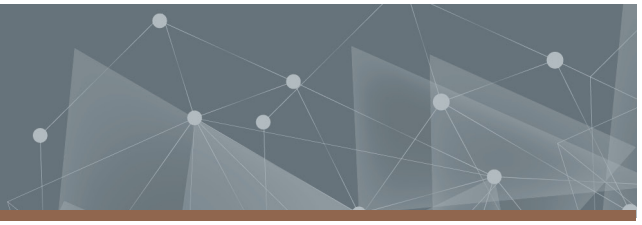
CHALMERS
UNIVERSITY OF TECHNOLOGY



System Log File Anomaly Detection with Sparse Transformer Models

Master's thesis in Engineering Mathematics and Computational Science
Master's thesis in Physics

JOEL HARF ABILI
MARCO CUSKIC

MATHEMATICAL SCIENCES

CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2022
www.chalmers.se

Master's thesis 2022

System Log File Anomaly Detection with Sparse Transformer Models

JOEL HARF ABILI
MARCO CUSKIC

MATHEMATICAL SCIENCES
Chalmers University of Technology
Gothenburg, Sweden 2022

System Log File Anomaly Detection with Sparse Transformer Models
JOEL HARF ABILI
MARCO CUSKIC

Supervisor: Jesper Derehag, Ericsson
Supervisor: Åke Johansson, Ericsson
Examiner: Larisa Beilina, Mathematical Sciences

Master's Thesis 2022
Mathematical Sciences
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

System Log File Anomaly Detection with Sparse Transformer Models
JOEL HARF ABILI
MARCO CUSKIC
Mathematical Sciences
Chalmers University of Technology

Abstract

Log anomaly detection is a useful tool for analyzing system log files and is based on identifying anomalous log messages in such files. Recent years have seen a surge in the use of automated, machine learning/artificial intelligence-based, methods for log anomaly detection. This is due to a general increase of system complexity, which has made manual methods a very time consuming and difficult task. The natural language processing based transformer model has seen success in the field of log anomaly detection but may fail in cases where log data is highly unstructured and where anomalous log messages may be far apart. One reason for this could be the transformer model's squared dependency on input length, limiting how many log messages can be used as input to the model. So called sparse transformers address this problem with different variants achieving sub-quadratic dependencies on input length. In this project, one transformer-based model and two sparse transformer-based models are investigated and compared in their effectiveness for log anomaly detection in system log files.

The transformer-based model uses a BERT-style architecture whereas the two sparse transformer-based models use a Big Bird- and a Longformer-type architecture. All three models then have a hyperspherical loss function applied directly on the raw model outputs. These outputs are then used to compute an anomaly score which in turn is used to classify a log message as being either normal or anomalous. Furthermore, all models are scaled down and trained from scratch on system log files in order to make them fit on the GPU. The log files used for evaluation in this project are the two open source data sets Hadoop Distributed File System (HDFS) and BlueGene/L (BG/L) as well as one Ericsson system log data set.

All models are evaluated on annotated test data sets and the two main metrics looked at are F1-scores and estimated anomaly score probability density functions. Across the data sets, the highest F1-scores are achieved by the sparse transformer based models suggesting that the increased input size does affect performance. However, the highest F1-scores vary among the data sets with some only being slightly higher than those achieved by the transformer-based model, suggesting future to work explore other areas to increase performance. The estimated anomaly score probability density functions show a general tendency of the models failing to separate normal and anomalous log messages, although some models show hints of separation on certain data sets.

Keywords: log anomaly detection, natural language processing, transformer, sparse transformer.

Acknowledgements

We would like to thank Ericsson and in particular our supervisors at Ericsson, Jesper Derehag and Åke Johansson for the opportunity to try our hands at a very challenging but interesting problem. Furthermore they were very helpful and provided regular opportunities for discussion.

We would also like to thank our supervisor/examiner at Chalmers, Larisa Beilina, who provided us with good feedback on the report as well as being very helpful in regards to academic support.

Joel Harf Abili Marco Cuskic, Gothenburg, June 2022

CHALMERS
UNIVERSITY OF TECHNOLOGY

Contents

1
Introduction

In this section, the background of this thesis along with related work will be presented. Furthermore, the aim of the thesis together with the guiding research questions are presented. Also brought up is the scope and delimitations taken into consideration in this thesis.

1.1 Background

In many areas, logging is an important feature when considering the development and maintenance of different systems [1, 2]. Logging is useful for communicating information about a system to its user and it is important to define how, what, and when to log, and how to make use of logs to extract useful data that can be used for analysis. Such information could for instance regard errors, which can then be used for troubleshooting, but it could also regard successful requests which could give an insight into how users work with a system.

In recent years, interest has grown for full automation of the logging process in order to aid with the increasing complexity of inspecting logs that arise from the increased complexity of a system [2, 3]. Semi-automated methods typically lack in this area due to the limitations in their sets of rules and their sensitivity to changes in the system [4] and the requirement of manual labor. Thus, full automation by the use of machine learning and artificial intelligence has grown as an area of interest.

