Karlsruhe Institute of Technology

Attention Based Selection of Log Templates for Automatic Log Analysis

Bachelor's Thesis of

Vincenzo Pace

at the Department of Informatics
Institute for Program Structures and Data Organization (IPD)

Reviewer: Prof. Dr.-Ing. Klemens Böhm
Advisor: M.Sc. Pawel Bielski

1st Apr 2023 – 1st August 2023

Karlsruher Institut für Technologie
Fakultät für Informatik
Postfach 6980
76128 Karlsruhe

I declare that I have developed and written the enclosed thesis completely by myself, and have not used sources or means without declaration in the text.
Karlsruhe, 24.7.23

. . . . . .
(Vincenzo Pace)

Abstract

Log analysis serves as a crucial preprocessing step in text log data analysis, including anomaly detection in cloud system monitoring. However, selecting an optimal log parsing algorithm tailored to a specific task remains problematic.
With many algorithms to choose from, each requiring proper parameterization, making an informed decision becomes difficult. Moreover, the selected algorithm is typically applied uniformly across the entire dataset, regardless of the specific data analysis task, often leading to suboptimal results.

In this thesis, we evaluate a novel attention-based method for automating the selection of log parsing algorithms, aiming to improve data analysis outcomes. We build on the success of a recent Master Thesis, which introduced this attention-based method and demonstrated its promising results for a specific log parsing algorithm and dataset. The primary objective of our work is to evaluate the effectiveness of this approach across different algorithms and datasets.

To accomplish this, we conduct extensive experiments to test the applicability and generalizability of the attention-based method in log analysis. By comparing multiple log parsing algorithms, we provide comprehensive evaluation within the context of various datasets. By addressing the limitations of existing log parsing approaches and proposing an attention-based method, we contribute to the advancement of automated log analysis. Furthermore, we provide practical guidelines on employing this method in log analysis projects, offering insights into how to approach log parsing algorithm selection for specific tasks.

Zusammenfassung

Die Log-Analyse dient als entscheidender Vorverarbeitungsschritt in der Text-Log-Datenanalyse, einschließlich der Anomalieerkennung in der Überwachung von Cloud-Systemen. Die Auswahl eines optimalen, auf eine spezifische Aufgabe zugeschnittenen Log-Parsing-Algorithmus bleibt jedoch problematisch.

Bei vielen zur Auswahl stehenden Algorithmen, die jeweils eine korrekte Parametrisierung erfordern, wird eine fundierte Entscheidung schwierig. Darüber hinaus wird der ausgewählte Algorithmus in der Regel einheitlich auf den gesamten Datensatz angewendet, unabhängig von der spezifischen Datenanalyseaufgabe, was oft zu suboptimalen Ergebnissen führt.

In dieser Arbeit bewerten wir eine neuartige, aufmerksamkeitsbasierte Methode zur Automatisierung der Auswahl von Log-Parsing-Algorithmen mit dem Ziel, die Ergebnisse der Datenanalyse zu verbessern. Wir bauen auf dem Erfolg einer kürzlichen Masterarbeit auf, die diese aufmerksamkeitsbasierte Methode eingeführt und ihre vielversprechenden Ergebnisse für einen spezifischen Log-Parsing-Algorithmus und Datensatz demonstriert hat. Das Hauptziel unserer Arbeit besteht darin, die Wirksamkeit dieses Ansatzes bei verschiedenen Algorithmen und Datensätzen zu bewerten.

Um dies zu erreichen, führen wir umfangreiche Experimente durch, um die Anwendbarkeit und Allgemeingültigkeit der aufmerksamkeitsbasierten Methode in der Log-Analyse zu testen. Durch den Vergleich mehrerer Log-Parsing-Algorithmen bieten wir eine umfassende Bewertung im Kontext verschiedener Datensätze. Indem wir die Einschränkungen bestehender Log-Parsing-Ansätze ansprechen und eine aufmerksamkeitsbasierte Methode vorschlagen, tragen wir zur Weiterentwicklung der automatisierten Log-Analyse bei. Darüber hinaus bieten wir praktische Leitlinien für den Einsatz dieser Methode in Log-Analyse-Projekten und geben Einblicke, wie die Auswahl von Log-Parsing-Algorithmen für spezifische Aufgaben angegangen werden kann.

Contents

List of Figures

List of Tables

1 Introduction

The increasing complexity of cloud systems and the vast amount of log data generated by these systems have led to a growing need for automated log analysis methods. Log parsing algorithms serve as a crucial component in this process, enabling the extraction of useful information from raw log data for various tasks, such as anomaly detection[10, 4] and system monitoring [9]. While numerous log parsing algorithms have been developed and extensively evaluated in the literature [13], selecting the most appropriate algorithm and its optimal parameterization for a specific task remains a challenging problem. Moreover, the same algorithm is often applied uniformly across the entire dataset, which may lead to sub optimal results.

Recently, an attention-based method for automating the selection of log parsing algorithms was proposed in a Master Thesis [12], demonstrating promising results with a single dataset and log parsing algorithm. However, it remains unknown whether this attention-based approach can generalize to other algorithms and datasets or if it can be extended to combine the results of multiple log parsers to further improve performance.

In this thesis, we aim to investigate the generalizability and applicability of the attention-based method across different log parsing algorithms and datasets. Our hypothesis is that this approach can be successfully employed as an industry-standard practice for automated log analysis. To test this hypothesis, we extend this approach with two additional log parsing algorithms and datasets.

Our experimental approach is novel and important for several reasons. First, it addresses the limitations of existing log parsing approaches, which often apply a single algorithm uniformly across the dataset, by utilizing an attention-based method to adaptively select the most suitable log parsing algorithm and parameterization for a given task. Second, our approach systematically explores the combination of multiple log parsers and their parameterizations, providing a more comprehensive evaluation of the attention-based method's potential to improve log analysis outcomes. Lastly, by examining the performance of this method across different algorithms and datasets, we contribute to the understanding of its generalizability and applicability in various log analysis scenarios, potentially paving the way for its adoption as an industry-standard practice.

2 Methods

In this section we want to introduce the basic concepts of our work. We start by presenting the current text log analysis process, the attention mechanism that is used to select from multiple log templates and then explain the different log parsing algorithms we have selected and describe the datasets used in this work. We close with describing our experimental setup and evaluation of our methods.

2.1 Fundamentals

2.1.1 Log message

A log message, typically a single line, is the output of the log printing system in the log file. Other works call this log message a "log event", "log entry", "log statement" or "log record". Since there is no clear distinction in the literature between these terms and they are often used synonymous, we will also use them interchangeably but try to stick with "log message" [7]. The log printing system and the shape of the logs is often implemented by individual developers and not unified across an organisation. This is amplified by the usage of third party libraries, where developers regularly have no control over the way that log messages are written. Furthermore, while developers are the ones creating the code for the log messages, administrators are often the ones that have to process and analyse them [7].

Log messages often contain contextual information like time stamps, executing program, program version, user and more. Sometimes this information is just noise, sometimes it is critical for tasks such as anomaly detection.

2.1.2 Log File

A log file is a collection of log messages that is persistently stored on a storage medium, also called "event logs", "execution logs" or just "logs". In most cases, these terms are used interchangeably as well and we prefer to use "logs". Modern systems can produce logs in the order of multiple gigabytes per hour, so processing them in an efficient manner is very important [13]. Sometimes these log messages are accompanied by extensive metadata and organised in a more structured manner, sometimes they are just the output to stdout from a system and need to be manually organised.

2.1.3 Log Parsing

Log parsing is a crucial process in the field of log analysis, where the primary objective is to extract structured information from unstructured or semi-structured log data. Logs, generated by various systems and applications, often contain valuable information about the system's state, performance, and potential issues. However, this information is typically embedded in textual log messages that can be difficult to process and analyze due to their unstructured nature and the sheer volume of data.

The idea of log parsing is to approximate how a given log line is produced in the source code. The log parsing process involves applying algorithms to identify recurring patterns and variable parts within log messages. By recognizing these patterns, log parsing algorithms can extract the essential components of each log message and represent them in a structured format. This structured representation, often in the form of log templates, simplifies the log data and makes it more accessible for further analysis. Traditionally, this log parsing has been done with regular expressions, but nowadays logging code can update on the scale of thousands of log statements per month in a big organisation, so keeping up with regular expressions is impossible [13].

Log parsing typically serves as the first step towards downstream log analysis tasks. Parsing textual log messages into a structured format enables efficient search, filtering, grouping, counting, and sophisticated mining of logs. The parsed logs can be used in numerous application domains, including but not limited to:

1. Anomaly Detection: By extracting structured information from log data, log parsing facilitates the identification of unusual patterns or behaviors that may indicate system faults or security breaches.

2. Root Cause Analysis: The extracted information can be used to trace the origin of system failures or performance issues and facilitate timely resolution.

3. System Performance Monitoring: With structured log data, system administrators and engineers can monitor various performance metrics, identify trends, and optimize system performance.

4. Security Analysis: Log parsing allows security analysts to detect potential threats, intrusions, or vulnerabilities in a system by analyzing structured log data for suspicious activities or patterns.

Through this log parsing step, log data can be treated as categorical input data to a machine learning model rather than unstructured textual data.

2.1.4 Log Templates

A log template is a simplified representation of a log message, containing fixed patterns and placeholders for variable parts. The fixed patterns represent the common structure shared by a group of log messages, while the placeholders capture the variable information

specific to individual messages. For example, a log template might look like this: "User <*> logged in at <*>", where <*> is a placeholder for the variable parts, in this case User ID and time stamp [13].

Different algorithms employ various techniques to create templates, such as clustering, pattern mining or longest common sub sequence. This way, they effectively reduce the complexity of the log data. The number of templates generated depends on the algorithm and its parameters and ranges between finer/more templates (less substitutions) and coarser/less templates (more substitutions).

Using log templates offers several benefits, including improved readability, reduced storage requirements, and faster processing times. However, generating log templates can be challenging due to noisy or inconsistent log messages, determining the optimal granularity of templates, and selecting suitable algorithms and parameters.

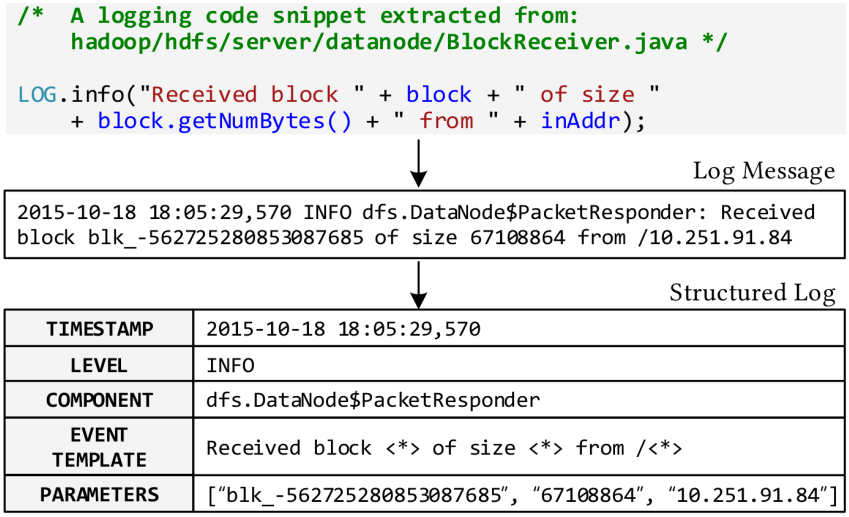


Figure 2.1: An illustrative example of log parsing and log templates [13].

