IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received 19 April 2024, accepted 30 May 2024, date of publication 4 June 2024, date of current version 11 June 2024.

Digital Object Identifier 10.1109/ACCESS.2024.3409425

RESEARCH ARTICLE

Unsupervised Log Sequence Segmentation

WOJCIECH DOBROWOLSKI[1,2], MIKOŁAJ LIBURA[1], MACIEJ NIKODEM[2], AND OLGIERD UNOLD[2]

[1]Nokia, 02-685 Warszawa, Poland
[2]Wrocław University of Science and Technology, 50-370 Wrocław, Poland

Corresponding author: Wojciech Dobrowolski (wojciech.dobrowolski@pwr.edu.pl)

This work was supported by the Polish Ministry of Education and Science, funds were allocated from the "Implementation Doctorate" Program.

ABSTRACT The log sequence is often referred to as a language in automated log analysis. The natural consequence of this is that the log sequence should have a structure consisting of words and sentences. However, the word definitions in the log sequence are not uniform in the literature. The first approach splits line-by-line, and the second retrieves word-like structures from the log sequence. The main challenge in the second approach is the measurement of results. There are approaches for constructing unsupervised metrics; however, we found them to be inconsistent. Other methods rely on manually prepared golden standards; however, a benchmark for golden segmentation is not available for any set of logs. To overcome this problem, we created a benchmark of preprocessed log event IDs gathered from the open-source CloudStack log and commercial Nokia software execution. We created a gold segmentation standard with the help of a human expert, and made it publicly available. We then tested known unsupervised segmentation methods used for log sequence segmentation and adapted the Nested Pitman-Yor Language Model. We found that the results of log segmentation performed by these methods vary significantly between the natural language domain and the log domain. VotingExperts achieved the best F-score, recording 97.3% for CloudStack and 44.1% for Nokia logs. The results are related to the uni-gram entropy of the log sequence, which differs across software platforms.

INDEX TERMS Automated log analysis, language abstraction, unsupervised sequence segmentation, software log segmentation, natural language processing, problem-solving, software reliability.

I. INTRODUCTION

Logs are the primary source of information about software failure. Engineers spend hours analyzing them, retrieving the execution flow, and looking for the root cause. However, log sequences are difficult to read, especially for inexperienced developers. Building a higher level of abstraction is one of the most desired functionalities of log analysis tools [1]. One way of doing this is to extract key logs and build a graph of communication [2]. However, this approach is limited to logs where such communication occurs. A more general system treats the entire sequence as a natural language and retrieves word-like segments. Fig. 1 shows that such segmentation simplifies the analysis by dividing it into blocks that are easier for humans to understand.

A log sequence is often regarded as a natural language in log anomaly detection methods [3], [4], [5], [6]. Language consists of a sequence of log events. The meaning, in terms of what happened during execution, is the relationship between the log lines, not its actual content. Therefore, without losing precision, we can abstract the exact log text with log events that the IDs can identify.

In terms of language, the log event ID is treated as a word [3], [4], [6] or letter [7], [8]. We follow the latter approach, as log events have more in common with letters than with words. Considering these quantities, the size of the log events is much closer to the size of the alphabet than that of a dictionary. For example, the number of characters in natural language is relatively small, from 26 in English

The associate editor coordinating the review of this manuscript and approving it for publication was Zhaojun Steven Li.

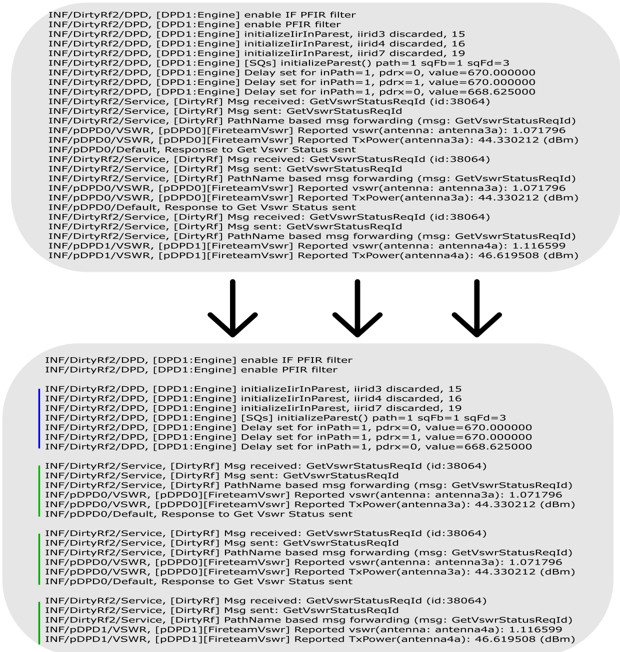


FIGURE 1. Segmentation results in log analysis. Coloured lines indicate repeated log segments for enhanced readability.

to more than tens of thousands of characters in Chinese [9]. At the same time, the English dictionary contains 470,000 words. Typically, tens of thousands of log events occur. Thus, it is more natural to treat a sequence of log events as a sequence of letters from a large alphabet than as a sequence of words from a small dictionary. It is also common to encounter the same sequence of log events in different places in the log file. If we treat log sequence as a sentence, it would mean that we encounter the same sentence in the text many times, in different locations, and there are many such sentences. However, encountering the same word at various locations in a text is normal.

Logs as a result of program execution are a subset of the execution path. This subset is designed to be sufficiently small, not to flood the user with useless information and large enough to provide all the information required to deduce the most important parts. Thus, the logs, reflect the structure of the execution path, revealing the software architecture. Using probabilistic relations between log lines to segment log sequences can help in understanding the execution in the same way that word segmentation helps understand the text.

In the literature, log segmentation can be found under different names, such as trace extraction [10], segmentation of discrete events [8], segmentation of categorical time series [7], log key separation [3], or log event grouping [11]. Segmenting the log sequence in log anomaly detection methods [3], [4], [6], [12] is trivial because it divides the log line by line. Voting Experts [7] and the Nested Pitman-Yor Language Model [13] methods attempt more sophisticated segmentation and divide the log into longer, word-like structures composed of several log lines. Another method is to use the source code to build an execution graph and discover separate traces. A trace is a sequence of logs where it is possible to traverse from the first log statement to the last in the reachability tree created based on the source code and execution [10]. However, this method requires a powerful code parser aware of all possible log printing source code statements, and many executions to build an execution graph. Even when all elements are correctly performed, there are known flaws in this approach [14]. The challenge with such a segmentation is to assess the quality and correctness of the resulting segments. One approach is to define a so-called golden standard [7], also pointed out in our previous field review [15], and to compare the segmentation output with this standard using F-score. Creating a gold standard requires human experts to segment the log. This is time consuming and may lead to arbitrary and suboptimal results [13]. However, with the help of a human expert, segments can be beneficial to humans, which is the primary purpose of segmentation. There are also methods to measure quality without knowing the golden segmentation – quality can be estimated through character-level perplexity [13] or conditional entropy [8].