Automation in this sense is of particular interest in the field of log anomaly detection, in which one wishes to be able to distinguish between normal and abnormal sequences of log messages [5]. Recently, deep learning methods, and in particular, recurrent neural networks (RNNs) have been used in this endeavor in several works [5, 6, 7]. This approach is quite intuitive as RNNs have proven useful for modeling sequential data, doing so by capturing sequential information via the recurrence formula. The general approach is to train the RNN to capture normal log message patterns and to label any violation of this pattern as an anomaly. As the name suggests, however, RNNs are only able to look at contextual information in one direction (either right or left context, not both) [8]. They also have the problem of vanishing gradients. To this end, the transformer model and *Bidirectional Encoder Representations from Transformers* (BERT) specifically has seen success within the field of log anomaly detection as it is able to capture contextual information in both directions and it does not have problems with vanishing gradients [9].

One drawback of the transformer model is its inherent dependency on sequence length. The standard transformer and BERT has a limit of 512 tokens and will fail to handle longer sequences. In the field of log anomaly detection, erroneous log messages in unstructured log files are often very sparsely distributed, which means that standard transformer-based solutions will often fail to capture the error signal inside its limited context window. Recently, studies have been issued investigating the possibility of using so-called sparse transformers to mitigate the standard transformers' inability to handle long sequences [10, 11].

This project was performed at Ericsson and the issue that the project considers is that the error signals in the system log files are very sparse, i.e. the problematic log messages in the log files are typically very sparsely distributed. In the case of log anomaly detection, this becomes problematic since the files typically contain a very large amount of log messages. As mentioned previously, transformer-based methods are sensitive to the sequence length, which limits the input size and therefore what part of the log file that can be given as context. Furthermore, a restriction was set by Ericsson that we are to keep anyform of data preprocessing at minimum.

1.1.1 Basic Structure of Log Messages

The structure of a log message can vary, with different developers, and is not fixed. However, there are some essential components that often are present in a message. These parts consists, for instance, of time stamps, facility codes, severity levels, and messages. Furthermore, these components can be written in various ways by developers and there is no "correct" way to write a component. A log message can, for example, be written as

```
March 4 08:42:23.416: INFO ABCD-XYZ: Connected to server
```

In this specific log message, the timestamp is written as

```
March 4 08:42:23.416.
```

Another way to write a timestamp could be

```
2020-03-04-08.42.23.416.
```

Therefore, there exists variation in both the structure of a message and components, making log messages complex. The complexity and the huge volume of the log data are two major challenges in anomaly detection [12].

An example of different severity levels of a log message is shown in Tab. 1.1.
Emergency messages implies that the system is unusable and alert messages indicates that one needs to act instantly to correct an erroneous condition [13]. Critical messages report that the system is in a critical condition and error messages indicates that something in the systems needs to be corrected immediately. A severity level

Table 1.1: Example of severity levels of a log message. For each severity level, there is a corresponding keyword and a value. The values ranges from 0 to 7.

Severity level	Keyword	Numerical Code
Emergency	emerg	0
Alert	alert	1
Critical	crit	2
Error	err	3
Warning	warning	4
Notice	notice	5
Informational	info	6
Debug	debug	7

of error and warning implies error and warning conditions, respectively. A notice level means that the condition can be considered to be normal, but significant. As for informational and debug levels, the messages are to be seen as informational and the need to debug. One naive thought, is that there exists a correlation between the severity level and abnormal log messages. This is not entirely correct, since the anomalies can consist of different severity levels.

1.1.2 Anomaly Types

Anomalies can be seen as rare messages in the log data, deviating from normal messages. The deviation from the normal messages can take different shapes. Anomalies can therefore be grouped into three major types: point anomaly, conditional anomaly, and collective anomalies [12].

The first type of anomaly is a *point anomaly*. A point anomaly consists of a data instance which deviates from the normal pattern of the data, where the instance is isolated from the context. Therefore, this type of anomaly does not depend on the context, and can be called a *non-contextual anomaly*. This is the most common type of anomaly [12]. The second type of anomaly is the *conditional anomaly*. The conditional anomaly depends on the context, why it can be referred to as a *contextual anomaly*. Conditional anomalies are data instances that are considered to be anomalies, and that this consideration is based on a specific context or condition. The third type of anomalies are the *collective anomalies*. Collective anomalies are instances that, when viewed in isolation, are not considered to be an anomaly. However, when the data is considered as a group of instances, it is said to be collective anomalies.

1.2 Related Work

1.2.1 Log Anomaly Detection

Automated methods for log anomaly detection have become ubiquitous due to the increasing complexity of systems and consequently the increased difficulty to rely on semi-automated methods. Machine learning based approaches such as the works by Zhang et al., Xu et al. and Reidemeister et al. have explored log anomaly detection using *Support Vector Machine* (SVM), *Principle Component Analysis* (PCA) and *Decision-Tree* (DT) based methods respectively [14, 15, 16]. More recently, deep learning based methods along with NLP techniques have shown promising results, exploiting semantic relationships between log messages [17]. These include but are not limited to RNN-, CNN- and transformer-based models [6, 8, 18, 19].

Most of these techniques follow similar steps that include some form of log data preprocessing. One commonly used preprocessing step is log parsing, which aims to structure the raw log messages into a more digestible format, so called log templates (also referred to as *log events* or *log keys*), which are then analyzed instead of the raw log messages. This step, however, introduces errors which affects the performance of the anomaly detector. This is one of the motivations behind alternatives not relying on log parsing, such as in the work by Van-Hoang et al. [3].

One deep learning method for log anomaly detection is *DeepLog*, introduced by Du et al. which leverages *Long Short-Term Memory* (LSTM) in order to detect log anomalies [6]. They first parse the log data in to a structured format and then train the model to learn normal log patterns. Then, during the detection stage, the model identifies an anomaly as a deviation from the model prediction.