2.1.5 Text Log Analysis Process

In the current industry standard text log analysis process, administrators face the challenge of selecting an appropriate log parsing algorithm and parameterization that best suits their specific log data. This process typically involves the following steps:

1. Research: Administrators begin by investigating various log parsing algorithms available in the literature or through open-source tools. They must consider the strengths and weaknesses of each algorithm, as well as their suitability for the specific log data they will be analyzing.

2. Preliminary Testing: With a selection of candidate algorithms, administrators may conduct a series of limited tests on a small subset of their log data. These tests often

involve experimenting with different parameterizations to gauge the performance of each algorithm.

3. Algorithm Selection: Based on the results of the preliminary tests, administrators choose the algorithm and parameterization that demonstrated the best performance on their specific log data. This decision is typically made using a combination of quantitative metrics, such as accuracy and efficiency, and qualitative factors, such as ease of implementation and maintainability.

4. Fixed Deployment: Once an algorithm and parameterization have been selected, administrators deploy the chosen solution for all future log analysis tasks. This fixed deployment approach assumes that the chosen algorithm and parameterization will continue to perform well on new log data, despite potential changes in log structure or system behavior.

This industry standard process, while practical and widely adopted, has certain limitations. Relying on a fixed algorithm and parameterization may lead to sub optimal results, as it does not account for variations in log data or evolving system requirements. Furthermore, administrators may not have the time or expertise to thoroughly test and compare all available algorithms and parameterizations, potentially leading to the selection of a sub optimal solution.

In this context, the development of an automated method for selecting log parsing algorithms and parameterizations, such as the attention-based approach proposed in this thesis, holds significant potential for improving the efficiency and effectiveness of the text log analysis process. By dynamically adapting to the specific log data and analysis task at hand, such a method can help address the limitations of the current industry standard process and contribute to the advancement of log analysis techniques.

Figure 2.2 shows the current log analysis process. Raw logs are produced by the system in a format that can not be utilized for machine learning tasks. A parsing algorithm takes these kind of log messages and converts them into a log template by identifying fixed (highlighted in blue) vs variable (highlighted in yellow) parts. These templates are then used to train a machine learning model for prediction tasks such as next template prediction.

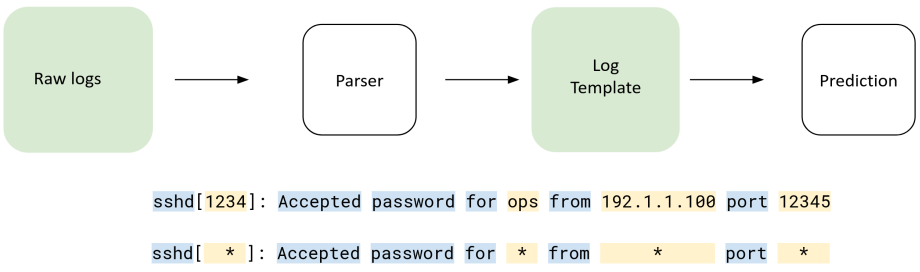


Figure 2.2: The current industry standard text log analysis process

2.1.6 Attention Mechanism and prediction model

The attention mechanism has its origins in natural language processing [3] and has since been successfully applied to various tasks in machine learning and artificial intelligence. Attention mechanisms were first introduced by Bahdanau et al. in the context of neural machine translation, where they were used to improve the alignment between input and output sequences by allowing the model to focus on different parts of the input sequence during the generation of each output element. At its core, the attention mechanism computes a weighted sum of input elements, where the weights represent the relevance or "attention" paid to each element.

This attention mechanism has been further developed for healthcare prediction by Choi et al [5]. They call their improved method "graph based attention", because it leverages a knowledge graph that is combined with the input and fed into the attention mechanism. The knowledge graph (symptom hierarchy) and input (symptoms per visit) are used to calculate attention embeddings, which are trained together with the model on the task of next symptom prediction. Figure 2.4 shows the overall structure of healthcare prediction with graph based attention which we then adapt in 2.5 for our log analysis task.

The knowledge graph in healthcare is based on ICD-9 codes, which model a hierarchy of symptoms and diseases and utilizes domain knowledge crafted by medical experts over several decades. An example can be seen in 2.3.

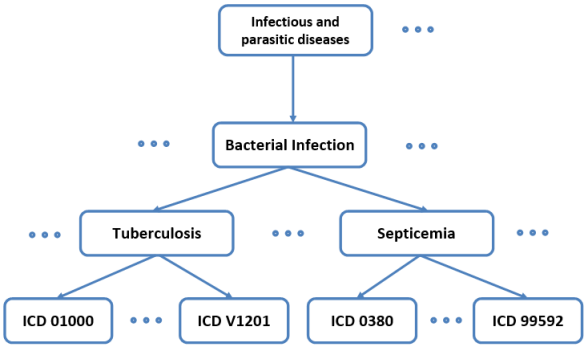


Figure 2.3: Example hierarchical knowledge graph based on ICD-9 codes [11].

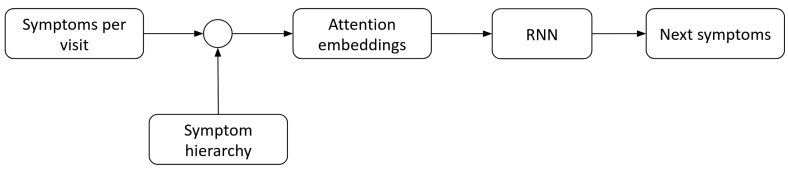


Figure 2.4: The structure of healthcare prediction using gram [5]

.

```
* ""GET / HTTP/1.1"" wally113 rally keystoneauth1/3.11.0 python-requests/2.21.0
CPython/2.7.12""" ,,keystone-apache-public-access,,,openstack.keystone
```

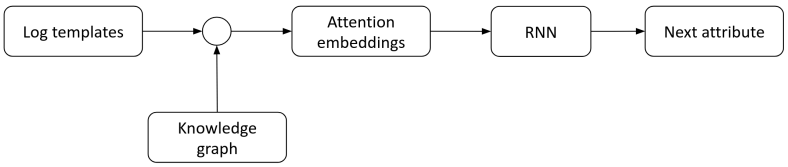


Figure 2.5: Our attention based log template selection. Highlighted in red are the attributes for next attribute prediction.

In healthcare, we calculate attention weights by first computing similarity scores between a leaf in the knowledge graph and all of its ancestors using a softmax function and a feed forward neural network with a single hidden layer. After calculating the individual attention weights, we multiply them with the base embeddings of each of our nodes and then sum the results up to get a final attention embedding for the leaf node. This is done to give more importance to diseases that are more rare in the dataset. Figure 2.6 visualises the computation of the attention weights for covid between its ancestors.

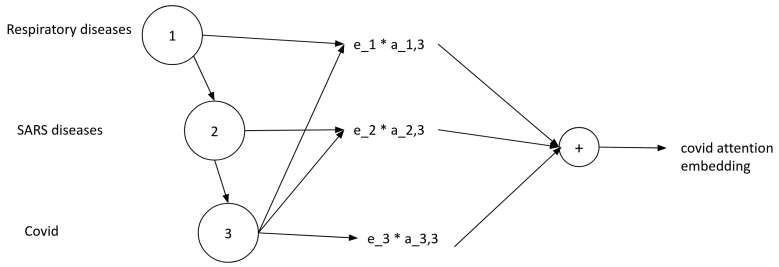


Figure 2.6: Computation of attention weights with gram for healthcare.

For our log analysis task, we utilize softmax and feed forward neural network as well, but instead of calculating the attention weights between leafs and ancestors, we calculate them for each node and its neighbors.

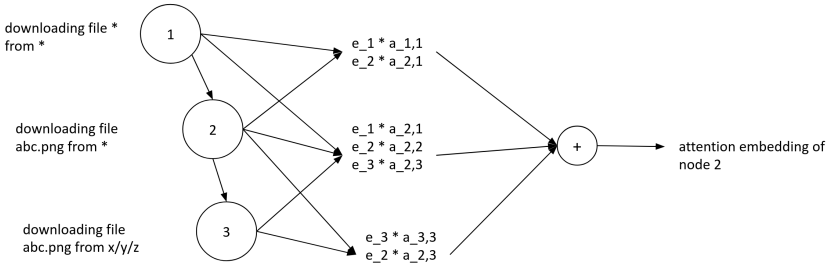


Figure 2.7: Computation of attention weights with gram for log analysis.

The general equation to calculate the attention embeddings is as follows, where v_j are the nodes in the connected features C_i and $a_{i,j}$ are the attention weights between the nodes i and j calculated by softmax and e_j^b are the base embeddings for node j.

$$e_i^a = \sum_{v_j \in C_i \cup \{v_i\}} \alpha_{i,j} e_j^b \in \mathbb{R}^{\dim_e} \tag{2.1}$$

2.2 Log Parsing Algorithms

In the prior Master Thesis [12], the heuristical log parsing algorithm Drain [8] was used. To test whether the promising method of attention based log template selection also applies to other parsing algorithms, we chose two other well performing algorithms that follow different parsing approaches, namely Spell [6] and Nulog [10]. All three of these have been evaluated on the datasets provided by [13] and performed very well. All of them process logs in a streaming manner, meaning that they can parse log messages as they are generated, without waiting for the whole log file to be available. This way, the log parser can provide real-time analysis and monitoring of the system behavior.

The Tools and Benchmarks for Automated Log Parsing paper [13] provides an extensive framework to evaluate the performance of log parsers on 16 different datasets. They also provide their own, uniform, implementation of each log parser. The additional datasets that we have chosen for this work have been taken from their paper.

The parameters that we have chosen for each algorithm stay the same for all datasets, to avoid introducing additional variability. They are most certainly not chosen optimally, but for our work this is ideal since it reflects the use case we try to improve.

2.2.1 Drain

Drain, which stands for depth tree based online log parsing [8], is a widely used log parsing algorithm known for its efficiency and effectiveness in extracting structured information from unstructured or semi-structured log data. Drain employs a heuristic-based approach, which enables it to efficiently group similar log messages and generate log templates that capture the underlying patterns in the data. It consists of multiple steps but starts by preprocessing the log messages using regex rules that are dependent on the dataset and defined by the administrator based on domain knowledge. Afterwards, the log text is split into a list of words. These are then used to build the parse tree. This tree has a fixed size, which is defined by the parameter Depth and avoids constructing a deep and unbalanced tree [8]. The difficulty with drain is to find good parameters for the dataset. Once these are found, it performs better than most other log parsing algorithms in the benchmarks [13].

