weights are assigned based on input from tagged data. The EM may adjust the assigned weights based on the production environment as appropriate. In

the future, an optimization model may be developed to assign optimal weights that get updated automatically based on the status changes in the production environment. The Ensemble model outputs the weighted logs to the Release Scoring Model and presents Remarkable logs to the dashboard.

2.4 Release Scoring Model

While the Machine Learning module classifies remarkable logs that help identify anomalous instances from the enormous volume of log data of a production release, this information becomes more useful when presented in an aggregated form that helps represent the state of the release with actionable insights for the EM. The EM is then better placed to make an informed decision based on a more actionable and explainable input.

To further simplify it for the EM, the LogAttention methodology assigns a Release Score (\bigcirc_j) to the release, which is a numerical representation of the release quality, enabling her to gauge the health of the release by referring to a single parameter. This release scoring model is depicted in Eq. 1 below. The release scoring model uses heuristics to assign the weight to the High Priority (Remarkable), Medium Priority, and Low Priority logs. The model provides an interface for the EM to adjust the weights based on the runtime environment. The EM may also tweak the weights based on her experience and preferences. The release score is calculated as follows:

$$\bigcirc_j = 1 - w_p \cdot \left| \min_{0 < i \leq j} \frac{|R_i|}{|L_i|} - \frac{|R_j|}{|L_j|} \right| - (w_r \cdot |R_j| + w_m \cdot |M_j| + w_o \cdot |O_j|) \quad (1)$$

where, j is the number of the current release in a production environment assessed by LogAttention. \bigcirc_i = LogAttention score for the i^{th} release, $L_i = \{\text{Set of Log Messages for the } i^{\text{th}} \text{ release}\}$, $R_i = \{\text{Set of Remarkable/High Priority Log messages for the } i^{\text{th}} \text{ release}\}$, $M_i = \{\text{Set of Medium Priority Log messages for the } i^{\text{th}} \text{ release}\}$, $O_i = \{\text{Set of Low Priority Log messages for the } i^{\text{th}} \text{ release}\}$, w_r = weight assigned to Remarkable/High Priority Logs, w_m = weight assigned to Medium Priority Logs, w_o = weight assigned to Low Priority Logs, and w_p = weight assigned to min detection prevalence.

2.5 Dashboard


LogAttention presents the release score (\bigcirc_j) and the remarkable logs on a dashboard for the EM to gauge the health of the release in production. The LogAttention dashboard allows the EM to conduct detailed analysis by looking at the logs before and after the remarkable logs and fetch further information regarding the issue highlighted in the log message through third-party sources like Stack Overflow. She may also reclassify a certain log as appropriate using the interface provided.

3 Results

The paper evaluates LogAttention against three benchmark models DeepLog, LogAnomaly, and RobustLog. The consolidated results are presented in Table 1.

DeepLog [16] and LogAnomaly [17] are Semi-Supervised models and are two of the best performing LAD techniques; whereas, RobustLog [18] is a Supervised Log Anomaly Detection technique that works very well after training which requires tagging of the entire dataset, but does not work well without training.

Table 1. Model Performance results across datasets.