In the article by Zhang et al. they address the innate instability of log data which they mean comes from the evolution of logging statements as well as the processing noise in log data [7]. They propose *LogRobust*, which is a log anomaly detection model that utilizes semantic information in *log events*. Detection is performed using an attention-based *Bidirectional LSTM* (Bi-LSTM) model that is able to learn contextual information in log sequences. This makes *LogRobust* more robust when it comes to unstable log data.

In an attempt to tackle some of the drawbacks of RNNs for log anomaly detection, Guo et al. leverage BERT, in a self-supervised framework they call *LogBERT* [8]. It is used to learn the patterns of normal log data which done by using two training tasks; 1) masked log key prediction 2) volume of hypersphere minimization. The first task consists of predicting masked *log keys* in a sequence of normal logs. The second task is used to project normal log sequences close to each other in the embedding space. Anomalous log sequences are then identified by using a criterion that the authors introduce.

All of the previously mentioned methods include some form of log parsing as a pre-

processing step to structure the raw log messages. This, however, does not come without its drawbacks. In the work by Van-Hoang et al. they address two errors that arise as a consequence of log parsing [3] and propose *NeuralLog*, a log anomaly detection method that does not require log parsing. The two problems they mention with parsing is the handling of OOV words and semantic misunderstandings which they mean can lead to a loss of important information when it comes to log anomaly detection. *NeuralLog* instead forms a semantic representation of the raw log messages and use a transformer-based classification model to detect anomalies.

In the thesis work by Wall et al. done at Ericsson, they investigate the performance transformer-based framework for log anomaly detection on highly unstructured system log files [20]. More specifically, they use a BERT-based model in conjunction with two different methods for anomaly detection. One method involves using next sentence prediction as a proxy task and the other method applies a hyperspherical loss on the BERT-model output. Additionally they employ two different methods for extracting relevant historical log context, which they call preceding sequence context and basic keyword search context. The preceding sequence context method uses the preceding lines in the log file as context whereas the basic keyword search context method uses lines containing specific keywords as context. They found that their approach generally failed to distinguish between normal and anomalous log messages with some models showing some indication of a normal/anomaly separation.

1.2.2 Sparse Transformer

The transformer model has risen to prominence within the field of NLP due to its parallelizability as a consequence of its attention module. One drawback of the transformer, however, is its inherent, quadratic, time- and memory-dependency on input length. As such, it has been of recent interest to develop transformer-based models that mitigate this.

In the work by Child et al. they address the quadratic scaling issue by proposing sparse factorizations of the attention matrix which reduces the dependency from $\mathcal{O}(n^2)$ to $\mathcal{O}(n\sqrt{n})$, where n is the sequence length [11]. The sparse factorization is done by modifying the attention matrix such that not all tokens attend to one another using specific *attention patterns* (as opposed to the fully dense case in which all tokens attend one another). Child et al. specifically use two patterns which they call *strided* and *fixed* attention. They call networks with this form of factorization, along with a few other changes to the architecture and methodology, *sparse transformers*.

Another attempt to combat the scaling issue of transformers was presented by Beltagy et al. [10]. The model which they call *Longformer*, has a linearly scaling attention mechanism which allows it to operate on longer input sequences. They adapt two local attention matrix patterns, namely *sliding window*- and *dilated sliding window* attention for capturing local context as well as *global* attention on a few input tokens used for classification. These global tokens are allowed to attend all

other tokens in the sequence and vice versa.

Zaheer et al. introduce *Big Bird*, a modified transformer model which also manages to achieve linear scaling [21] by leveraging a sparse attention mechanism. They use the sliding window attention and global attention as in the work by Beltagy et al. as well as *random* attention, the latter in which each query block attends to a certain number of random key blocks.

Whereas the previously mentioned attempts to reduce the quadratic scaling have all relied on fixed attention patterns, Wu et al. introduce *Smart Bird*, a model with a learnable sparse attention pattern [22]. They raise the point that manually designing the attention pattern may come at a cost of being uninformative for context modeling. Instead, they compute a sketched attention matrix with a single-head, low-dimensional transformer. The goal of the transformer is to learn potentially important token interactions and then use this information to sample token pairs based on probability scores obtained from the sketched attention matrix. These samples are then used to create index matrices for different attention heads which in turn are used to form inputs to sparse attention networks.

1.3 Aim

The purpose of this project is to try to mitigate the problem of capturing the error signal in the context provided to a model for anomaly detection. It should however be stressed that the problem is of high complexity, since the data set of Ericsson is highly unstructured. We will mainly explore one area that we think could potentially combat this problem, namely the use of sparse transformers for anomaly detection in unstructured logs. The expectation is that the sparse transformer's ability to process longer input sequences can be used to spot contextual anomalies more effectively than the non-sparse counterparts. More specifically, this thesis will investigate the performance of two sparse transformer-based models (Big Bird and Longformer) on log anomaly detection, leveraging the hyperspherical loss method as mentioned in Sec. 1.2.1.

1.4 Research Questions

The research questions in the project are as follows:

- Can sparse transformers be useful for log anomaly detection when handling very unstructured logs?
- Is it possible to extract explanatory information of the given data set without the usage of parsing methods?