Key features of the Drain algorithm include:

1. **Incremental processing**: Drain processes log messages incrementally, allowing it to handle large-scale log datasets and adapt to changes in log patterns over time.

2. **Fixed-depth parse tree**: Drain organizes log messages in a fixed-depth parse tree, which significantly reduces the search space and speeds up the clustering process.

3. **Online parsing**: Drain parses logs in a streaming manner, instead of processing batches of log data offline to train a data mining model, enabling it to efficiently group log messages in real-time as they are processed.

4. **Parameterization**: Drain offers two primary parameters, log depth and log similarity threshold, which control the granularity of the generated log templates and can be tuned to optimize the algorithm's performance for specific log analysis tasks.

In our research, Drain is a suitable choice for several reasons:

1. **Scalability**: Drain's incremental processing and efficient clustering approach make it well-suited for handling large-scale log datasets and adapting to changes in log patterns over time, which are common challenges in log analysis.

2. **Flexibility**: Drain's parameterization allows us to explore different levels of granularity in the generated log templates, enabling us to investigate the impact of varying parameter settings on the performance of our attention-based method.

3. **Comparability**: As a widely used and well-established log parsing algorithm, Drain serves as a useful benchmark for comparing the performance of our attention-based method with existing approaches in the literature.

The implementation for drain that we use stems from [1], which unfortunately is not available anymore.

Parameter Setting	Log Depth	Log Similarity Threshold
Fine Drain	10	0.75
Medium Drain	7	0.4
Coarse Drain	4	0.2

Table 2.1: Drain parameter settings

We have taken the coarse and fine settings from the original master thesis [12] and expanded with medium drain settings, which are roughly in the middle of their respective range. Note, that these settings were originally chosen for the Huawei dataset, but will also be chosen for the other datasets. The concrete parameters can be seen in table 2.1.

2.2.2 Spell

Spell, which stands for Streaming Parser for Event Logs using an LCS [6], is an approach that uses the longest-common sub sequence technique (LCS). The algorithm is available in two versions, the original version from 2016 and an improved, parallelized version from 2019. In this work we use the older, less efficient version, because no public implementation of the improved version is available. We took the implementation from [13] and adapted it for our framework [12].

Spell underlies a key assumption: If we view the output by a log printing statement (which is a log entry) as a sequence , in most log printing statements, the constant that represents a log template often takes a majority part of the sequence and the parameter values take only a small portion. If two log entries are produced by the same log printing statement stat , but only differ by having different parameter values, the LCS of the two sequences is very likely to be the constant, implying a message type.

Spell is another popular log parsing algorithm that has been widely used for extracting structured information from unstructured or semi-structured log data. It employs a sequential approach that processes log messages one by one, aligning them with existing log templates or creating new templates as needed.

Key features of the Spell algorithm include:

1. **Independence of domain knowledge**: contrary to drain needing regular expressions, spell aims to avoid domain knowledge all together

2. **Parameterization**: Spell offers a single primary parameter, its threshold, which controls the granularity of the generated log templates and can be tuned to optimize the algorithms's performance for specific log analysis tasks.

3. **Potential for speedup**: with parallelization and the additional parameters split threshold and merge threshold, the performance of spell can be greatly increased.

Similar to drain, spell starts by preprocessing the incoming log messages. This step involves tokenization, where each log message is split into a sequence of tokens based on pre-defined delimiters, such as space and equal sign. For each incoming log message, spell compares the message with existing log templates in the template pool. If the incoming log message matches an existing log template with a similarity above the message type threshold, the log message is aligned with that template. If no matching template is found, a new log template is created for the incoming log message.

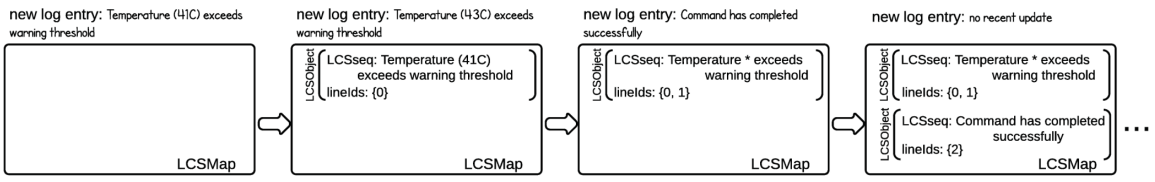


Figure 2.8: Basic Spell workflow [6].

For the threshold parameter we selected values between 0.5 and 0.9, because in our tests 0.5 already produced the coarsest templates and there was no benefit of choosing a lower value. As with drain, these values are chosen identically for all datasets that we run our experiments on.

Parameter Setting	Message Type Threshold
Fine Spell	0.9
Medium Spell	0.7
Coarse Spell	0.5

Table 2.2: Threshold parameters for Spell

2.2.3 NuLog

NuLog is a state-of-the-art log parsing algorithm that leverages neural parsing techniques to extract structured information from unstructured or semi-structured log data [10]. Unlike traditional rule-based or clustering-based approaches, NuLog formulates the log parsing task as a masked language modeling (MLM) problem and employs self-supervised learning to train the model. By leveraging self-supervised learning, NuLog can effectively learn from the log data without requiring explicit supervision in the form of labeled data. This makes the algorithm more scalable and applicable to a wide range of log datasets.

The NuLog algorithm consists of three main steps:

1. **Preprocessing**: Each log message is split into tokens, and a percentage of tokens are replaced by masking tokens. This step prepares the log data for the subsequent masked language modeling task.

2. **Model training**: A transformer-based neural network is trained to predict the masked tokens in the log messages. During training, the model learns to capture the structure and patterns in the log data.

3. **Template extraction**: Based on the predictions made by the neural network model, a vector representation of the log messages is generated. Using this representation, NuLog extracts log templates that capture the underlying structure of the log data.

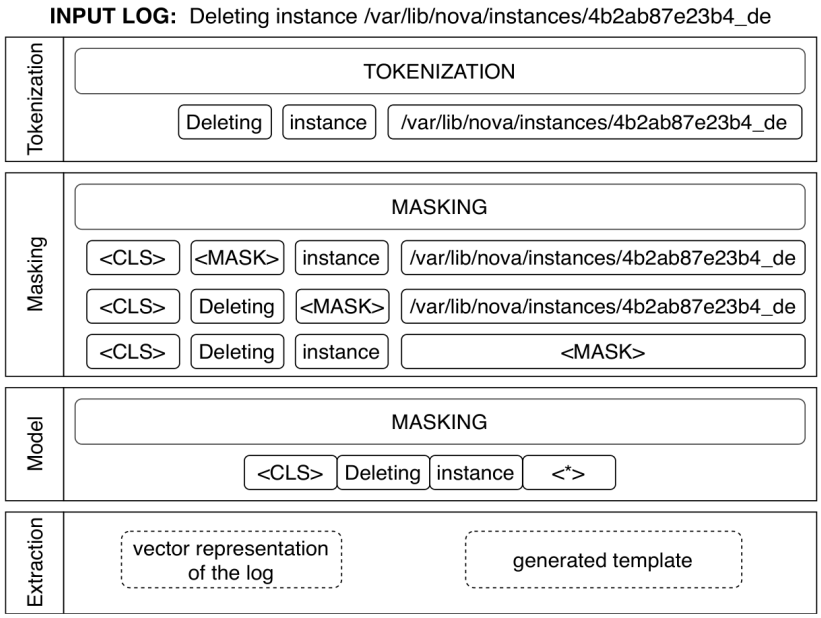


Figure 2.9: Instance of parsing of a single log message with NuLog [10].

One of the main strengths of NuLog is its robustness across various environments and log data types. Due to its neural parsing approach, NuLog can effectively adapt to diverse log patterns, making it a suitable choice for log analysis tasks in different domains and systems. A big downside is its runtime, being orders of magnitude slower than drain and spell.

The following figure shows a benchmark that illustrates the superiority of NuLog in terms of robustness. The figure shows the accuracy distribution of each log parser across the log datasets provided in [13]. We can see that the parsers selected by us are the three top performing ones.

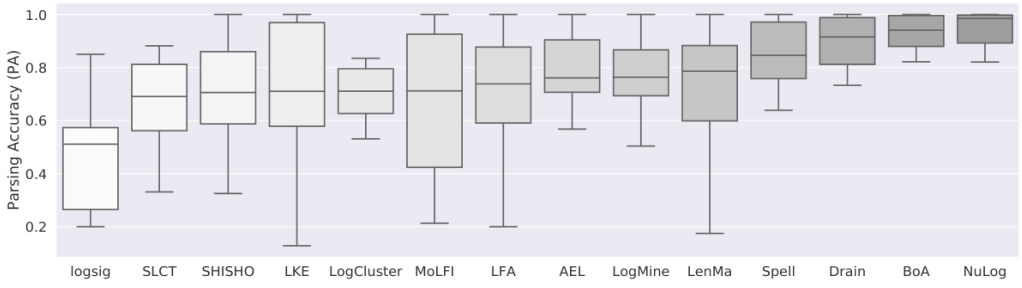


Figure 2.10: Robustness evaluation on the parsing accuracy of the log parsers [10].

NuLog can be controlled by manipulating a parameter k, which the index of each element in the positional vector p. It describes an exponential relationship between each value of

the vector p in the following equation, where j is the positional index of each token:

$$p_{2k} = \sin\left(\frac{j}{10000^{\frac{2k}{v}}}\right), p_{2k+1} = \cos\left(\frac{j}{10000^{\frac{2k+1}{v}}}\right) \tag{2.2}$$

For the selection of this parameter, we have taken inspiration from the parameters that the NuLog authors chose when performing the benchmarks from [13]. We selected the highest value chosen by them as our "coarse" value and the lowest value as our "fine". It was not immediately apparent how the variation of this parameter affects the template generation, so we stuck with the naming scheme for the sake of unity.

Parameter Setting	**Parameter k**
Fine NuLog	5
Medium NuLog	20
Coarse NuLog	50

Table 2.3: Parameter settings for NuLog

2.3 Datasets

In this work, we build upon the dataset utilized in the preceding master's thesis that serves as the foundation for our work. To ensure diversity in our analysis, we have incorporated two additional datasets, which were previously employed in the tools and benchmarks paper [13]. These datasets exhibit distinct characteristics compared to our initial dataset, making them a valuable addition to our research.

The log parsing algorithms have been evaluated on these supplementary datasets, and their respective implementations have been provided by the same authors who contributed the datasets. This ensures a seamless integration of the algorithms with the datasets, enhancing the reliability of our experiments.

It is important to note that while the complete Huawei dataset was used in the original master's thesis, we have opted to limit our analysis to a subset of 2,000 log messages due to the slower performance of other algorithms. This decision aligns our study with the tools and bechmarks and the NuLog paper, both of which also evaluated their algorithms on 2,000 log messages, thereby facilitating a more accurate comparison between our work and these studies.