Contribution of this article includes:

1) created the first golden standard benchmark for log segmentation based on CloudStack logs and Nokia's Component X logs;
2) adapting existing Nested Pitman-Yor language model for log sequence segmentation;
3) using Bayes optimization for hyper-parameter tuning for VotingExperts.

The remainder of this paper is organized as follows. First, we describe the probabilistic algorithms for log sequence segmentation (Section III). We compared two methods, VotingExperts and the Nested Pitman-Yor Language Model (NPYLM), on the benchmark. Next (Section III), we describe a series of experiments on the application of segmentation methods to log sequences from the selected datasets. We analyze the results and discuss their impact (Section V). Finally, we conclude the paper and propose future work (Section VI).

II. MOTIVATING EXAMPLE

Our motivation comes from the real-world problems encountered when dealing with the Nokia component X and CloudStack logs. Unlike logs from HDFS [16] and many other open-source software packages, logs from Nokia and CloudStack are much longer. They can be split by thread ID; however, a single thread can contain thousands of lines. The HDFS log sequence has tens of logs after segmentation by block ID. Thus, finding an anomalous HDFS sequence using any known method [3], [6], [17] is equivalent to locating the erroneous part of the source code. The process of determining the root cause begins immediately. However, the journey only begins with Nokia and CloudStack logs. Segmentation can be crucial for reducing the number of logs for analysis, such as segmentation into log traces to obtain more precise anomalous log sequences [10].

Thousands of logs in a possibly anomalous thread are challenging to read, and require further processing. However, they are well-structured because they reflect the object-oriented implementation of the code. Logs form patterns related to their functionality. For example, if classes A, B, and C realize carrier configuration functionality, then logs from those classes will appear successively with some moderate permutations related to different input parameters. Segmenting these moderately changed patterns will allow humans to see the execution from a bird's-eye view and ease the process of understanding the system's behavior, resulting in a faster fault analysis. Such segments are more easily assigned functional or test labels [18]. We follow the intuition of previous researchers that log patterns will have similar properties to words, and that word segmenting methods can be used successfully.

III. MATERIALS AND METHODS

Let S represent the set of all sequences of discrete events, that is, all sequences from the available log files.

$$\mathcal{S} = \{\hat{e}^0, \ldots, \hat{e}^m\} \tag{1}$$

A single sequence $\hat{e}^j$ from $\mathcal{S}$ contains a sequence of discrete events e_i (log events or letters):

$$\hat{e}^j = < e_0, \ldots, e_n > \tag{2}$$

Each e_i belongs to a finite, known alphabet $\mathcal{A}$, called the closed alphabet. Segment s_k is a sequence of discrete events from $\hat{e}^j$.

$$s_k = < e_{i_k}, \ldots, e_{i_{k+1}} > \tag{3}$$

where $i_k >= 0$ and $i_k < n$, and e_{i_k} is included in segment, while $e_{i_{k+1}}$ is not. A single sequence $\hat{e}^j$ may contain a number of segments s_k^j ($k = 0, 1, 2, \ldots$) such that no two segments share common events e_i. By segmentation w we call sequence of indexes expressed as follows:

$$w = < i_0, \ldots, i_t > \tag{4}$$

where t is the number of segment indexes. Beginning of the sequence and end of the sequence are always incorporated to the segmentation. The lexicon $\mathcal{L}$ is a set of all segments found in sequences $\hat{e}^j$ of set $\mathcal{S}$.

$$\mathcal{L} = \{s_k^j\} \tag{5}$$

where j iterates over all sequences $\hat{e^j}$ and k iterates over all segments within $\hat{e^j}$. Lexicon is equivalent to a dictionary.

There are known segmentation approaches based on the frequency of log events [19], in which the most frequent subsequences are used. However, measuring frequencies alone is vulnerable to moderate sequence changes, similar to word declination. This approach disregards the tight internal correlation between letters in words and leads to overly segmented text, where, for example, each prefix and suffix are separated. VotingExperts and Nested Pitman-Yor Language Model(NPYLM), more advanced probabilistic approaches based on n-grams, are inspired by natural language segmentation and apply unsupervised probabilistic methods to extract words from the sequences. A known approach uses a Control Flow and Reachability Graph to segment logs into traces. However, extracting printing log statements from the source code and building control and reachability graphs is complex and tightly coupled to the given software. However, they cannot be easily ported to different software platforms or programming languages. Such a complex process is usually error-prone. There are also known flaws in source code analysis that are crucial for complete and exact log template extraction [14]. Thus, we concentrate our efforts on probabilistic methods as they are not dependent on the software platform or programming language and require less effort for software companies to implement, maintain and port on many products.

Intuitively, segmentation is challenging for datasets with considerable uncertainty and is accessible to well-structured and predictable sequences. The method to measure this is to calculate the entropy [20]. This study uses the uni-gram entropy (6) and relates it to the segmentation quality. Uni-gram entropy was calculated using the following formula:

$$H = -\sum_x p(x) \log p(x) \tag{6}$$

where $p(x)$ is the probability of character x in given dataset.

A. ASSUMPTIONS

We outline the key assumptions underlying our study. Due to the lack of any benchmark in the field, we aim to start a broader discussion by providing a reasonably good starting point. Although we are not CloudStack experts and do not claim that the segments we created are complete, we believe that our segmentation helps visualize and understand execution, even for someone unfamiliar with the code. Gold segmentations were prepared based on full log lines, while the segmentation methods used event log IDs. These IDs were collected from the Drain algorithm, which is based on carefully designed regular expressions and has its own imprecision. As a consequence, it is possible that some templates are not precise. Preparing and applying regular expressions for Drain are the most time-consuming and error-prone tasks. They were used to extract log events from the log lines. They require considerable testing, careful tuning, and constant maintenance, as the log lines often change owing to the normal software development and maintenance process.

B. DATASETS

The experiments were conducted using three datasets. One text dataset and two log datasets. The information regarding the datasets is presented in Table 1. It contains the name of the dataset, number of sentences (or threads), number of letters, maximum, minimum and average sentence length, and uni-gram entropy of the corpus. The table is ordered in descending order of value uni-gram entropy (Entropy).

TABLE 1. Parameters of the datasets used: Nokia 100 - a collection of 100 unsegmented Nokia log files; Nokia golden - selected Nokia log file; PTB - a well-known text dataset, and open-source CloudStack.

Dataset	Size (MB)	Sentences/ Threads	Letters	Max sentence	Min sentence	Avg sentence	Entropy
Nokia 100	228	47171	1979593	6670	1	41	6.15
Nokia golden	2	44	15833	5096	1	359	4.9
PTB	5.7	49199	4816367	439	2	97	3.04
CloudStack	186	105	391262	25690	21	3726	1.31

The first dataset consists of the CloudStack logs used in [10]. These consist of 1009280 lines from 279553 threads. We removed all short threads consisting of less than 20 lines, as they are not required to be segmented, are often treated as one segment for most segmentation methods, and present no challenge for humans to understand. Because our main focus was to improve our understanding of logs, we decided not to consider them. There were 105 threads longer than 20 lines, resulting in 391262 lines. The longest thread consists of 25690 log lines, and the average thread length was 3726 lines. These threads were segmented into 217297 segments. It is worth mentioning that threads shorter than five constituted more than half of the log content. The logs were preprocessed in a standard manner, as described below. The log format is:

<timestamp> <level> <location> <thread_id> <log_text>

where <level> is logging level, <location> is the name of the module, <thread_id> is the number of the thread, and <log_text> is the text of the printed log.