1.5 Delimitations

In this section, the limitations of the project will be discussed. The limitations of the project are listed below:

- In this project, the models used trying to solve the problem will be restricted to transformer-based models.
- Three data sets will be used in this project, namely the Ericsson data set, the Hadoop Distributed File System log data set (HDFS), and the BlueGene/L (BG/L) data set.
- There will not be an extensive focus on hyper-parameter tuning the parameters of the model due to time restrictions.
- Due to time and resource limitations, any form of training models from scratch will be on smaller data sets than many of the larger, pretrained models available which might affect performance.
- The used data will not be subjected to any form of parsing methods as we want to keep any form of preprocessing as limited as possible. This is done per request by Ericsson to keep generalization across data sets as high as possible.

2
Theoretical Framework

In this section, an in-depth description of the theoretical aspects deemed relevant to this thesis is presented. The chapter leads off with an introduction to preprocessing and moves on to present the traditional and sparse transformer models. Then model pretraining, the hyperspherical loss and the used evaluation metrics are discussed.

2.1 Preprocessing

An important step in log anomaly detection is the preprocessing of log data. The degree of preprocessing varies in different works, but the main function of this step is to transform the raw log messages into a format the model can process, i.e. digit form [23].

2.1.1 Log Parsing

Many established log anomaly detection methods employ log parsing as a first step of preprocessing [5, 6, 7, 8]. Log parsing is applied on the raw log messages in order to structure them. Log messages can be divided into two parts, namely a message *header* and a message *body*. The message header usually exists for all log messages and typically contains information such as time and date, type of log and severity level. The body part can be further divided and often contain a variable part and a fixed part, where the fixed part of a log message is the text written by the developers to describe a system event and the variable part contains "runtime-specific" program parameter values. The goal of log parsing is to identify and separate the fixed and variable parts of a log message and to structure them. The fixed parts are what form so called *log templates* and the variable parts make up the key parameters related to a specific log template. Certain models (such as SVM- or Decision Tree methods) require log parsing as a preprocessing step [3]. They use the log templates in order to form so called *log count vectors* (based on occurrence of log templates) which are then used in machine learning models for anomaly detection.

2.1.2 Tokenization

Another approach is to directly *tokenize* the raw log messages. Tokenization is the process of breaking a text down into its constituents (which is typically either words, characters, or subwords). There exists different types of tokenization, such as

word- or character tokenization, which define the constituents (words and individual characters respectively). Another type of tokenization is subword tokenization which is utilized, for instance, in BERT and whose main advantage is that it addresses the problems of word tokenization (out-of-vocabulary (OOV) tokens, large vocabularies, etc.). In this case, common words can be found in the dictionary whereas OOV tokens can be further broken down into subwords the size of individual characters if need be.

2.1.3 Word Embedding

Word embedding in NLP is a learned numerical representation, typically a real-valued vector, of words [24]. The numerical representation is encoded such that words similar in meaning are close to each other in a defined vector space. Word embeddings are typically obtained either through stand-alone unsupervised training of a model or through joint training with a model on some specific language modeling task [25]. Common approaches to obtain word embeddings are through the use of techniques such as *Word2Vec* and *GloVe* which are able to capture semantic similarities between words [26, 27]. Using BERT to obtain word embeddings is another alternative that generates representations of words that depend on the context they are used in [9].

2.2 Transformers

In 2017, Vaswani et al. presented a new network architecture in [28], called the transformer. Prior to the transformer, state-of-the-art methods in the field of natural language processing consisted of Recurrent Neural Networks (RNN's), such as Long Short-Term Memory (LSTM).

In this section, transformers will be thoroughly discussed. Moreover, the section will be dedicated to the original transformer presented by Vaswani et al. in [28], often referred to as the *vanilla transformer*. The section will begin with presenting the general structure of the transformer, and will then move on to explain self-attention more rigorously.

2.2.1 Transformer Architecture

At a high level, the transformer consists of an encoder-decoder structure. The encoders are stacked upon each other, creating a stack of encoders. In the same way, the decoders create a stack of identical layers (decoders). These two stacks of layers can be seen as the two blocks of a transformer. The full architecture of the transformer can be seen in Fig. 2.1. The encoder consists of two sublayers and the decoder consists of three sublayers. The sublayers of the encoder are the multi-head self-attention and the feed-forward neural network. Each sublayer is connected via a normalization layer and a residual connection, see Fig. 2.1. Furthermore, the decoder consists of the two sublayers as the encoder, but with an additional sublayer. The additional sublayer is the masked multi-head self-attention layer. As with the

other sublayers, the masked multi-head attention is connected with a normalization layer and with residual connections.

In the encoders, the dimension of the input sequence must equal the dimension of the output sequence [29]. To see this, let $\boldsymbol{x}$ be a d-dimensional input vector, i.e. $\boldsymbol{x} \in \mathbb{R}^d$. For the residual connections to hold, i.e. for $\boldsymbol{x} + \text{sublayer}(\boldsymbol{x}) \in \mathbb{R}^d$ to be feasible, $\text{sublayer}(\boldsymbol{x}) \in \mathbb{R}^d$ must hold. Here, $\text{sublayer}(\cdot)$ denotes the output of any sublayer of the encoder. Therefore, the dimension of the input sequence will be preserved throughout the encoder.