2.3.1 Huawei Logs

This dataset, originating from the study by [9], was specifically designed to facilitate research in the realm of multi-source, AI-driven analytics. The dataset encompasses logs, metrics, and traces; however, for the purposes of this study, only the logs and the

concurrent setting are utilized. The log messages possess 23 attributes, but only eight are relevant to the prediction task and used to train our model. These include:

1. Hostname
2. log_level
3. programname
4. python_module
5. http_status
6. http_method
7. Payload
8. http_url

Originally, the master thesis also used timestamps, but we found them out to be noise and removed them. More on that in a later section.

Number of log lines	169230
Number of log templates	1446
Number of log attributes	176
Avg. number of attributes per log	4.2
Max. number of attributes per log	5

Table 2.4: Number of log templates and attributes [12]

Log Attribute Type	Number of Attributes	Example Attribute
Hostname	5	wally113
Log Level	3	info
Programname	24	cinder-scheduler
Modulename	95	cinder.scheduler.host_manager
Endpoint	34	auth/login/
HTTP Method	4	post
HTTP Status	11	404

Table 2.5: Log Attributes

2.3.2 Thunderbird Logs

Thunderbird is an open dataset of logs collected from a Thunderbird supercomputer system. The data has been collected for 244 days and encompasses over 200 million log messages and having a total size of 29.60GB. For our experiments we use a sample log containing 2000 log messages, available under [2].

The dataset has five different features, of which we only identify one as relevant for our NAP task: AdminAddr, because the other attributes are only datetime information. The data is also labelled, so it could be used for anomaly detection tasks.

In the collection of datasets utilized for this study, the Thunderbird logs exhibit the lowest degree of diversity, characterized by a high level of repetition because of the number of clusters and an abundance of noisy data, predominantly consisting of timestamps and date-related information. The attribute that we want to predict would be dn228/dn228 in the example log below.

```
- dn228/dn228 crond(pam_unix)[2915]: session closed for user root
```

2.3.3 HDFS Logs

HDFS stands for hadoop distributed file system log. The data has been collected over 38.7 hours and contains over eleven million log messages. The datasize is 1.47GB. As with the Thunderbird logs, we use a random sample of 2000 log messages.

The dataset has six different features, of which we use all except the datetime information for training and three attributes as prediction targets, namely loglevel, component and Pid (if available), as can be taken from the structured log shown in 2.1.

```
 INFO dfs.DataNode$PacketResponder: PacketResponder 1 for block * terminating
```

2.4 Timestamps

During the course of this thesis, an interesting observation emerged regarding the role of time stamps in log messages. Although not the primary focus of this work, it was deemed worthwhile to explore this aspect further due to its potential impact on next sequence prediction.

Time stamps are a common feature in log messages, providing valuable information about the chronological order of events. However, they may also introduce noise, particularly when the data size increases. To investigate this hypothesis, we conducted a series of experiments using the Huawei dataset with and without time stamps.

We started by preprocessing the Huawei dataset to remove time stamps from the log messages. We then created several subsets of the dataset with varying sizes to examine

the effect of data size on the role of time stamps. The experiments were conducted on these subsets, both with and without time stamps, to assess their impact on next sequence prediction performance. The log parsing algorithm that we used for these experiments was drain.

2.5 Next Sequence Prediction

Next sequence prediction is a common task in machine learning, where the goal is to predict the subsequent element or sequence of elements in a given dataset based on the previously observed data. This task is often used as a pseudo task, meaning that it serves as a proxy for solving a more complex or less well-defined problem. In such cases, the performance of a model on the next sequence prediction task can provide insights into the model's ability to solve the underlying problem of interest.

One prominent application of next sequence prediction as a pseudo task is in anomaly detection [10]. Anomaly detection involves identifying patterns or data points that deviate significantly from the expected behavior, which can be indicative of errors, faults, or malicious activities. In this context, next sequence prediction can be used as a proxy task to model the normal behavior of a system. By training a model to predict the next sequence of events in a log, for example, we can capture the typical patterns and relationships within the data. Once the model is trained, it can be used to identify deviations from these patterns, which may signal anomalies.

In our experiments, we use a derivation of next sequence prediction, which we call "next attribute prediction". Instead of predicting the next log template, we predict the attributes associated with. These attributes are listed in 2.3.1 in the case of the Huawei logs.

2.6 DomainML

This work builds upon an earlier work called DomainML, which was developed by a master student to evaluate the impact of domain knowledge on different data science tasks. We use and enhance the existing DomainML codebase by implementing and incorporating our different log parsing algorithms. DomainML used the attention mechanism in combination with various types of domain knowledge, which we leave out to investigate the sole influence of the attention based selection.

2.7 Experimental Setup and Evaluation

In this section, we describe the methodology used to implement the attention-based selection of log templates for automatic log analysis. We build upon the existing framework established in DomainML [12] and modify the log parsing step to accommodate the attention mechanism. Our approach consists of the following steps:

Modification of the Log Parsing Step The original DomainML framework used two log templates. We expanded this to three log templates with varying levels of granularity. We then implemented two additional log parsing algorithms, taking the total number of parsers to three. The new parsers were integrated into the framework, extending the original one.

Creation of Log Templates We executed all three parsers sequentially, allowing each parser to create three log templates. This resulted in a total of nine log templates generated by the three parsers. We also experimented with the different combinations of parsers, namely:

1. Spell + NuLog
2. Spell + Drain
3. Drain + NuLog

We investigated the performance and quality changes between these combinations to understand the impact of the attention mechanism on log parsing.

Training the model We trained the model with a batch size of 128 for 25 epochs, the data was split into 80% train and 20% test, keeping the rest of the setup the same as in the original work[12]. We run our experiments on a machine with 128GB of RAM, two Nvidia RTX 3090 GPUs with Cuda 11.7 and a 48-core CPU.

Evaluation on Multiple Datasets To validate our approach, we repeated the aforementioned steps on two additional datasets: Thunderbird and HDFS. This allowed us to assess the generalizability of our method across different log data sources.

Evaluation Metrics We employed the same evaluation metrics as used in the DomainML thesis to ensure a fair comparison. These metrics allowed us to quantify the benefits of the attention mechanism compared to the baseline approach of not using attention. By not incorporating any domain knowledge, we focused solely on the impact of the attention mechanism on log analysis.

The main evaluation metric that we and the former work look at is top-k categorical accuracy, which quantifies the number of true labels in the top k predicted labels relative to the total number of true labels.

In the following sections, we present the results of our experiments and discuss the performance of the attention-based selection of log templates for automatic log analysis.

3 Results

We successfully showed that the method of attention based selection can be used with other log parsing algorithms and that it improves the prediction quality for some datasets, but not all. Additionally, we show that for small datasets, removing timestamps greatly increased the prediction quality.

3.1 Influence of Timestamps

We evaluated the prediction performance of our ML pipeline on three different dataset sizes. The data shows a strong improvement for small dataset sizes that shrinks with increasing dataset size. The prediction quality without timestamps is always superior compared to the experiments run with timestamps. When using attention based selection, the variance is already much lower compared to fixed parameter templates with timestamps. We named the full dataset 200k logs, but in reality the full log size was only 169k logs. We stuck with it for the naming scheme, but the exact size is not as important as the order of magnitude in comparison to the smaller sets.

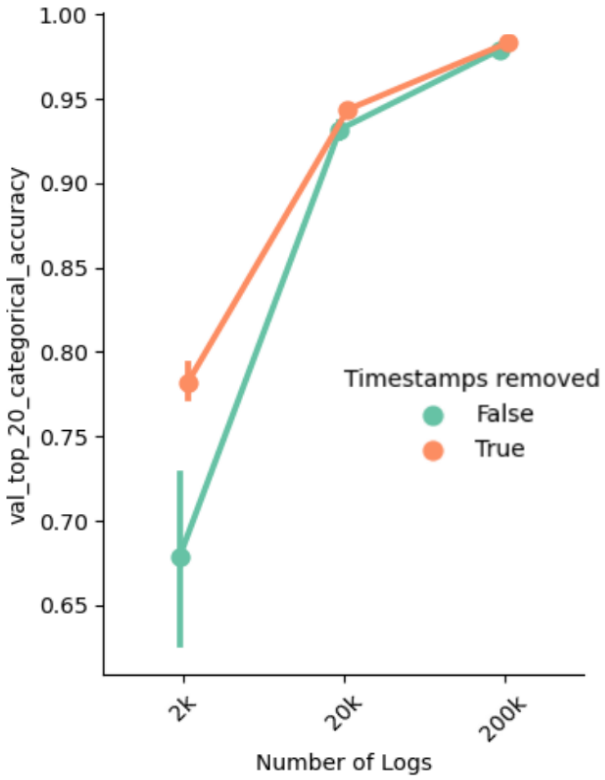


Figure 3.1: Two line plots showing the prediction quality at different dataset sizes, with and without timestamps.

When we look at the particular results for the biggest dataset, we can see that removing timestamps decreases variance of prediction quality across all fixed parameter templates. Finer granularity templates especially benefit from removing timestamps.

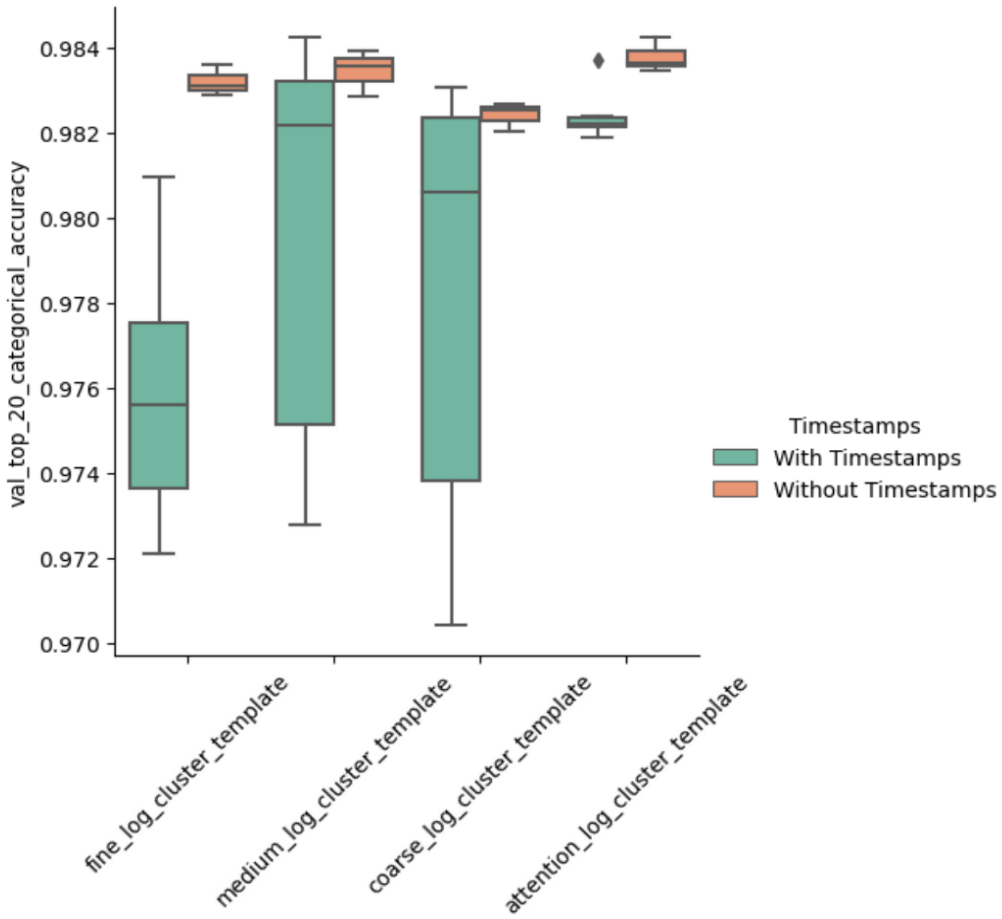


Figure 3.2: The effect of timestamps on different levels of log template granularity and attention based selection. Dataset size 200k logs.