Model	Logs Processed	Remarkable Logs	Detection Prevalence	TP	FP	TN	FN	Accuracy	Precision	Recall	F1 Score	Macro F1 Score	MCC
<i>SE-A dataset</i>													
DeepLog	32,873	3,764	0.1145	558	3,206	28,445	664	0.8823	0.1482	0.1566	0.2238	0.5801	0.2111
LogAnomaly	32,873	4,130	0.1458	457	3,673	27,978	765	0.8689	0.1497	0.1500	0.2108	0.5886	0.1728
LA (USAD)	32,873	2,664	0.0810	846	1,818	29,833	376	0.9333	0.3176	0.6923	0.4354	0.7000	0.4401
LA (USAD+WC)	32,873	2,664	0.0810	846	1,818	29,833	376	0.9333	0.3176	0.6923	0.4354	0.7000	0.4401
RobustLog (SR)	25,971	2,375	0.0914	430	1,945	23,082	514	0.9053	0.1811	0.4555	0.2591	0.6043	0.2453
RobustLog (FT)	24,012	824	0.0358	791	33	22,124	64	0.9958	0.9600	0.9251	0.9422	0.9700	0.9402
Log Attention (SR)	24,027	697	0.0290	694	3	23,151	179	0.9924	0.9957	0.7950	0.8841	0.9401	0.8862
Log Attention (FT)	24,027	721	0.0360	721	0	23,154	152	0.9937	1.0000	0.8259	0.9046	0.9507	0.9058
<i>SE-C dataset</i>													
DeepLog	106,906	20,911	0.1988	1,245	19,666	83,792	2,203	0.7954	0.0595	0.3611	0.1023	0.4934	0.0762
LogAnomaly	106,906	20,492	0.1917	1,338	19,154	84,204	2,110	0.8011	0.0653	0.3881	0.1118	0.4999	0.0911
LA (USAD)	106,906	15,255	0.1427	1,992	13,263	90,195	1,456	0.8623	0.1306	0.5777	0.2130	0.5688	0.2271
LA (USAD+WC)	106,906	8,675	0.0811	2,831	5,844	97,614	617	0.9396	0.3263	0.8211	0.4670	0.7175	0.4947
RobustLog (SR)	63,647	1,595	0.0251	202	1,393	61,005	1,047	0.9617	0.1266	0.1617	0.1421	0.5612	0.1237
RobustLog (FT)	74,835	2,536	0.0339	2,414	122	72,299	0	0.9984	0.9519	1.0000	0.9754	0.9875	0.9748
Log Attention (SR)	75,295	2,376	0.0316	2,092	284	72,595	324	0.9919	0.8805	0.8659	0.8731	0.9345	0.8696
Log Attention (FT)	75,295	2,414	0.0321	2,414	0	72,879	2	1.0000	1.0000	0.9992	0.9996	1.0008	0.9996
<i>DN-C dataset</i>													
DeepLog	55,290	2,224	0.0402	221	2,003	52,643	423	0.9561	0.0994	0.4442	0.1541	0.5658	0.1674
LogAnomaly	55,290	2,874	0.0520	394	2,480	52,166	250	0.9506	0.1371	0.6118	0.2240	0.5992	0.2738
LA (USAD)	55,290	1,570	0.0284	413	1,157	53,489	231	0.9749	0.2631	0.6413	0.3731	0.6801	0.4006
LA (USAD+WC)	55,290	1,208	0.0218	507	701	53,945	137	0.9848	0.4197	0.7873	0.5475	0.7699	0.5684
RobustLog (SR)	41,809	15,110	0.3614	265	14,845	26,615	84	0.6429	0.0175	0.7593	0.0343	0.4076	0.0760
RobustLog (FT)	38,703	471	0.0122	451	20	38,232	0	0.9995	0.9575	1.0000	0.9783	0.9899	0.9783
Log Attention (SR)	38,235	361	0.0094	358	3	37,724	150	0.9960	0.9917	0.7047	0.8239	0.9110	0.8343
Log Attention (FT)	38,235	481	0.0126	481	0	37,727	27	0.9993	1.0000	0.9469	0.9727	0.9862	0.9727

LogAttention is an ensemble of WC, USAD and SAD. The performance of the LogAttention USAD module alone is labeled as LA (USAD). The performance of the LogAttention USAD module together with the WC module labeled as LA (USAD + WC). The training dataset is then used to evaluate the performance of the LogAttention SAD module. The dataset for SE-A, SE-C, and DN-C is randomly split into 30% training and 70% test data. After training on the training dataset, this fine-tuned LogAttention model was evaluated, which is an ensemble of WC, USAD, and SAD against the test dataset. The results of this fine-tuned model are labeled as LogAttention (FT). To eliminate the possibility of Overfitting, a separate training dataset is curated by dropping the messages (event templates) from the training dataset that were also found in the test dataset. The LogAttention SAD model was retrained on this similarity removed training dataset and evaluated using the test dataset ensembling all three modules of Log Attention, and report the results of this model labeled as LogAttention (SR). The same split dataset was used to train RobustLog. However, since the process of removing similar logs from the data set is random during reassigning labels to original structured logs, RobustLog's manner of preprocessing structured logs both during training and inference yields marginally different amounts of resulting test logs when reassigned to the original datasets, as compared to LogAttention. Just like in the case of log attention, two separate results of RobustLog are reported, fine-tuned, and similarity-removed with the results labeled as RobustLog (FT) and RobustLog (SR). Figure 5 presents the evaluation in terms of Macro F₁ score. It can be observed that even the unsupervised models of

LogAttention perform better than the semi-supervised benchmark models DeepLog and LogAnomaly for all datasets across all metrics.

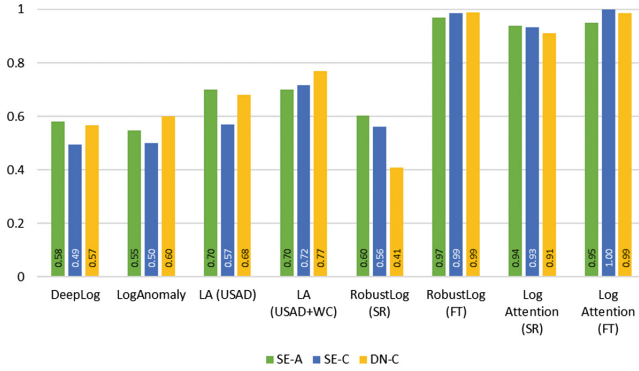


Fig. 5. Model Macro F_1 score across datasets

LogAttention (SR) yields very good results in supervised settings which are almost as good as RobustLog (FT). Without removing the similar templates between the training and test datasets, LogAttention (FT) yields near-perfect Precision, a very high Recall, and high Macro F_1 . With the LogAttention (FT) model, one may suspect overfitting of data. However, log data is inherently repetitive and has repetitive patterns, and LogAttention (FT) seeks to benefit from these repetitive patterns to give accurate insights to the EM after analyzing a very small proportion of the dataset.