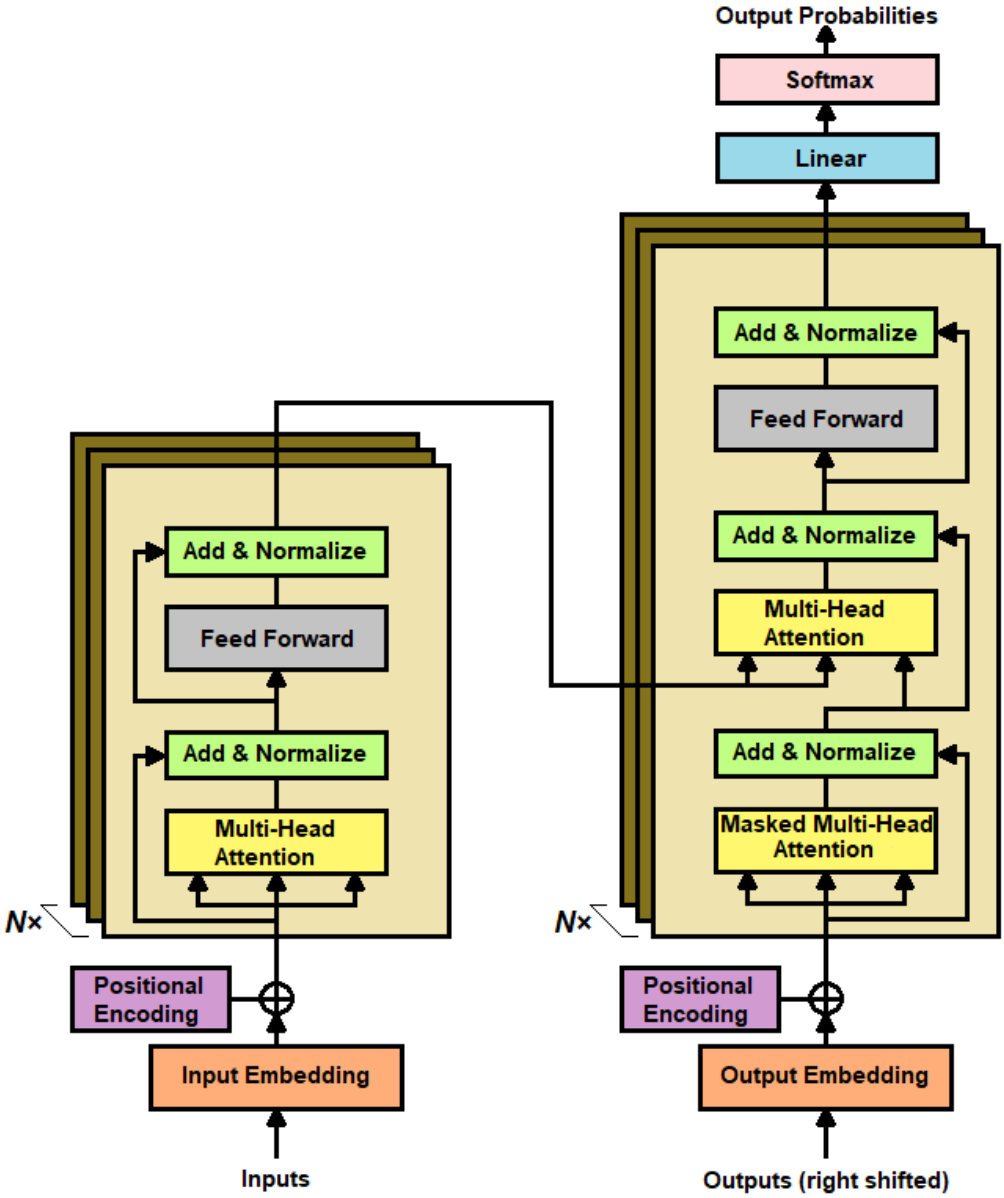


Figure 2.1: The vanilla transformer.

2.2.2 Self-Attention

The core part of the transformer model is the *attention mechanism*. It is inspired from the attention mechanisms of primates, being able to outsource redundant information and put full attention to certain objects [29]. Even if the concept of attention mechanisms are not entirely new, the introduction of them in the field of deep learning is. It has been the foundation of many new models recently. It is mainly adopted to the field of natural language processing, but it is also used in other areas such as computer vision, reinforcement learning, and speech recognition [29]. Consequently, the attention mechanism is used in many modern deep learning applications.

The attention mechanism proposed by Vaswani et al. in [28], is referred to as *scaled dot-product attention*. The name is rather intuitive since scalar multiplications are fundamental in the attention mechanism. In a particular step of the attention mechanism, one applies a *softmax function*, which is defined below.

Definition 1. *For a vector* $\boldsymbol{z} \in \mathbb{R}^N$*, with* $N > 1$*, the softmax function* $\boldsymbol{\sigma}$*:* $\mathbb{R}^N \to (0,1)^N$ *is given by*

$$\boldsymbol{\sigma}(\boldsymbol{z})_j = \frac{\exp(z_j)}{\sum_{n=1}^{N} \exp(z_n)}, \quad j = 1, ..., N. \tag{2.1}$$

Note that the softmax function takes each value of a vector $\boldsymbol{z} \in \mathbb{R}^N$ and transforms it to a numerical value in the interval (0,1). It is done by taking the exponential of the input value and normalizing it.