As can be seen in table 3.1, running the experiments with and without timestamps took approximately the same time for all dataset sizes.

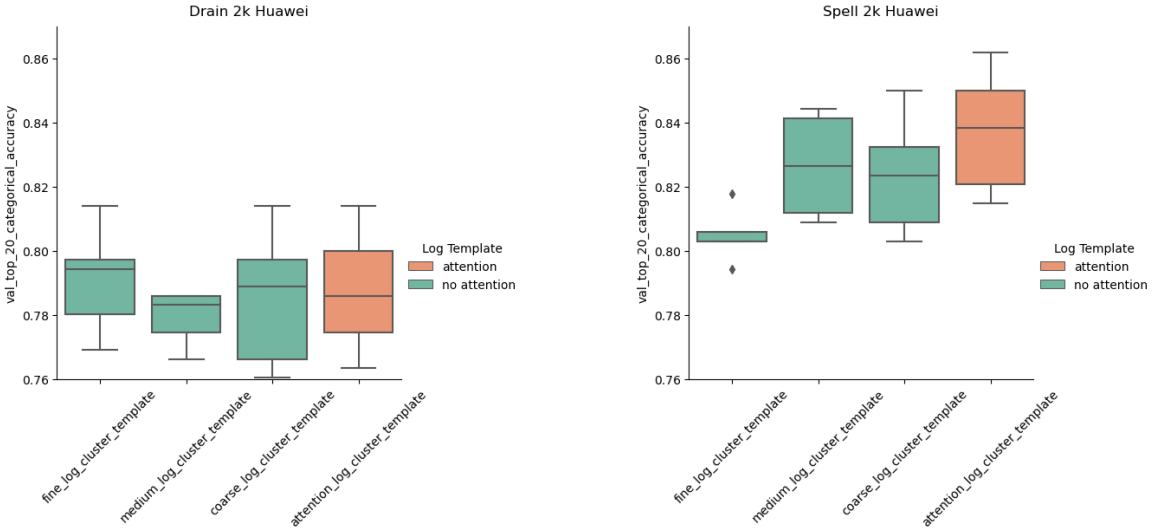
Dataset Size	Runtime (With Timestamps)	Runtime (Without Timestamps)
200k	21.6 min	22 min
20k	3.7 min	3.7 min
2k	1.3 min	1.3 min

Table 3.1: Runtime comparison with and without timestamps for different dataset sizes

3.2 Individual Algorithms

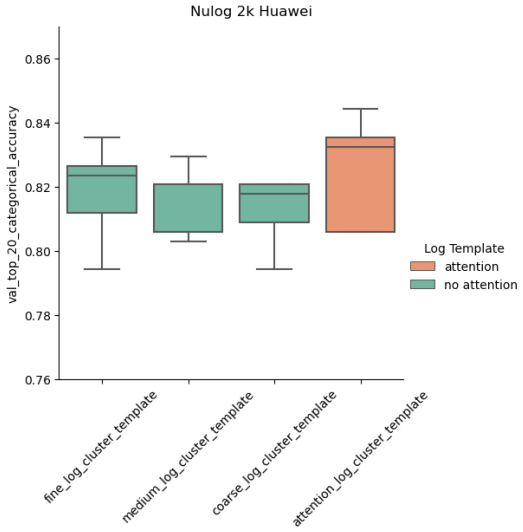
The attention mechanism, illustrated in figure 3.3, delivered very good results, especially for Nulog and Spell, where it lead to the highest median accuracy of 0.83 and 0.84

respectively. Additionally, the data shows that not one type of granularity is best for each algorithm. Drain and Nulog benefited from more "finer" parameterization, while for Spell the "medium" settings lead to the best results, bar attention based selection. Overall, the results between the different log parsing algorithms are very similar at this dataset size.



(a) Drain

(b) Spell



(c) Nulog

Figure 3.3: Three plots comparing attention based selection to fixed parameter template baseline for Drain, Spell and Nulog. Dataset size 2000 logs.

With 2000 logs, the training time for our model is around 1.5 minutes for Spell and Drain and 3.5minutes for Nulog. Attention based selection had no impact on runtime at this dataset size.

As can be seen in 3.2, changing the parameters of the log parsers has great influence on the number of templates. Drain especially produces much more templates than the other two algorithms, which will become more obvious in the Thunderbird dataset.

Parser	Fine	Medium	Coarse	Attention
Drain	198	174	154	526
Spell	172	117	85	374
Nulog	106	123	133	363

Table 3.2: Effect of log parser parameterization on median number of generated log templates. Median values. Attention is the sum of the other three types of templates. Dataset size 2000 logs.

3.3 Combining multiple algorithms

3.3.1 All Algorithms

When combining the templates generated by our log parsing algorithms Drain, Spell and Nulog and feeding it into the attention mechanism, the results are not as clear as with the individual parsers. While the difference in quality is lower, the attention mechanism still selected templates that led to the highest median accuracy of 0.822. We observe that this value is slightly lower compared to using just Spell or Nulog. The runtime when using the templates generated by all log parsing algorithms is very similar compared to using only Nulog.

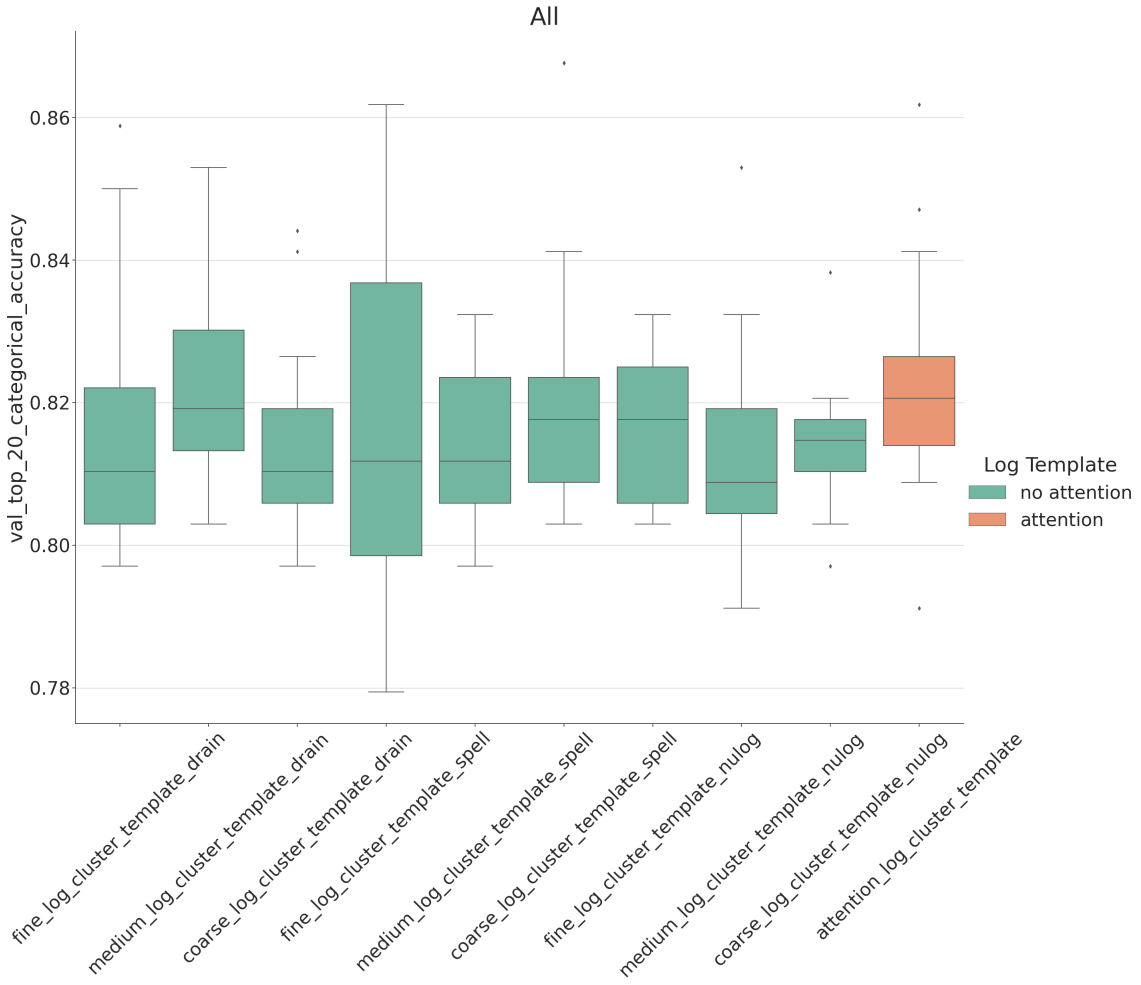


Figure 3.4: Attention based selection from nine log templates generated by three log parsers with three parameterizations. Dataset size 2000 logs.

Parser	Runtime
Drain	1.3 min
Nulog	4.1 min
Spell	1.2 min
All	4.2 min

Table 3.3: Runtime Comparison of Log Parsers

3.3.2 Combination of individual algorithms

We not only ran experiments using all log parsing algorithms at once, but also did tests with all possible combinations of two out three parsers. We can see in every figure, that no combination of log parsers resulted in a definite improvement compared to baseline as observed in the experiments using only a single log parsing algorithm. Interestingly,

combining Drain and Spell led to better results than using all algorithms, as can be seen in figure 3.6. The median accuracy with attention based selection in this experiment was 0.835, compared to 0.822 in figure 3.4.