The second dataset consisted of logs collected from 101 normal and abnormal executions of Nokia software. We split this into Nokia 100 for training purposes, leaving out one file for the gold standard. The thread ID split the log. The total number of log lines in the training set is 1979593 in 47171 threads, and in the golden standard, there are 15833 lines in 44 threads. We treated the thread content as a sentence and the log event ID as a letter. Thus, sentences are significantly longer than the usual natural language sentences. The longest sentence in the gold standard had 5096 letters, with an average of 359. For Nokia 100, the maximal sentence length was 6670, but on average, sentences were 41, shorter than the gold standard. Our main goal was to determine the best algorithm for log segmentation in terms of F-score. We semi-automatically created a gold standard for Nokia logs using the VotingExperts algorithm and expert knowledge. The implementation and datasets are available online [21]. Log files are in the following format:

<component_id> <timestamp> <thread_id> <level> <log_text>

where <component_id> is the name of the component; <thread_id> is the number of the thread, <level> is logging level Info, Debug, Warning or Error, and <log_text> is the text of the printed log. The logs were preprocessed in the following manner (Fig. 2):

- select logs with the above log format,
- remove <component_id> and <timestamp> from every log line to speed up the next Drain step because these columns do not influence the content of retrieved log events, whereas they significantly slow the process,
- retrieve log events with Drain [22],
- separate lines from different threads,
- substitute log lines with corresponding log event ID,
- concatenate all threads from all files into one dataset.

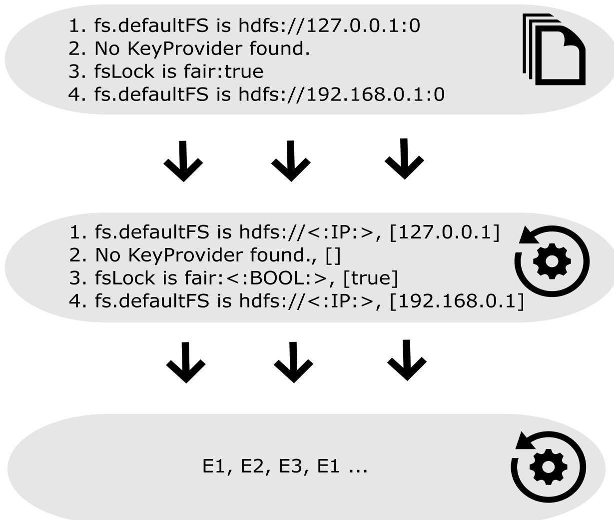
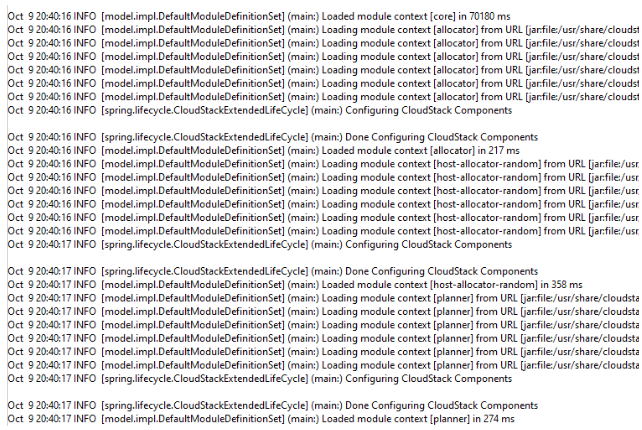


FIGURE 2. Drain log processing workflow. The process begins with log event extraction, followed by conversion of the log sequence into log event ID sequences.



```
Oct  9 20:40:16 INFO  [model.impl.DefaultModuleDefinitionSet] (main:) Loaded module context [core] in 70180 ms
Oct  9 20:40:16 INFO  [model.impl.DefaultModuleDefinitionSet] (main:) Loading module context [allocator] from URL [jar:file:/usr/share/cloudst
Oct  9 20:40:16 INFO  [model.impl.DefaultModuleDefinitionSet] (main:) Loading module context [allocator] from URL [jar:file:/usr/share/cloudst
Oct  9 20:40:16 INFO  [model.impl.DefaultModuleDefinitionSet] (main:) Loading module context [allocator] from URL [jar:file:/usr/share/cloudst
Oct  9 20:40:16 INFO  [model.impl.DefaultModuleDefinitionSet] (main:) Loading module context [allocator] from URL [jar:file:/usr/share/cloudst
Oct  9 20:40:16 INFO  [model.impl.DefaultModuleDefinitionSet] (main:) Loading module context [allocator] from URL [jar:file:/usr/share/cloudst
Oct  9 20:40:16 INFO  [model.impl.DefaultModuleDefinitionSet] (main:) Loading module context [allocator] from URL [jar:file:/usr/share/cloudst
Oct  9 20:40:16 INFO  [spring.lifecycle.CloudStackExtendedLifeCycle] (main:) Configuring CloudStack Components

Oct  9 20:40:16 INFO  [spring.lifecycle.CloudStackExtendedLifeCycle] (main:) Done Configuring CloudStack Components
Oct  9 20:40:16 INFO  [model.impl.DefaultModuleDefinitionSet] (main:) Loaded module context [allocator] in 217 ms
Oct  9 20:40:16 INFO  [model.impl.DefaultModuleDefinitionSet] (main:) Loading module context [host-allocator-random] from URL [jar:file:/usr,
Oct  9 20:40:16 INFO  [model.impl.DefaultModuleDefinitionSet] (main:) Loading module context [host-allocator-random] from URL [jar:file:/usr,
Oct  9 20:40:16 INFO  [model.impl.DefaultModuleDefinitionSet] (main:) Loading module context [host-allocator-random] from URL [jar:file:/usr,
Oct  9 20:40:16 INFO  [model.impl.DefaultModuleDefinitionSet] (main:) Loading module context [host-allocator-random] from URL [jar:file:/usr,
Oct  9 20:40:16 INFO  [model.impl.DefaultModuleDefinitionSet] (main:) Loading module context [host-allocator-random] from URL [jar:file:/usr,
Oct  9 20:40:16 INFO  [model.impl.DefaultModuleDefinitionSet] (main:) Loading module context [host-allocator-random] from URL [jar:file:/usr,
Oct  9 20:40:17 INFO  [spring.lifecycle.CloudStackExtendedLifeCycle] (main:) Configuring CloudStack Components

Oct  9 20:40:17 INFO  [spring.lifecycle.CloudStackExtendedLifeCycle] (main:) Done Configuring CloudStack Components
Oct  9 20:40:17 INFO  [model.impl.DefaultModuleDefinitionSet] (main:) Loaded module context [host-allocator-random] in 358 ms
Oct  9 20:40:17 INFO  [model.impl.DefaultModuleDefinitionSet] (main:) Loading module context [planner] from URL [jar:file:/usr/share/cloudsta
Oct  9 20:40:17 INFO  [model.impl.DefaultModuleDefinitionSet] (main:) Loading module context [planner] from URL [jar:file:/usr/share/cloudsta
Oct  9 20:40:17 INFO  [model.impl.DefaultModuleDefinitionSet] (main:) Loading module context [planner] from URL [jar:file:/usr/share/cloudsta
Oct  9 20:40:17 INFO  [model.impl.DefaultModuleDefinitionSet] (main:) Loading module context [planner] from URL [jar:file:/usr/share/cloudsta
Oct  9 20:40:17 INFO  [model.impl.DefaultModuleDefinitionSet] (main:) Loading module context [planner] from URL [jar:file:/usr/share/cloudsta
Oct  9 20:40:17 INFO  [model.impl.DefaultModuleDefinitionSet] (main:) Loading module context [planner] from URL [jar:file:/usr/share/cloudsta
Oct  9 20:40:17 INFO  [spring.lifecycle.CloudStackExtendedLifeCycle] (main:) Configuring CloudStack Components

Oct  9 20:40:17 INFO  [spring.lifecycle.CloudStackExtendedLifeCycle] (main:) Done Configuring CloudStack Components
Oct  9 20:40:17 INFO  [model.impl.DefaultModuleDefinitionSet] (main:) Loaded module context [planner] in 274 ms
```