Before the introduction of the attention scoring function, i.e. the scale dot-product attention, the concept of queries, keys, and values will first be presented. The queries, keys, and values all have a corresponding weight matrices, denoted $\boldsymbol{W_Q} \in \mathbb{R}^{d\times d}$, $\boldsymbol{W_K} \in \mathbb{R}^{d\times d}$, and $\boldsymbol{W_V} \in \mathbb{R}^{d\times d}$. Let the input sequence be denoted by n embedded vectors $\boldsymbol{x_i} \in \mathbb{R}^d$, $i = 1, .., n$. In matrix form, the input sequence may be written as

$$\boldsymbol{X} = \begin{bmatrix} \cdots & \boldsymbol{x}_1^T & \cdots \\ \cdots & \boldsymbol{x}_2^T & \cdots \\ & \vdots & \\ \cdots & \boldsymbol{x}_n^T & \cdots \end{bmatrix} \in \mathbb{R}^{n\times d}. \tag{2.2}$$

Then, for a specific query vector $\boldsymbol{q_i}$, it holds that

$$\begin{gathered} \boldsymbol{q}_i = \boldsymbol{W}_Q \boldsymbol{x}_i, \\ \boldsymbol{k}_1 = \boldsymbol{W}_K \boldsymbol{x}_1 \,,\, \boldsymbol{k}_2 = \boldsymbol{W}_K \boldsymbol{x}_2 \,,\, \ldots \,,\, \boldsymbol{k}_n = \boldsymbol{W}_K \boldsymbol{x}_n, \\ \boldsymbol{v}_1 = \boldsymbol{W}_V \boldsymbol{x}_1 \,,\, \boldsymbol{v}_2 = \boldsymbol{W}_V \boldsymbol{x}_2 \,,\, \ldots \,,\, \boldsymbol{v}_n = \boldsymbol{W}_V \boldsymbol{x}_n. \end{gathered} \tag{2.3}$$

The weight matrices are trainable (change sentence). Furthermore, let n denote the number of queries, and let m denote the number of items in the sequence i.e. the number of keys and values. Also, let the matrices of the queries, keys, and values be denoted as $\boldsymbol{Q} \in \mathbb{R}^{n\times d}$ and $\boldsymbol{K}$, $\boldsymbol{V} \in \mathbb{R}^{m\times d}$. The matrices $\boldsymbol{Q}$, $\boldsymbol{K}$, and $\boldsymbol{V}$ are constructed

such that each row consists of a vector $\boldsymbol{q}_i$, $\boldsymbol{k}_i$, or $\boldsymbol{v}_i$, $i = 1, \ldots, n$, respectively, defined in Eq. (2.3). In other words, the matrices are given by

$$\boldsymbol{Q} = \begin{bmatrix} \cdots & \boldsymbol{q}_1^T & \cdots \\ \cdots & \boldsymbol{q}_2^T & \cdots \\ & \vdots & \\ \cdots & \boldsymbol{q}_n^T & \cdots \end{bmatrix},$$

$$\boldsymbol{K} = \begin{bmatrix} \cdots & \boldsymbol{k}_1^T & \cdots \\ \cdots & \boldsymbol{k}_2^T & \cdots \\ & \vdots & \\ \cdots & \boldsymbol{k}_m^T & \cdots \end{bmatrix}, \tag{2.4}$$

$$\boldsymbol{V} = \begin{bmatrix} \cdots & \boldsymbol{v}_1^T & \cdots \\ \cdots & \boldsymbol{v}_2^T & \cdots \\ & \vdots & \\ \cdots & \boldsymbol{v}_m^T & \cdots \end{bmatrix}.$$

This can also be written as

$$\begin{aligned} \boldsymbol{Q} &= \boldsymbol{W_Q}\ \boldsymbol{X}, \\ \boldsymbol{K} &= \boldsymbol{W_K}\boldsymbol{X}, \\ \boldsymbol{V} &= \boldsymbol{W_V}\boldsymbol{X}. \end{aligned} \tag{2.5}$$

Finally, the output of the scaled dot-product attention can be written as

$$\mathrm{Att}\big(\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}\big) = \boldsymbol{\sigma}\left(\frac{\boldsymbol{Q}\boldsymbol{K}^{\mathrm{T}}}{\sqrt{d}}\right) \cdot \boldsymbol{V} \in \mathbb{R}^{n \times d}. \tag{2.6}$$

As mentioned in section 2.2.1, one can observe that the dimension of the input and output is equal. From Eq. (2.2) and (2.6), it holds that $\dim(\boldsymbol{X}) = \dim(\mathrm{Att}(\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}))$.

The attention scoring function in Eq. (2.6) may be overwhelming at first, but as same as the softmax function, it is also rather intuitive. Firstly, a scalar multiplication is done between the query matrix $\boldsymbol{Q}$ and key matrix $\boldsymbol{K}$, given by $\boldsymbol{Q}\boldsymbol{K}^T$. In this way, the similarity between the matrices is being calculated. After the scalar multiplication, the resulting matrix is normalized, by a factor of $\sqrt{d}$, leading to the equation $\boldsymbol{Q}\boldsymbol{K}^T/\sqrt{d}$. The next step is to turn the resulting elements into probabilities, which is done by applying a softmax function, i.e. taking $\sigma\left(\boldsymbol{Q}\boldsymbol{K}^T/\sqrt{d}\right)$. In the last step, these probabilities are multiplied with the value matrix $\boldsymbol{V}$ to obtain final output, which leads to Eq. (2.6), i.e. the equation $\sigma\left(\frac{\boldsymbol{Q}\boldsymbol{K}^T}{\sqrt{d}}\right) \cdot \boldsymbol{V}$. The elements with the highest values in Eq. (2.6) are the elements which the model will put the most attention to.