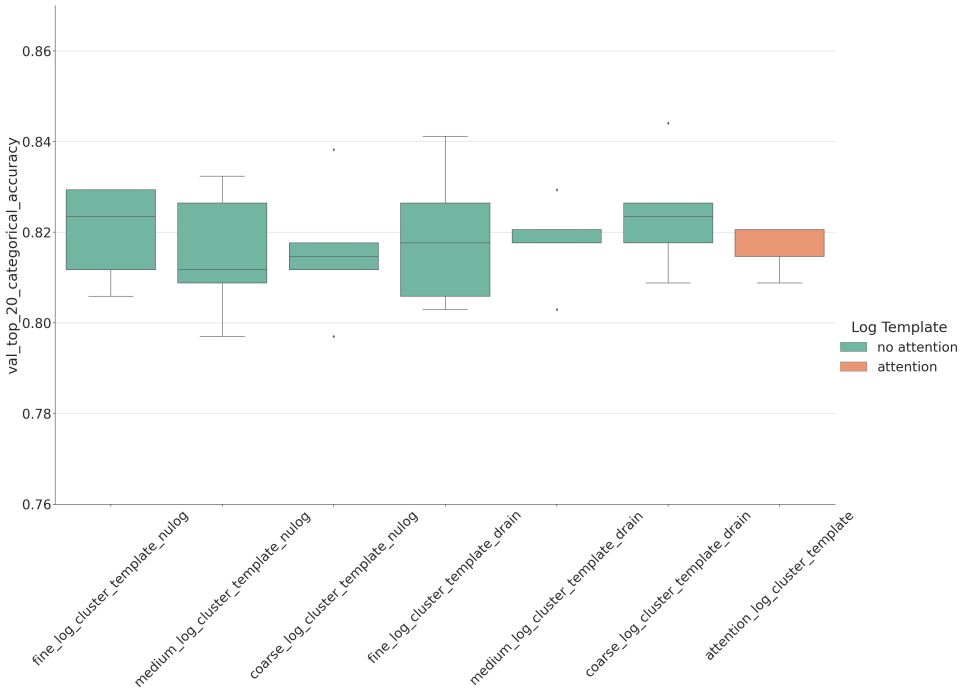


Figure 3.5: Attention based selection from log templates generated Nulog and Drain. Dataset size 2000 logs.

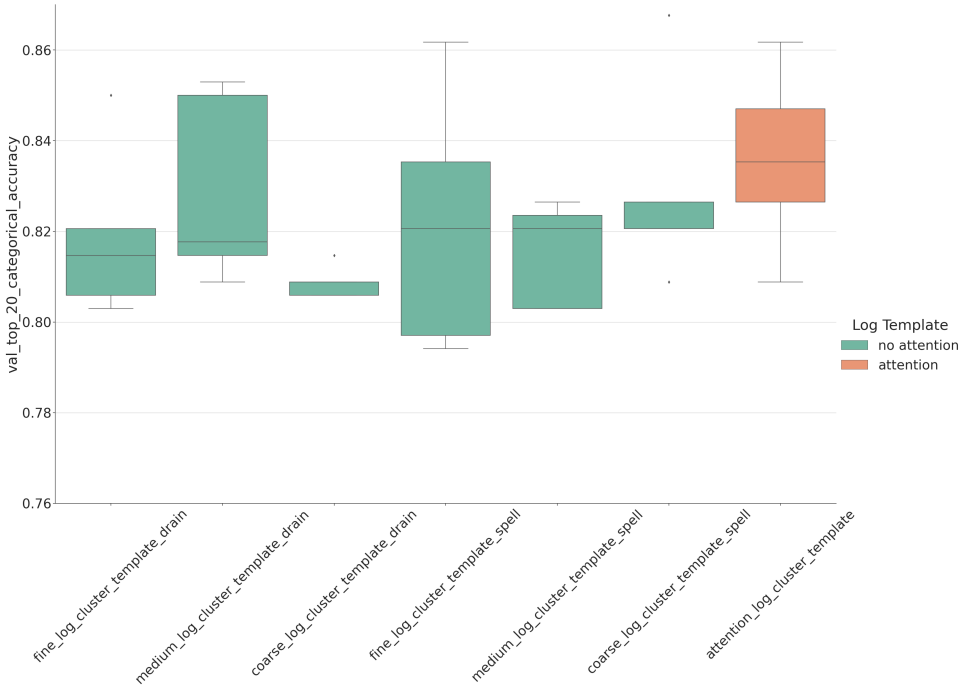


Figure 3.6: Attention based selection from log templates generated by Spell and Drain. Dataset size 2000 logs.

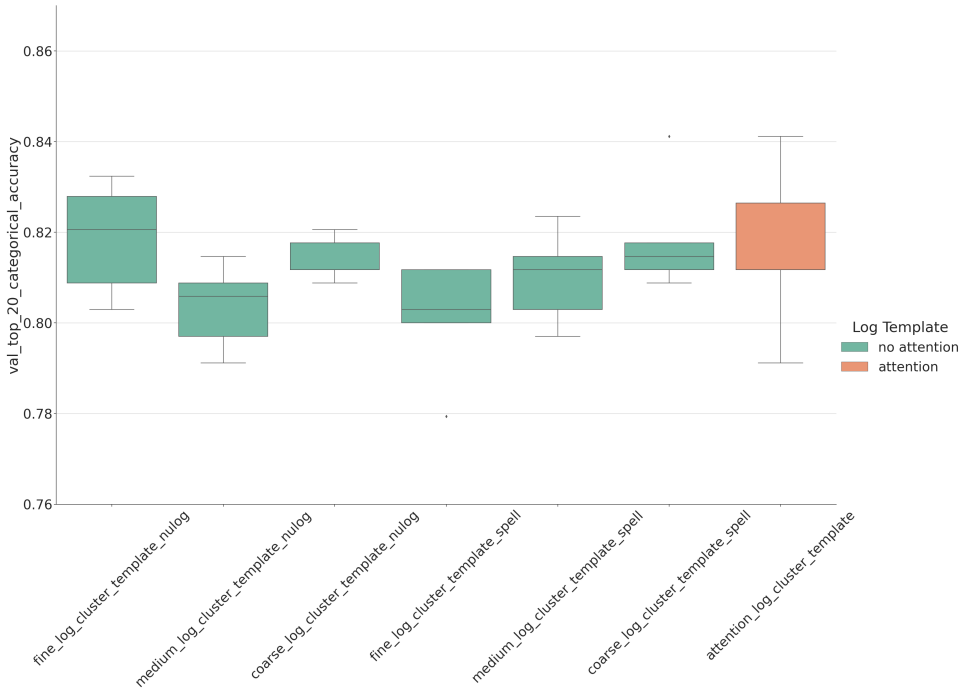


Figure 3.7: Attention based selection from log templates generated by Nulog and Spell. Dataset size 2000 logs.

3.4 Other Datasets

We repeated some of the experiments conducted with the original dataset with two other datasets, in particular, we evaluated the attention based selection of log templates generated by the individual algorithms and by a combination of all of them.

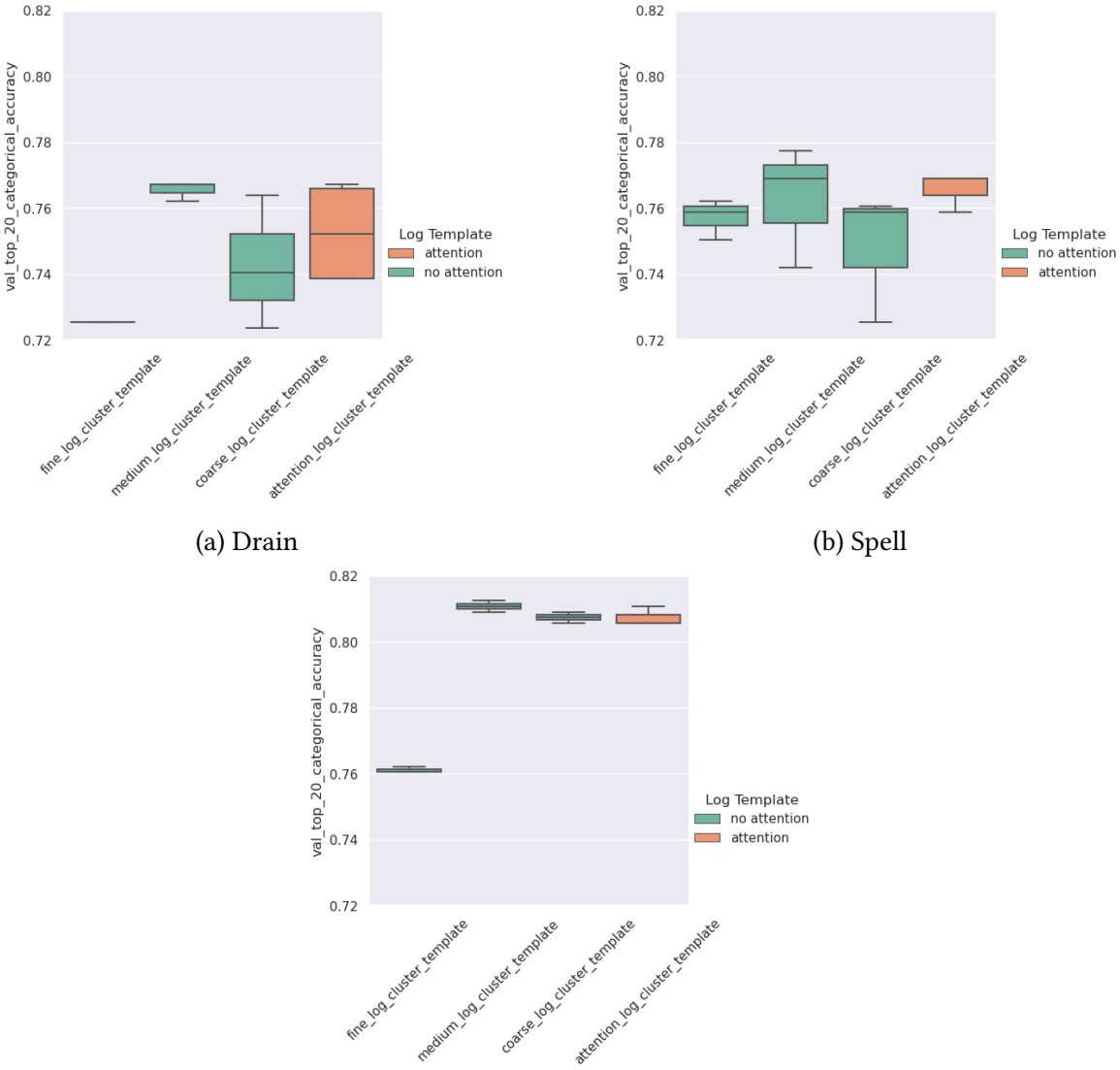
3.4.1 HDFS

On the HDFS dataset, we observed various differences compared to our experiments on the original logs. We can see in table 3.4 that the number of templates generated by Drain is almost ten times higher compared to Nulog and Spell, which were much closer to each other for the original dataset. Furthermore, our model trained for much shorter than the 25 epochs that we set, which was also not the case with our previous experiment. The total runtime stayed very similar.