FIGURE 3. Failure to segment periodic patterns in CloudStack Logs - no discernible segments were identified, significantly impeding analysis.

FIGURE 4. Incorrect segmentation of CloudStack logs exceeding length limit. Lengthy sequence hinders readability.

FIGURE 5. Incorrect segmentation of exception in CloudStack logs: sequence erroneously split in the middle of exception stack.

The third one is the Penn Treebank (PTB) [23], consisting of 49199 English sentences, 4816367 letters in total, with a maximum sentence length of 439 and 97 on average (Table 1). The text was preprocessed by removing spaces, punctuation marks, new lines, and special characters.

C. GOLD SEGMENTATION PREPARATION FOR LOGS

All examples utilized in this study are sourced from CloudStack, as logs from Nokia are proprietary. For each log dataset, we conducted gold segmentation. Initially, the VotingExperts algorithm was applied with default settings, followed by the manual application of specific rules to refine the automatic segmentation. Refinement was conducted by an expert engineer experienced in log analysis. We have presented a few instances of incorrect segmentations produced by the VotingExperts algorithm in Figures 3, 4, and 5, along with their respective corrections achieved by applying the rules outlined in Figures 6, 7, and 8 To identify incorrect segments we have used the following assumptions:

- **Inconsistent Functionality:** An incorrect segment may contain log lines that do not pertain to the single functionality like starting virtual machine.
- **Misalignment with Keywords:** If a segment lacks coherence or relevance to the keywords, meaning starting and ending some functionality, it could be considered incorrect. Like *Loading* can be considered the beginning of the block and *Loaded* the end (Figure 3).
- **Misinterpretation of Patterns:** Segments should accurately represent visible periods. Incorrect segments might arise if the patterns are misinterpreted or if unrelated patterns are mistakenly included (Figure 4).
- **Mismatched Object Identifier:** Segmentation must accurately capture logs related to the specific instance. If logs from unrelated instances are included in the segment, it could be deemed incorrect.
- **Exceptions Mishandling:** If the exceptions stack is split into many segments, they could be considered incorrect (Figure 5).
- **Length Exceedance:** Segments exceeding the defined length criterion (e.g., more than 60 lines) might contain excessive information, leading to potential confusion and incorrect interpretation (Figure 4).

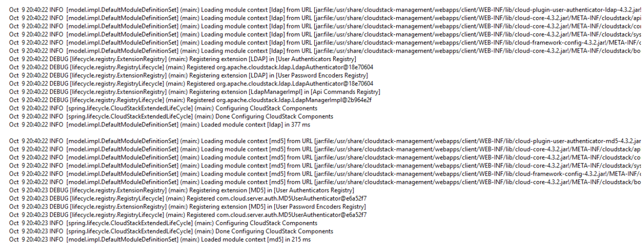


FIGURE 6. Correct segmentation of CloudStack logs based on keywords signifying start and end of functionality: utilizing 'Loading' and 'Loaded' to capture periodic patterns in logs.

FIGURE 7. Correct segmentation of CloudStack logs exceeding length limit: enhanced readability achieved through accurate period identification.

By adhering to these conditions and assumptions, one can identify segments within the log dataset that deviate from the expected criteria, indicating potential inaccuracies or errors. Correct segmentation was achieved through the utilization of the following assumptions:

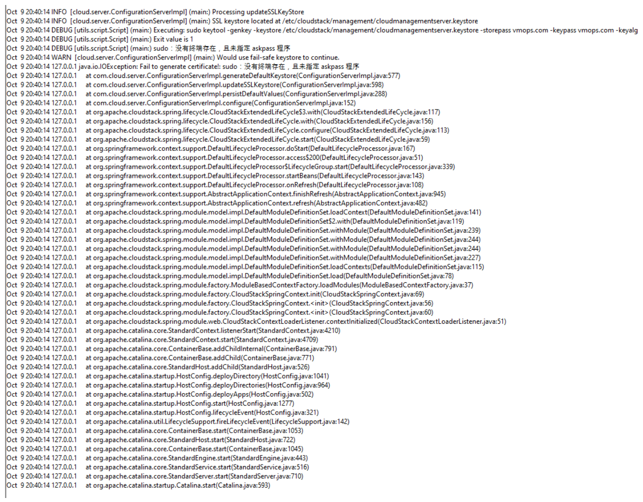


FIGURE 8. Correct segmentation of CloudStack logs for exceptions: ensuring uninterrupted exception stack within a single segment.

- Keyword-based segmentation: Segments were extracted based on specific keywords, such as *injectkeys.sh*.
- Utilization of keywords signifying the beginning and end, such as 'Loading' and 'Loaded' (as illustrated in Figure 6).
- Extraction of periodic patterns to ensure the accurate capture of true periods (as depicted in Figure 7).
- Segmentation based on the identifier of the object instance under processing (as demonstrated in Figure 6).
- Consistent treatment of exceptions within a single segment (as shown in Figure 8).

Our objective was to delineate segments corresponding to distinct functionalities — sufficiently lengthy to facilitate log comprehension yet concise enough to remain manageable for human analysis. Through experimentation, we determined that segments comprising up to 60 lines were most conducive to our analysis.

D. ALGORITHMS

This chapter delves into the theoretical underpinnings of two algorithms: VotingExperts and the Nested Pitman-Yor Language Model. Rather than providing exhaustive details, we focus on familiarizing the reader with the core concepts and theoretical aspects that form the basis of these algorithms.

1) VOTINGEXPERTS

VotingExperts [7] collects information about the boundary entropy and frequency of every n-gram. These measures are stored in a tree, where the root is an empty node, 1-level nodes represent 1-grams, 2-level nodes represent bi-grams, etc., up to n-grams. As a result, the tree has $n+1$ depth. The frequency of a n-gram is straightforward to calculate.