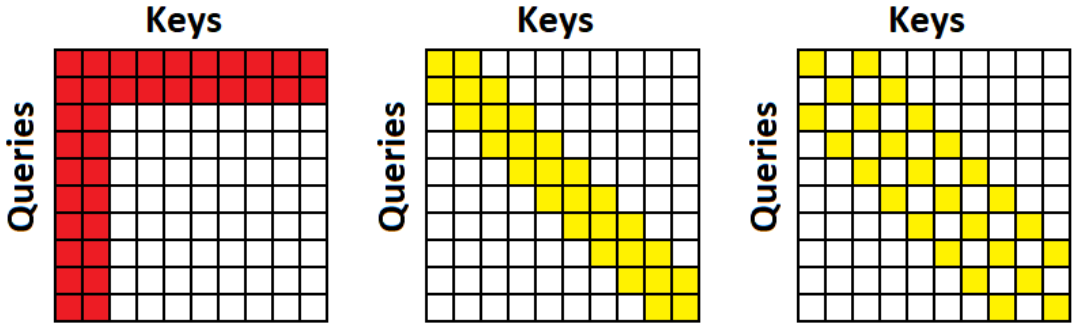
2.3 Sparse Transformers

The transformer has proven useful for many NLP tasks, however, it suffers from a quadratic complexity with regards to sequence length. This is due to the fact that in the self-attention mechanism, every sequence token attends all the other tokens. In mathematical terms, this is due to the scaled dot-product in the expression for self-attention Eq. (2.6). This results in a $\mathcal{O}(n^2 d)$ complexity, where n is the sequence length and d is the hidden dimension. Sparse transformers address this problem and several works aim at reducing the quadratic complexity by applying sparse modifications to the self-attention matrix [11, 21, 22]. The modifications are often either in the form of fixed attention patterns, where tokens attend to each other in a predefined manner, or in the form of a learnable attention pattern, where the model learns which tokens should attend to one another. Both approaches aim to approximate the full, dense, self-attention matrix while they gain a lowered complexity due to the lowered number of operations as a result of the sparse attention pattern.

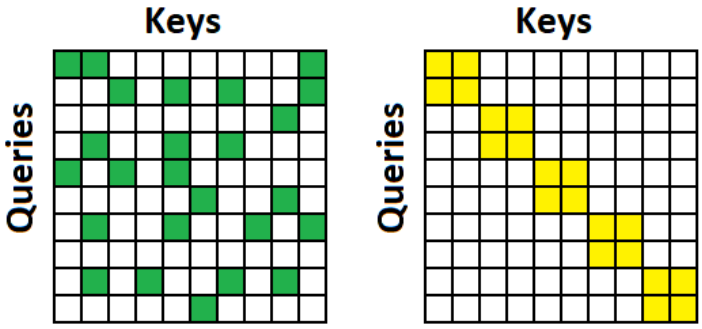
2.3.1 Fixed Attention Pattern

There exists different types of fixed attention patterns which, in general, can be divided in to five categories, namely *global attention*, *band attention*, *dilated attention*, *random attention* and *block local attention* [30]. Examples of the different types of attention patterns are shown in Fig. 2.2. *Global attention* refers to specific tokens in the pattern that are allowed to attend all tokens in the sequence and all tokens in turn are allowed to attend to it. These tokens are used in order to somewhat retain the standard transformers' ability to handle long-range dependencies. *Band attention* (also referred to as *local-* or *sliding window* attention) is employed as a result of data often inhibiting strong property of locality. This type of attention restricts tokens to only attend to a specified number of neighboring tokens. *Dilated attention* is used for its ability to potentially increase the receptive field of band attention by including gaps of dilation (analogous to dilated CNNs). *Random attention* refers to the allowance of certain tokens to attend to a select number of random tokens in order to increase the ability of non-local interactions. *Block local attention* divides the input sequence into different, non-overlapping, query block segments with a corresponding local memory block. Here, tokens in a specific query block attend to tokens in the associated local memory block.

Often, models employ a combination of the above-mentioned attention patterns. Longformer, introduced by Beltagy et al. use a combination of global and local attention by employing band- and global attention patterns. Big Bird, by Zaheer et al. utilize band-, global and random attention patterns. Both models' attention patterns are illustrated in Fig. 2.3.

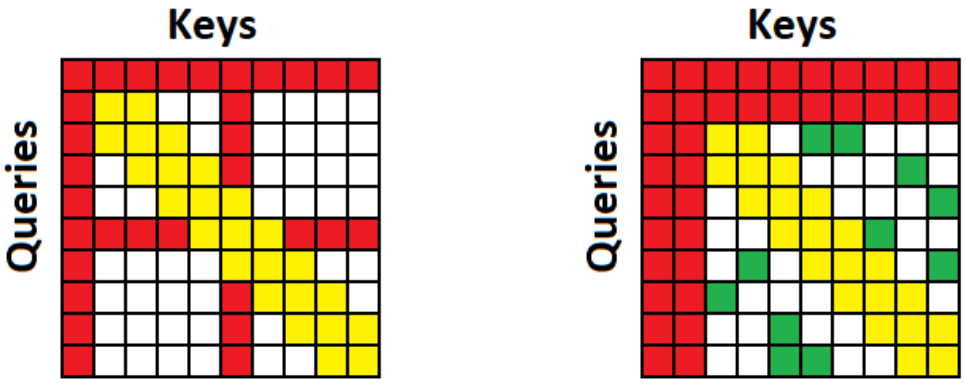


(a) Global attention pattern. (b) Band attention pattern. (c) Dilated attention pattern.