Parser	No. of Templates	Runtime	Epochs learned
Drain	441	1.8min	9
Spell	57	1.9min	10
Nulog	54	4.6min	16
All	553	4.7min	16

Table 3.4: Number of log templates, epochs learned and runtime for different log parsers. HDFS logs.

Figure 3.8 shows the results for the individual log parsing algorithms, similar to figure 3.3. We can see that the overall prediction quality is slightly worse and that Nulog performs best again. For all algorithms the templates selected by the attention mechanism do not lead to a better result, but in all cases the medium parameters delivered the best prediction quality.



(a) Drain

(b) Spell

(c) Nulog

Figure 3.8: Three plots comparing attention based selection to fixed parameter template baseline for Drain, Spell and Nulog. Dataset size 2000 logs.

Figure 3.9 shows the average prediction quality of all individual log parsers. We can see here that attention selected templates lead to a higher median accuracy, but that this improvement is very small.

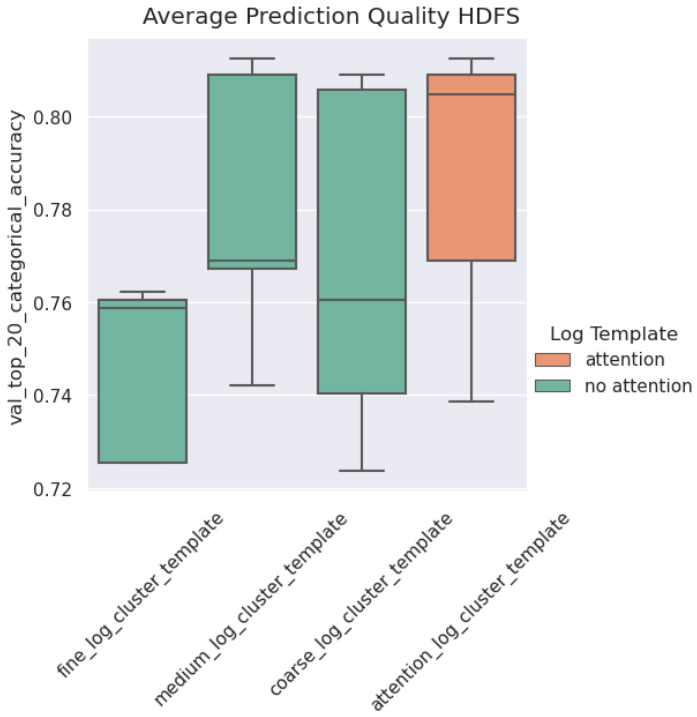


Figure 3.9: Average prediction quality over all log parsing algorithms. Dataset size 2000 logs.

When feeding all the templates from all our parsers into the attention mechanism, we observe that the attention results are very good and on par with the Nulog results, but are not the best overall as can be seen in figure 3.10.

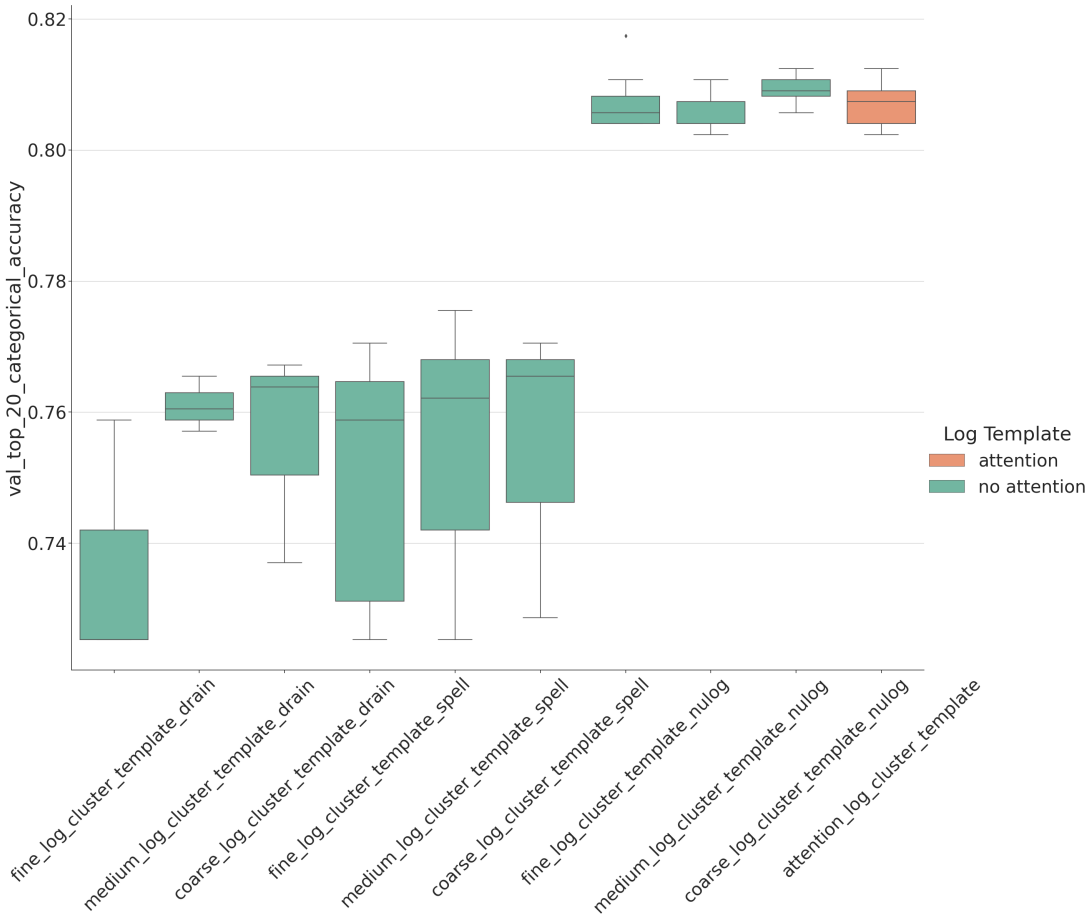


Figure 3.10: Attention based selection from nine log templates generated by three log parsers with three parameterizations. Dataset size 2000 logs.

Motivated by our earlier results that showed that using Drain and Spell performed better than combining all three algorithms [Figure 3.6], we ran another experiment just with Nulog and Spell, because drain performed badly on this dataset. Contrary to the earlier experiment, we saw no improvement.

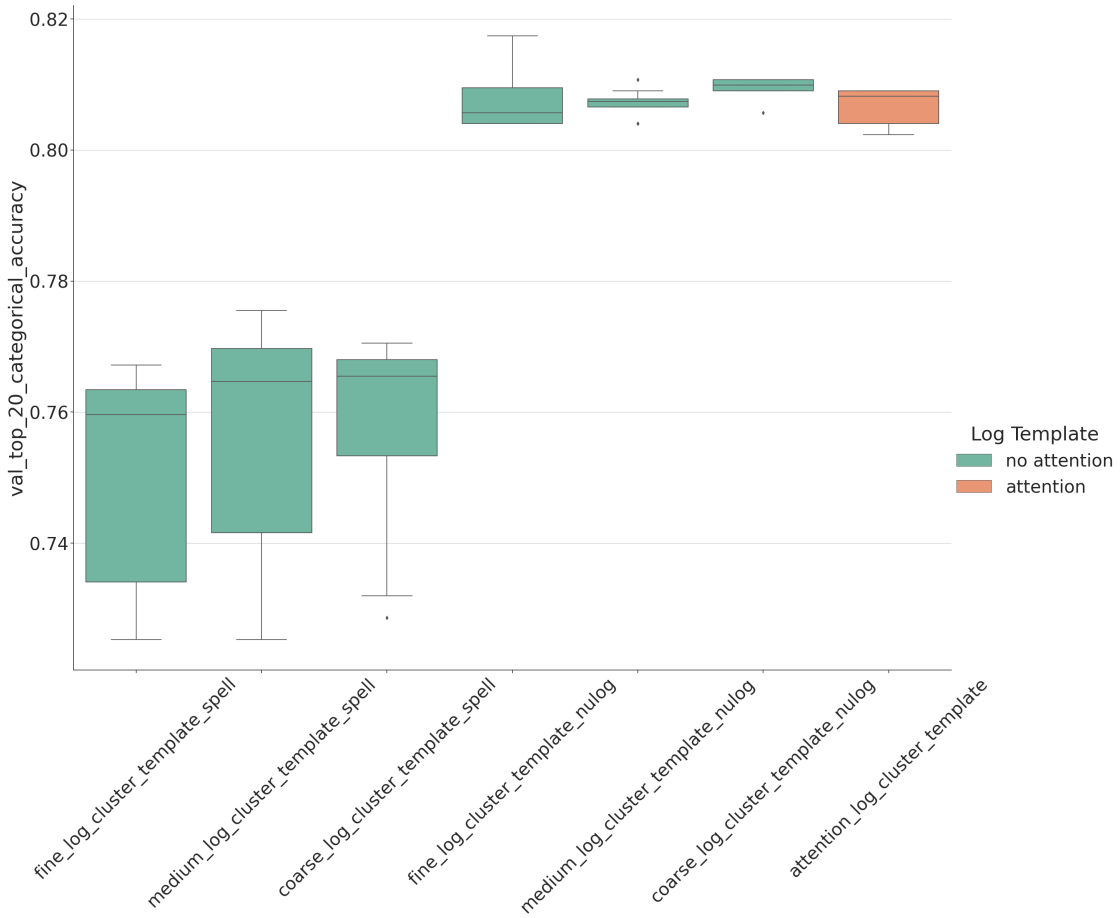


Figure 3.11: Attention based selection from six log templates generated by Spell and Nulog with three parameterizations. Dataset size 2000 logs.

3.4.2 Thunderbird

The results from our Thunderbird experiments differed the most from our original dataset. Drain produced by far the most templates, while the number of templates was generally already much higher than for the other datasets. Using drain templates alone also led to the highest experiment runtime, longer even than when using the combination of all templates. During our Drain and Spell experiments, the model also stopped learning very early and did not improve its validation accuracy beyond the initial value.

Parser	No. of Templates	Runtime	Epochs learned
Drain	2661	12.1min	6
Spell	456	1.2min	7
Nulog	397	4.3min	14
All	3402	4.2min	14

Table 3.5: Number of log templates, epochs learned and runtime for different log parsers. Thunderbird logs.

We can see in figure 3.12 several identical results for the prediction quality with no variance and generally very close, low values.