The boundary entropy (BE) is calculated as the entropy of the distribution of tokens that can extend the n-gram [7].

$$BE(n) = -\sum_{x \in X_n} p(x) \log p(x) \tag{7}$$

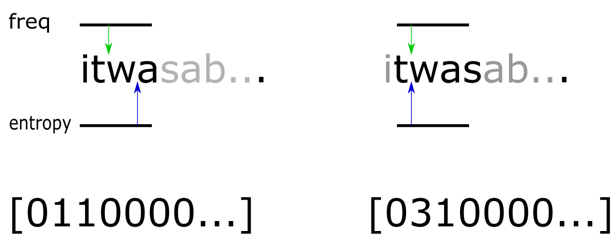


FIGURE 9. Visualization of two iterations of the VotingExperts algorithm on two consecutive sliding windows.

where X_n is the set of child nodes of a fixed node n, thus, the probability $p(x)$ is the conditional probability

$$p(n) = p(e_n | e_0, \ldots, e_{n-1}). \tag{8}$$

With both types of information stored, text is processed sequentially from the beginning with a sliding window. For every window, one vote is determined by a boundary entropy expert and the second is determined by a frequency expert. The first aims to minimize entropy, and the second to maximize frequency. Each expert votes only once for each n-gram (Fig. 9). This approach is fast, considering only $k-1$ voting instead of 2^{k-1}. However, it cannot deal with a situation in which a specific n-gram from the test set is not observed during the training. It also has the disadvantage of considering only one-word boundaries at the time; it leverages only bi-gram word dependencies, a limitation addressed by the NPYLM language model.

2) NESTED PITMAN-YOR LANGUAGE MODEL

This Bayesian approach segments words in an unsupervised manner using the Nested Pitman-Yor Language Model (NPYLM) [13]. This is performed by maximizing the probability of word segmentation w for a given string s.

$$\hat{w} = \text{argmax}_w p(w|s) \tag{9}$$

In terms of discrete events, it is the maximization of $j-th$ segmentation of word w, given a previous event sequence.

$$\hat{w} = \text{argmax}_j p(< i_n^j \ldots i_{n+t}^j > | < e_0 \ldots e_{i_n^j - 1} >) \tag{10}$$

To calculate $p(w|s)$ (9), the NPYLM uses Kneser-Ney smoothing of n-grams, where crucial novelty includes embedding a character n-gram into a word n-gram. The character n-gram allows the model to have no 'unknown word' problem, as in VotingExperts. NPYLM also has another advantage over VotingExperts by leveraging the n-gram word dependencies; in practice, 3-grams are used as 4-grams are too computationally expensive.

E. EVALUATION METRIC

To compare methods, we use the F-score metric calculated as follows:

$$F_{score} = \frac{2 * precission * recall}{precision + recall}. \tag{11}$$

To calculate this metric, we constructed two arrays, each of length equal to the number of non-space characters in the text. The first array corresponds to the original segmentation, and the second to the retrieved segmentation. The arrays are first filled with zeros. Next, the value is changed to one for each position, after which the word is separated into corresponding segments. Then, to calculate the precision and recall, we used the following formulas:

$$precision = \frac{true\_positives}{true\_positives + false\_positives} \tag{12}$$

$$recall = \frac{true\_positives}{true\_positives + false\_negatives}. \tag{13}$$

True_positives are indices from the retrieved array, where the value is one in both arrays. *False_positives* are indices in which the value is set to one in the retrieved array but not in the original array. *False_negatives* are indices from the original array, which are set to one and zero in the retrieved array.

F. HYPER-PARAMETER TUNING FOR VOTINGEXPERTS

The selection of the best hyper-parameter set for a model is not trivial. For example, the most popular technique is grid search [5]. However, a more efficient method for this task is to use a Gaussian Process (GP). The performance can be treated as a sample from the GP. Previous results reduce the range of uncertainty and allow us to choose the next hyper-parameter. It was shown that GP can find better hyper-parameters than a grid search with a fraction of the number of experiments [24].

IV. RESULTS

We analyzed the results generated from the algorithms running on the gold standard as the input data. We checked the segmentation quality first on the CloudStack and Nokia logs and then on the English Text. In English Text, NPYLM achieved significantly better results than VotingExperts. However, for CloudStack and Nokia's golden segmentation, VotingExperts performed better.

A. CLOUDSTACK

First, we evaluated the methods using open-source logs. We assessed the quality compared to the gold standard we created based on VotingExperts' segmentation. The CloudStack gold standard consists of 223030 segments. The threads used for segmentation were longer than 20 lines. An experiment was performed for each segmentation method. The training and testing sets consisted of full logs. VotingExperts achieved the best result with 97% of F-score.

The results are presented in Table 2 and the confusion matrices are shown in Fig. 10. We found that VotingExperts achieved very good results for segmenting CloudStack logs, with a 97% of F-score. NPYLM scores were significantly worse, at 42%, mostly because the recall for this method was very low, 27%. In terms of precision, the NPYLM scores were somewhat higher at 98.9%, whereas for VotingExperts, it was 97,3%.

TABLE 2. Comparative F-score analysis of CloudStack log segmentation using VotingExperts and NPYLM methods.

Algorithm	F-score (%)	Precision (%)	Recall (%)
VotingExperts	**97,3**	97,7	**96,8**
NPYLM	42,5	**98,9**	27,0

TABLE 3. F-score comparison of segmentation algorithms applied to Nokia logs across three experiments using varying training data: golden standard, 100 historical files, and a combination of golden standard and 100 historical files.

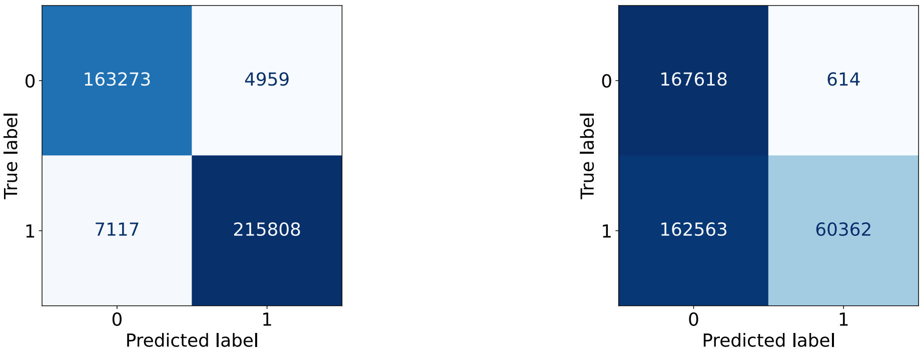
Algorithm	F-score (%)	Precision (%)	Recall (%)
VotingExperts exp1	**44,1**	**40,7**	**48,1**
VotingExperts exp2	n/a	n/a	n/a
VotingExperts exp3	38,1	34,9	42,1
NPYLM exp1	22,5	**16,3**	36,0
NPYLM exp2	22,5	14,6	**45,8**
NPYLM exp3	**23,6**	16,1	44,2

B. NOKIA LOGS

The Nokia logs were used as the primary target. We verified the results of the algorithms compared to the golden segmentation prepared by human experts (see Subsection III-B for more details). Our experiments showed a clear advantage of VotingExperts over NMLP.