The Macro F_1 score and Matthews Correlation Coefficient (MCC) of LogAttention are superior to DeepLog, LogAnomaly, and RobustLog. It is apparent that with training, the performance of LogAttention significantly improves and becomes near perfect whereas even in new environments it has a good headstart over other techniques.

4 Conclusion

LogAttention uses Language Models to generate more representative embeddings for the log messages. Instead of working with just the template of the log message, it proposes a mechanism to ingest the complete message and let the model learn from the embedded reply in the message as well. This is a novel approach and has not been seen in literature before. The approaches used in the literature DeepLog and LogAnomaly rely on efficient log parsing because they expect a log message divided into event templates and log parameters. State-of-the-art log parsers are not as efficient in parsing logs from datasets in the wild. This also affects the efficiency of Anomaly detection. The LogAttention approach circumvents this issue by using the whole content of the log message and provides superior results when compared to other approaches in LAD.

LogAttention is pioneering work in using embeddings from Attention-based Transformer Language Models for LAD to analyze run-time logs from a software release in production to assess the software release quality by accurately identifying anomalous

logs to predict and preempt failure. To give the models a head start, LogAttention also uses heuristic-based weak classifiers enabling the model to achieve a certain level of even without the availability of labeled data. The model also uses heuristics such as the log levels already built in the messages, the execution time of the log event, and an evolving error lexicon to detect any unusual text in the log message itself. These classifiers, although not as efficient individually, allows the system a better start instead of relying solely on un-trained classifiers.

The paper also introduces a novel data curation system for HIL labeling. Learning from the rapidly evolving field of active learning, the paper introduces MCR Sampling (marginal sampling, certainty sampling, and random sampling) to curate the most pertinent tagged data for the model. It curates the most relevant 20% from a log dataset consisting of one to three days of log data by selecting the equal parts, data about which the model is unsure (marginal sampling), data regarding which the model is certain (certainty sampling), and random data to include random samples in the data as well. The results demonstrate this technique to be superior to just randomly sampling data.

References

1. Di Francesco, P., Malavolta, I., Lago, P.: Research on architecting microservices: trends, focus, and potential for industrial adoption. In: ICSA 2017 (2017)
2. Mariani, L., et al.: Localizing faults in cloud systems. In: IEEE ICST 2018 (2018)
3. Wu, L., Tordsson, J., Elmroth, E., Kao, O.: MicroRCA: root cause localization of performance issues in microservices. In: IEEE/IFIP NOMS 2020 (2020)
4. He, P., Zhu, J., He, S., Li, J., Lyu, M.R.: An evaluation study on log parsing and its use in log mining. In: IEEE/IFIP DSN 2016 (2016)
5. Nedelkoski, S., Bogatinovski, J., Acker, A., Cardoso, J., Kao, O.: Self-supervised log parsing. In: Dong, Y., Mladenović, D., Saunders, C. (eds.) ECML PKDD 2020. LNCS (LNAI), vol. 12460, pp. 122–138. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-67667-4_8
6. Cheng, J., Dong, L., Lapata, M.: Long short-term memory-networks for machine-reading
7. Vaswani, A., et al.: Attention is all you need (2017)
8. Klosowski, P.: Deep learning for NLP and language modeling. In: SPA 2018 (2018)
9. Clark, K., Luong, M., Le, Q.V., Manning, C.D.: ELECTRA: Pre-training text encoders as discriminators rather than generators (2020)
10. Sanh, V., Debut, L., Chaumond, J., Wolf, T.: DistilBERT, a distilled version of BERT: Smaller, faster, cheaper, and lighter. In: Co-Located with NeurIPS 2019, 5th edn. (2019)
11. Pennington, J., Socher, R., Manning, C.: GloVe: global vectors for word representation. Proceedings of the 2014 Conference on Empirical Methods in NLP (EMNLP) (2014)
12. Vellido, A., Gibert, K., Angulo, C., Martín Guerrero, J.D. (eds.): WSOM 2019. AISC, vol. 976. Springer, Cham (2020). <https://doi.org/10.1007/978-3-030-19642-4>
13. Liu, F.T., Ting, K.M., Zhou, Z.: Isolation forest. In: Paper presented at the Eighth IEEE International Conference on Data Mining, pp. 413–422 (2008)
14. Kazerouni, A., et al.: Active Learning for Skewed Data Sets (2020)
15. Liu, Y., et al.: RoBERTa: A robustly optimized BERT pretraining approach (2019)
16. Du, M., Li, F., Zheng, G., Srikumar, V.: DeepLog: anomaly detection and diagnosis from System Logs through deep learning. In: CCS 2017 (2017)
17. Meng, W., et al.: LogAnomaly: Unsupervised detection of sequential and quantitative anomalies in unstructured logs (2019)
18. Zhang, X., et al.: Robust log-based anomaly detection on unstable log data (2019)