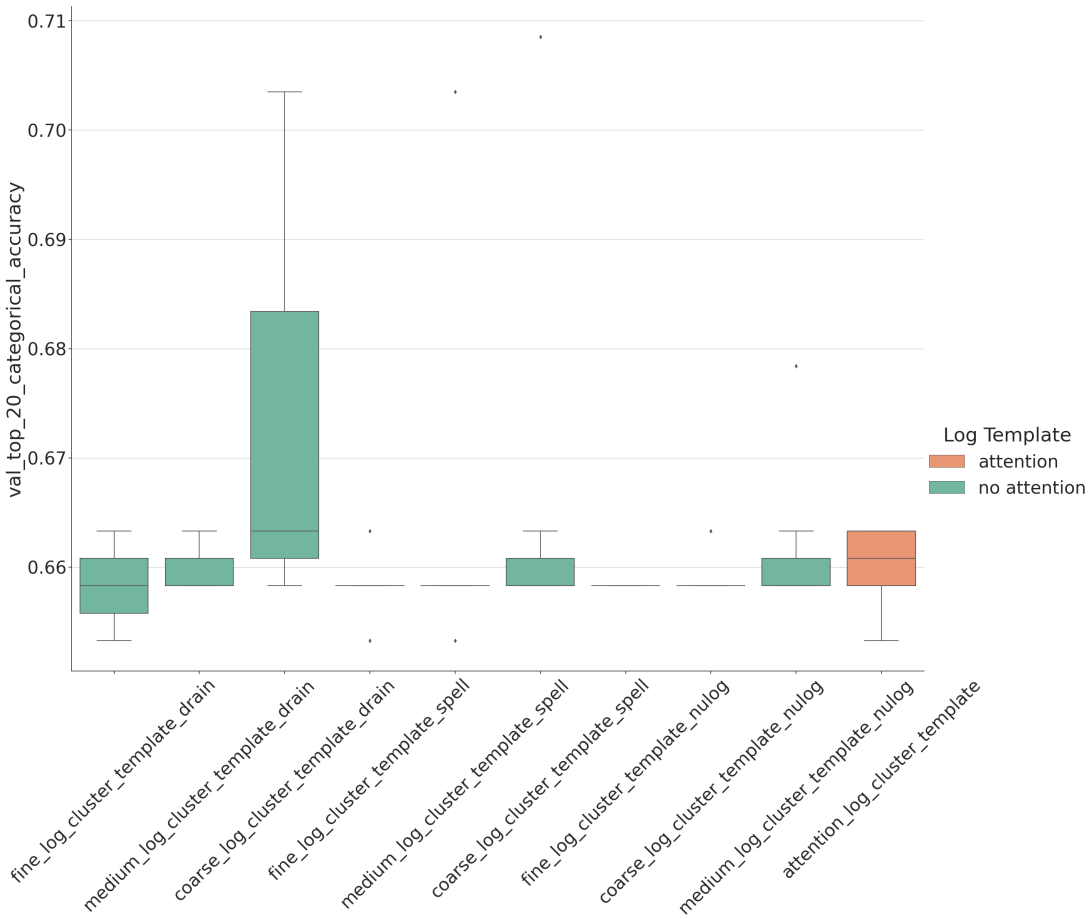


Figure 3.12: Attention based selection from nine log templates generated by three log parsers with three parameterizations. Dataset size 2000 logs.

4 Discussion

In chapter 3 we presented the detailed results for each of our conducted experiments. This section focuses on some high-level observations that we made during the course of this project.

4.1 Generalizability

4.1.1 Other Algorithms

We found that the attention mechanism can be used with other log parsing algorithms as can be seen in figure 3.3. The data also shows that the overall quality of the prediction is dependent on the used parser and that using a more suitable parser for the task leads to better prediction results. In the prior thesis that developed this approach [12], we could see a strong improvement of attention based selection over baseline, but the author only presented the top five categorical accuracy, while we are more interested in the top 20 prediction quality. The author also observed this kind of improvement for a much larger dataset, which we could replicate in our timestamp experiments where we also used the same size dataset as can be seen in figure 3.2. We hypothesise that when using larger amounts of data, we would observe a similar improvement for Nulog and Spell on the original dataset. At our smaller dataset size of 2000 logs, the attention mechanism still performed well and could be used to increase the robustness of the parsing step towards changes in the log structure, especially since there is no runtime penalty for using it.

Our experiments showed that when combining more templates from different parsers and feeding more information into the attention mechanism, the selected templates do not lead to a significantly better prediction quality, as can be seen in figure 3.4. One possible explanation is that there is a sweet spot for the numbers of templates to offer as a selection that might also depend on the dataset size. So offering nine different templates per log at 2000 total logs might be too much. Another reason could be the overall similarity of the different log templates so that the attention mechanism does not learn clearly some most important template for the prediction task.

4.1.2 Other datasets

When performing our experiments on the other two datasets, Thunderbird and HDFS, we could not observe an additional benefit of using attention. For Thunderbird, our baseline results were already much worse than the other datasets'. We try to explain these results

in this section.

One specialty of our original dataset is its large amount of distinct attributes, compared to the other two datasets. While we look at eight different attributes for our next attribute prediction in the Huawei logs, we only use one attribute as prediction target with Thunderbird logs and three with HDFS logs. This lack of structure in the underlying data could explain some part of the bad results. Our original dataset also contained different types of logs: application logs, error logs, system logs, while HDFS and Thunderbird are more homogeneous, each in their own way.

The parameters that were set up for the different log parsing algorithms were tuned on the original dataset and have just been copied for our new datasets. It is possible that they are highly unsuited for this type of logs and that the results would look different with other parameters.

As mentioned throughout this work: the sizes of the dataset could've also played a large role in the results. As our overall quality and also the difference between attention based selection and baseline decreased when we went from the full Huawei log dataset, it is possible that scaling up the experiments to bigger dataset sizes could lead to better results.

4.2 Timestamps

Before we conducted our timestamp experiments, our hypothesis was that timestamps act as noise at large dataset sizes and removing them should improve the prediction accuracy, while at smaller dataset sizes they should be beneficial to learning and contain important information for the prediction. We wanted to find the point where timestamps turn from helpful to hindering. We also predicted that log templates with finer granularity should especially benefit from the removal of timestamps.

Our experiments then showed it is true that fine log templates benefited the most from the removal, as in these templates the time stamps often remain intact or get parsed as below:

```
23 06 11 17:33:25 -> * * 11 17 33 25
```

This is in contrast to the more coarse templates, which might look like this, depending on the parser and specific parameters:

```
23 06 11 17:33:25 -> * * * * * 25
```

Furthermore, we observed that the robustness of our prediction greatly increased, similar to the effect of using attention based selection. One possible explanation could be, that the attention mechanism assigned very low importance to the timestamp part of the log templates and that our removal of the timestamps has a similar effect to this ranking.

On the other hand, it was at the dataset size of 2000 logs that the removal of timestamps had the most positive and significant impact on prediction quality. We believe that the

reason for this is that at a lower dataset size, the relative share of timestamps compared to the rest of the log messages is higher, while also not containing helpful information for the prediction part and acting as the kind of noise we hypothesised for large datasets.

We only conducted our timestamp experiments on a single dataset and with one algorithm, but we believe that these results would translate to other datasets and algorithms as well and that the removal of timestamps should be incorporated into the log analysis workflow.

5 Conclusion

At the beginning of the project we assumed that when combining multiple log parsing algorithms and feeding the generated templates into an attention mechanism, this would lead to improved results compared to only using a single algorithm.

We started off by reproducing the original results and then extending the existing codebase to handle multiple different log parsing algorithms and datasets. The original results were improved even more by using other log parsing algorithms that performed better. Parallel to this, we came up with our timestamp hypothesis and conducted the corresponding experiments. We showed that the removal of timestamps has a beneficiary effect on prediction quality across all dataset sizes, but smaller datasets benefited the most. Thus, we recommend to incorporate timestamp removal as a preprocessing step in the log analysis workflow.

After evaluating different log parser combinations and datasets, we can not conclude that this attention based selection method is effective at improving prediction quality.

5.1 Further research

We conducted the majority of our experiments only on rather small datasets. Future research could repeat these experiments on much larger and more realistically sized datasets and check whether the originally observed positive effects would materialize again. Since we had good results with one dataset, it would be interesting to research what made the experiments successful on this dataset and whether it has some specific properties that made this method work, if the reason for our negative results was not the dataset size.

Additionally, we recommend further experiments to solidify our timestamp results and establish the removal of timestamps as a standard preprocessing step.

6 Acknowledgements

I would like to thank Prof. Klemens Böhm for giving me the opportunity to work on this interesting topic.

Special thanks have to go out to my supervisor Pawel Bielski. He taught me a lot about the research process, invested a lot of time in me and this project and pushed me to give my best. I think him for all the meetings, the valuable feedback on my presentations, his patience and tenacity.

I also want to use this chance to thank my friends Jan, Max, Ismar and Yoki for their open ears and their support during all the years of studying at KIT. Without you guys the time at university would not have been half as fun.

Bibliography

[1] https://appsrv.cse.cuhk.edu.hk/~pjhe/Drain.py.

[2] https://github.com/logpai/loghub/blob/master/Thunderbird/Thunderbird_2k.log.

[3] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. *Neural Machine Translation by Jointly Learning to Align and Translate*. 2016. arXiv: 1409.0473 [cs.CL].

[4] Jasmin Bogatinovski and Sasho Nedelkoski. *Multi-Source Anomaly Detection in Distributed IT Systems*. 2021. arXiv: 2101.04977 [cs.LG].

[5] Edward Choi et al. "GRAM: graph-based attention model for healthcare representation learning". In: *Proceedings of the 23rd ACM SIGKDD international conference on knowledge discovery and data mining*. 2017, pp. 787–795.

[6] Min Du and Feifei Li. "Spell: Streaming Parsing of System Event Logs". In: *2016 IEEE 16th International Conference on Data Mining (ICDM)*. 2016, pp. 859–864. DOI: 10.1109/ICDM.2016.0103.

[7] Sina Gholamian and Paul A. S. Ward. *A Comprehensive Survey of Logging in Software: From Logging Statements Automation to Log Mining and Analysis*. 2022. arXiv: 2110.12489 [cs.SE].

[8] Pinjia He et al. "Drain: An Online Log Parsing Approach with Fixed Depth Tree". In: *2017 IEEE International Conference on Web Services (ICWS)*. 2017, pp. 33–40. DOI: 10.1109/ICWS.2017.13.

[9] Sasho Nedelkoski et al. "Multi-source Distributed System Data for AI-Powered Analytics". In: *Service-Oriented and Cloud Computing*. Ed. by Antonio Brogi, Wolf Zimmermann, and Kyriakos Kritikos. Cham: Springer International Publishing, 2020, pp. 161–176. ISBN: 978-3-030-44769-4.

[10] Sasho Nedelkoski et al. *Self-Supervised Log Parsing*. 2020. arXiv: 2003.07905 [cs.LG].

[11] Milan Vukicevic et al. "A Data and Knowledge Driven Randomization Technique for Privacy-Preserving Data Enrichment in Hospital Readmission Prediction." In: May 2016.

[12] Lena Witterauf. *DomainML - A Modular Framework for Domain Knowledge Guided Machine Learning*. 2021.

[13] Jieming Zhu et al. *Tools and Benchmarks for Automated Log Parsing*. 2019. arXiv: 1811.03509 [cs.SE].