Three experiments were performed. The first had the golden standard file as the training and testing set, the second had 100 historical files as training and golden standard for testing, and the third had 100 historical files and golden standard as the training and golden standard as the testing dataset. In the first experiment, VotingExpert achieved the best segmentation with 44,1% of a maximal word length of seven and a threshold of four. The average segment length was 8.3. NPYLM performed the second-best segmentation with 22,5% after 100 epochs and an average segment length of 4.9. The recall for VotingExperts was 48% and for NPYLM, it was 36%. In the second experiment, VotingExpert could not perform any segmentation owing to the lack of testing n-grams in the training dataset. NPYLM achieved a 22% F-score, with 45% recall and an average segment length of 3.1. In the third experiment, VotingExperts performed 38,1% of F-score (window 17, and threshold 4), average 8.1 segment length, and NPYLM 23% and average 3.6 segment length. In this case, the recall was 42,1% for VotingExperts and 44,2% for NPYLM.

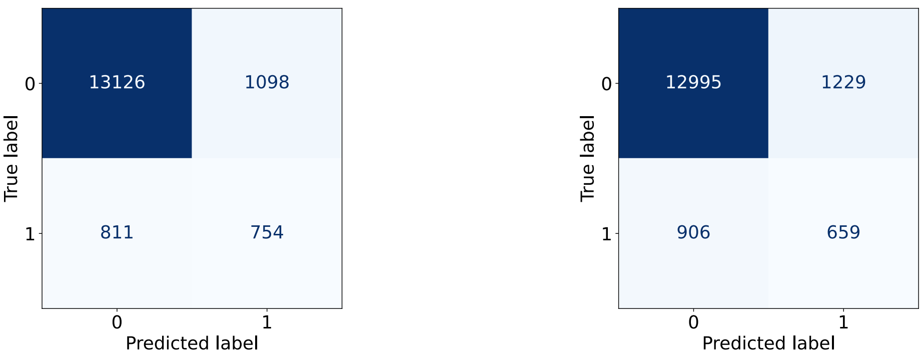
The results are presented in Table 3, the confusion matrices for VotingExperts are shown in Fig. 11, and those for NPYLM are shown in Fig. 12. We found that VotingExperts achieved better results in Nokia log segmentation. Regarding precision, VotingExperts was always better than NPYLM, and for recall, this was also the case, except for Experiment 3, where NPYLM had a 2% advantage over VotingExperts. We can see a significant discrepancy between the precision and recall for NPYLM. For NPYLM, a growing amount of data to learn improved recall but had little impact on precision. For VotingExperts, an increase in the training data reduced the overall F-score. From the average segment length, NPYLM introduced smaller segments than VotingExperts.



(a)

(b)

FIGURE 10. Confusion matrices of word boundaries introduced by VotingExperts (a) and NPYLM (b), of CloudStack logs.



(a)

(b)

FIGURE 11. Confusion matrices of word boundaries introduced by VotingExperts of Nokia logs after training on golden standard (a), and 100 files plus golden standard (b).

TABLE 4. Comparative analysis of results from the original study versus Bayesian optimization hyper-parameter tuning.

Algorithm	parameters	F-score (%)	Precision (%)	Recall (%)
VotingExperts	w:7,t:4	74.9	**85,7**	70,4
VotingExperts improved	w:9,t:3	**80,7**	82,7	**78,7**

TABLE 5. F-Score comparison of PTB segmentation Using NPYLM and VotingExperts.

Algorithm	F-score (%)
NPYLM	**86,3**
VotingExperts	78,3

C. ENGLISH TEXT

Finally, we evaluated two algorithms, VotingExperts and NPYLM, on PTB [23] text.

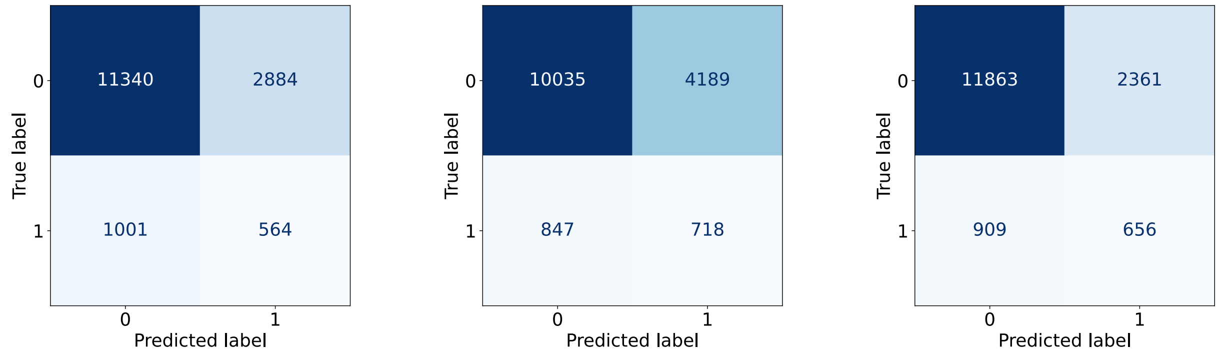
We evaluated the task of identifying word boundaries using an F-score. The segmentation of a single sentence is stored in an array of zeros of a length equal to the number of letters. The values of the indices of letters, after which space was present, were set to one.

NPYLM, with an 86,3% of F-score, achieved the best segmentation. VotingExperts achieved 80,7% of the F-score. It is worth mentioning that with the unsupervised hyper-parameter tuning, we found better hyper-parameters than in the original paper [7] for word length nine and threshold three (Table 4). The results of both methods are presented in Table 5, and the confusion matrix is shown in Fig. 13. The average length of the segment introduced by the NPYLM is 5.1, whereas for VotingExperts, it is 3.7.

V. DISCUSSION

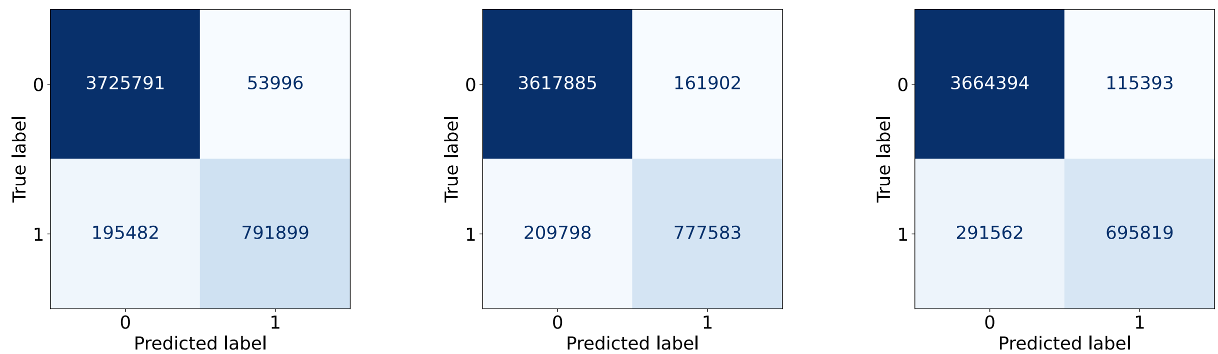
We obtained the gold standard for log sequence segmentation using a semi-automated algorithm. We first used the VotingExperts algorithm with hyper parameters obtained from the original study [7]. Then, using our best knowledge, we corrected the segments that appeared incorrect to us. This is, thus, subjective segmentation. This problem resembles that in Chinese Word Segmentation, in which different segmentation guidelines promote different algorithms [25]. Nevertheless, no benchmark has been published, and such segmentation is required in the field. We hope that this will open the discussion and help develop a standard for log segmentation.

VotingExperts is a computationally cheap algorithm ($k-1$ operations for text of length k). However, it has no smoothing



(a) (b) (c)

FIGURE 12. **Confusion matrices of word boundaries introduced by NPYLM of Nokia logs after training on golden standard (a), 100 files (b), and 100 files plus golden standard (c).**



(a) (b) (c)

FIGURE 13. **Confusion matrices of word boundaries introduced by VotingExperts of PTB English text by NPYLM (a), VotingExperts with hyperparameters (9,3) (b) and (7,4) (c).**

mechanism, making it vulnerable to unseen n-grams. Owing to this limitation, we could not segment the logs of Nokia using VotingExperts trained on a relatively large training set. The NPYLM could not obtain a high F-score on the CloudStack and Nokia logs; however, the larger the training dataset, the better the recall. In the Nokia experiment, with the largest training dataset, the NPYLM recall was better than that of VotingExperts. With low precision on Nokia logs, NPYLM introduced more segments than expected by the expert. This is supported by the average length of the introduced segments, which was 3.8 on average across experiments, whereas for VotingExperts, it was 8. For CloudStack, NPYLM achieved better precision than VotingExperts but had a very low recall. This shows that the segments proposed by NPYLM are very good, but they miss many of the proposed golden segments.

We also observed an interesting and intuitive relationship between entropy and segmentation results. This pattern indicates that lower entropy is associated with better segmentation results. This is intuitive because one can expect sequences with a rich alphabet and considerable uncertainty to be challenging for the segmentation algorithm. CloudStack has an extended part of a sequence with a simple request-response pattern. It is also CloudStack, in which the segmentation result is the highest.

TABLE 6. **F-score, precision, and recall of segmentation performance on CloudStack, Nokia, and PTB datasets.**

Dataset	Algorithm	F-score (%)	Precision (%)	Recall (%)	Entropy
CloudStack	VotingExperts	**97,3**	97,7	**96,8**	2.1
CloudStack	NPYLM	42,5	**98,9**	27,0	
PTB	NPYLM	**86,3**	**93,6**	**80,2**	8.01
PTB	VotingExperts	80,7	82,7	78,7	
Nokia	NPYLM exp3	23,6	16,1	44,2	10.22
Nokia	VotingExperts exp1	**44,1**	**40,7**	**48,1**	

The Drain algorithm is the main choke point for applying the log segmentation algorithm. Its operation depends on regular expressions and thus requires a domain expert effort to carefully fine-tune before use. The quality of log parsing performed by Drain directly impacts subsequent algorithms [26], and is not different in the case of log sequence segmentation. We expect that using the full potential of Deep Learning by learning the embedding of direct log lines will solve this problem and render the algorithm more robust and less dependent on human effort.

VI. CONCLUSION

We compared unsupervised probabilistic segmentation methods for log sequences. We used two methods: VotingExperts, used for text and log segmentation, and NPYLM, a language segmentation method. We adapted the NPYLM to the domain of log segmentation. We found that VotingExperts achieved the best results in terms of F-score in log segmentation (97% and 43%), whereas NPYLM achieved the best results in text segmentation (86%). Increasing the size of the training set was not advantageous for VotingExperts; however, for NPYLM, it significantly improved recall (Table 6). The NPYLM creates many more log segments than those expected by human experts. For a segmentation method to be useful for long log sequences, as in the case of Nokia's logs, the segmentation method should introduce relatively large segments to create a helpful bird's-eye view of the execution path. The solution might be to apply the technique recurrently or allow the user to specify the expected segment length.

The second conclusion is that log segmentation is not a uniform problem but differs significantly between platforms. In the case of CloudStack, a large number of short segments in the gold standard (lengths of 1 and 2) with low entropy allowed segmentation methods to achieve an F-score above 97%. Nokia's gold standard contains longer segments and high entropy. Consequently, segmentation is considerably more challenging, and the results are worse.

Third, unsupervised measures of log segmentation may be affected by log-parsing noise(Section III-A). It is shown that methods based on logs are affected by the quality of log parsing methods. To ensure the usefulness of the log segmentation method, the original logs must be considered. Log-parsing procedures, such as Drain, which is based on regular expression, may introduce noise which misleads the subsequent methods to solve different problems than expected. Therefore, we attempted to create log segments of the original log file, the quality of which was ensured by a human expert. We share gold segmentation in the public repository.

Log segmentation presents a promising avenue for future research in more accurately pinpointing software failures. Conventional detection methods primarily focus on thread-level analysis or a sliding window technique. However, our hypothesis suggests that segment-based methods might outperform those relying on sliding windows. This concept parallels the language processing field, where analyzing unsegmented text through a sliding window is generally less effective in identifying grammatical errors compared to analyses conducted on distinctly segmented text. The success of segment-based methods in log analysis, nevertheless, hinges on the quality of the segmentation process.

Labelling a segment as anomalous instead of a full-thread log sequence would also immediately benefit the user(Section II and Section III-A). The question is how to identify anomalous segments, preferably in an unsupervised manner. Another direction could be to label segments with the names of functionality to which they are related. This provides a better overview of the executed scenario. This requires the embedding of segments in the vector space. The question is whether embeddings would behave similarly to word embeddings while maintaining semantic and syntactic relationships.

REFERENCES

[1] N. Yang, P. Cuijpers, R. Schiffelers, J. Lukkien, and A. Serebrenik, "An interview study of how developers use execution logs in embedded software engineering," in *Proc. IEEE/ACM 43rd Int. Conf. Softw. Eng., Softw. Eng. Pract. (ICSE-SEIP)*, May 2021, pp. 61–70.

[2] I. Beschastnikh, P. Wang, Y. Brun, and M. D. Ernst, "Debugging distributed systems: Challenges and options for validation and debugging," *Commun. ACM*, vol. 59, no. 8, pp. 32–37, Aug. 2016.

[3] M. Du, F. Li, G. Zheng, and V. Srikumar, "DeepLog: Anomaly detection and diagnosis from system logs through deep learning," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.* New York, NY, USA: Association for Computing Machinery, Oct. 2017, pp. 1285–1298, doi: 10.1145/3133956.3134015.

[4] H. Guo, S. Yuan, and X. Wu, "LogBERT: Log anomaly detection via BERT," in *Proc. Int. Joint Conf. Neural Netw. (IJCNN)*, Jul. 2021, pp. 1–8.

[5] L. Yang, J. Chen, Z. Wang, W. Wang, J. Jiang, X. Dong, and W. Zhang, "Semi-supervised log-based anomaly detection via probabilistic label estimation," in *Proc. IEEE/ACM 43rd Int. Conf. Softw. Eng. (ICSE)*, May 2021, pp. 1448–1460.

[6] X. Zhang, Y. Xu, Q. Lin, B. Qiao, H. Zhang, Y. Dang, C. Xie, X. Yang, Q. Cheng, Z. Li, J. Chen, X. He, R. Yao, J.-G. Lou, M. Chintalapati, F. Shen, and D. Zhang, "Robust log-based anomaly detection on unstable log data," in *Proc. 27th ACM Joint Meeting Eur. Softw. Eng. Conf. Symp. Found. Softw. Eng.*, Aug. 2019, pp. 807–817.

[7] P. Cohen, B. Heeringa, and N. M. Adams, "An unsupervised algorithm for segmenting categorical timeseries into episodes," in *Proc. ESF Explor. Workshop*. London, U.K.: Springer, Sep. 2002, pp. 49–62.

[8] G. Shani, C. Meek, and A. Gunawardana, "Hierarchical probabilistic segmentation of discrete events," in *Proc. 9th IEEE Int. Conf. Data Mining*, Dec. 2009, pp. 974–979.

[9] X.-Y. Zhang, F. Yin, Y.-M. Zhang, C.-L. Liu, and Y. Bengio, "Drawing and recognizing Chinese characters with recurrent neural network," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 40, no. 4, pp. 849–862, Apr. 2018.

[10] L. Bao, Q. Li, P. Lu, J. Lu, T. Ruan, and K. Zhang, "Execution anomaly detection in large-scale systems through console log analysis," *J. Syst. Softw.*, vol. 143, pp. 172–186, Sep. 2018.

[11] X. Zhao, Z. Jiang, and J. Ma, "A survey of deep anomaly detection for system logs," in *Proc. Int. Joint Conf. Neural Netw. (IJCNN)*, Jul. 2022, pp. 1–8.

[12] Z. Chen, J. Liu, W. Gu, Y. Su, and M. R. Lyu, "Experience report: Deep learning-based system log analysis for anomaly detection," 2021, *arXiv:2107.05908*.

[13] D. Mochihashi, T. Yamada, and N. Ueda, "Bayesian unsupervised word segmentation with nested Pitman–Yor language modeling," in *Proc. Joint Conf. 47th Annu. Meeting ACL 4th Int. Joint Conf. Natural Lang. Process.*, 2009, pp. 100–108.

[14] W. Xu, *System Problem Detection by Mining Console Logs*. Berkeley, CA, USA: University of California, Berkeley, 2010.

[15] W. Dobrowolski, M. Nikodem, and O. Unold, "Software failure log analysis for engineers—Review," *Electronics*, vol. 12, no. 10, p. 2260, May 2023.

[16] J. Zhu, S. He, P. He, J. Liu, and M. R. Lyu, "Loghub: A large collection of system log datasets for AI-driven log analytics," 2020, *arXiv:2008.06448*.

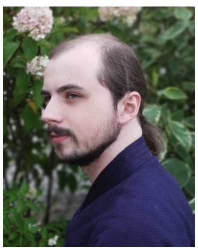
[17] Y. Chen, N. Luktarhan, and D. Lv, "LogLS: Research on system log anomaly detection method based on dual LSTM," *Symmetry*, vol. 14, no. 3, p. 454, Feb. 2022.

[18] W. Dobrowolski, M. Nikodem, M. Zawistowski, and O. Unold, "Improved software reliability through failure diagnosis based on clues from test and production logs," in *Proc. Int. Conf. Dependability Complex Syst.* Springer, 2022, pp. 42–49.

[19] A. Zaidman and S. Demeyer, "Managing trace data volume through a heuristical clustering process based on event execution frequency," in *Proc. 8th Eur. Conf. Softw. Maintenance Reeng.*, Mar. 2004, pp. 329–338.

[20] C. E. Shannon, "A mathematical theory of communication," *Bell Syst. Tech. J.*, vol. 27, no. 3, pp. 379–423, Jul. 1948.
[21] W. Dobrowolski. (2024). *Unsupervised Log Segmentation*. Accessed: Apr. 18, 2024. [Online]. Available: https://github.com/dobrowol/unsupervised_log_segmentation
[22] P. He, J. Zhu, Z. Zheng, and M. R. Lyu, "Drain: An online log parsing approach with fixed depth tree," in *Proc. IEEE Int. Conf. Web Services (ICWS)*, Jun. 2017, pp. 33–40.
[23] T. Mikolov, M. Karafiát, L. Burget, J. Černocký, and S. Khudanpur, "Recurrent neural network based language model," in *Proc. Interspeech*, Sep. 2010, pp. 1045–1048.
[24] J. Snoek, H. Larochelle, and R. P. Adams, "Practical Bayesian optimization of machine learning algorithms," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 25, 2012.
[25] Z. Sun and Z.-H. Deng, "Unsupervised neural word segmentation for Chinese via segmental language modeling," 2018, *arXiv:1810.03167*.
[26] Y. Fu, M. Yan, Z. Xu, X. Xia, X. Zhang, and D. Yang, "An empirical study of the impact of log parsers on the performance of log-based anomaly detection," *Empirical Softw. Eng.*, vol. 28, no. 1, p. 6, Jan. 2023.

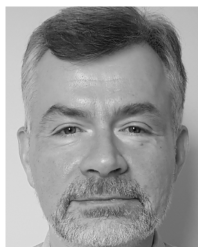
WOJCIECH DOBROWOLSKI received the M.Sc. degree in mathematics from the John Paul II Catholic University of Lublin, in 2008. He is currently pursuing the Ph.D. degree with Wrocław University of Science and Technology. He is currently leading a research project at Nokia. His research interests include artificial intelligence, natural language processing, and their application to software logs analysis.



MIKOŁAJ LIBURA received the B.Sc. degree in applied computer science from Wrocław University of Science and Technology, in 2023, where he is currently pursuing the master's degree in artificial intelligence. He is also employed with Nokia, working on a research project to apply natural language processing pipelines to software logs.

MACIEJ NIKODEM received the M.Sc. and Ph.D. degrees in computer engineering from Wrocław University of Science and Technology, Wrocław, Poland, in 2003 and 2008, respectively.

Since 2008, he has been an Assistant Professor with the Faculty of Information and Communication Technologies, Wrocław University of Science and Technology. His research interests include low-power wireless networks, dependable, reliable and secure communication protocols, and indoor localization.



OLGIERD UNOLD received the M.Sc. degree in automation systems, the M.Sc. degree in information science, and the Ph.D. and D.Sc. degrees in computer science, in 1989, 1991, 1994, and 2011, respectively. He is currently a Full Professor with the Department of Computer Engineering, Wrocław University of Science and Technology. His research interest includes adaptive machine learning methods.