(d) Random attention pattern. (e) Block local attention pattern.

Figure 2.2: Different types of fixed attention patterns. An element in the matrix refers to a specific query-key token pair. Coloured squares indicate that the corresponding attention scores will be calculated for those query-key token pairs, whereas blank squares indicate that the corresponding attention scores will be discarded.



(a) Longformer attention pattern. (b) Big Bird attention pattern.

Figure 2.3: Typical attention patterns of the Longformer and Big Bird models. Longformer uses a combination of band- and global attention. Big Bird uses band-, global- and random attention.

2.4 Model Pretraining

Model pretraining is an important step for many downstream language modeling tasks such as text classification or sentiment analysis. The concept of pretraining

revolves around training a model on some specific task (such as Masked Language Modeling or Next Sentence Prediction which was done by Devlin et al. [9]) before *fine-tuning* model on a downstream task. One of the benefits of pretraining, atleast within the field of NLP, lies in the fact that it can be done on huge amounts of data by big organizations with access to large amounts of resources and then be fine-tuned by smaller groups where data and resources may be scarce.

2.4.1 Proxy Task

In natural language processing one often deals with large amounts of data which makes labeling it a very costly task. As a consequence, self-supervised and unsupervised methods are often sought after. In the case of self-supervised learning in NLP, so called *proxy tasks* (also called *pretext tasks*) are often employed for model pretraining. These tasks commonly involve an algorithm that generates proxy-task-specific labels, allowing for supervised training on a specific proxy task. The hope is that a good choice of proxy task will lead to the model being able to yield good results on downstream tasks (such as text classification, sequence labeling etc.). BERT, for instance, leverages two such tasks, namely *Masked Language Modeling* (MLM) and *Next Sentence Prediction* (NSP) allowing it to be trained on huge amounts of data [9].

2.4.1.1 Masked Language Modeling

The task of MLM was first introduced by Devlin et al. and is a common proxy task for pretraining NLP models [9]. Given a sequence of input tokens, the masked language model randomly masks some of them and the objective is to predict the masked tokens, represented by the `[MASK]` token, given the surrounding tokens. The strength of this proxy task lies in the masked language model's ability to utilize both right- and left-hand side context for prediction, as opposed to unidirectional alternatives. In order to train the masked language model, the final hidden vectors representing the masked words go through a softmax layer over all the words in the model vocabulary, with a loss function. For example, cross entropy loss was used by Devlin et al. The vocabulary is generated by a word embedding algorithm, such as WordPiece.

Since the masked tokens are only present in the pretraining step and not in any downstream task necessarily, this creates a mismatch between pretraining and fine-tuning. One way to combat this is to only replace *some* of the masked words with `[MASK]` tokens and otherwise replacing them with either a random token or leaving them unchanged.

2.4.1.2 Next Sentence Prediction

Next sentence prediction is used in order to make the model learn relationships between sentences. Introduced by Devlin et al., next sentence prediction involves predicting whether a sentence A is followed by sentence B or not [9]. For training, a sentence pair (A, B) is chosen from a corpus and with a certain probability B is a

sentence that actually follows A, otherwise it is a random sentence from the corpus. A label is added to the sentence pair specifying whether B follows A or not. The model is then trained to predict the correct labels.

2.5 Hyperspherical Loss

One approach for log anomaly detection is the one based on using a hyperspherical loss function in place of the traditional sigmoid loss, as in the work by Nedelkoski et al. [31]. The objective is to learn to distinguish between normal and anomalous log messages by forcing normal log samples to be close to the center of a high-dimensional sphere and forcing anomalous log samples to be further from the center. Here, normal log samples are log messages from a target system whereas anomalous log samples are data taken from an auxiliary data set. This effectively allows for the model to be trained in a self-supervised fashion since the data points can be labeled based on which data set they belong to. The idea is that the data from the auxiliary data set provides enough information such that the model can effectively distinguish between auxiliary and normal data, while also being diverse enough to reduce the risk of overfitting.

The hyperspherical loss function, $L_{HS}(\theta)$, is derived from the binary cross-entropy loss and is given by

$$L_{HS}(\theta) = -\frac{1}{n}\sum_{i=1}^{n}(1-y_i)\log l\left(\phi\left(\boldsymbol{x_i};\theta\right)\right) + y_i \log\left(1 - l\left(\phi\left(\boldsymbol{x_i};\theta\right)\right)\right), \tag{2.7}$$

where y_i is the label and $y_i = 0$ if the data point belongs to the target system and $y_i = 1$ if the data point belongs to the auxiliary data set. Furthermore, $\boldsymbol{x_i} \in \mathbb{R}^{d\times|r_i|}$ is the training data with samples from both the target system and the auxiliary data set, where r_i is the number of tokens in the log message and d is the dimension of the vector representation of said tokens. The function $\phi(\cdot\,;\theta) : \mathbb{R}^d \to \mathbb{R}^p$ denotes the encoder network, with parameters θ, which maps the log message input embeddings to a p-dimensional space. Finally, $l(\cdot) : \mathbb{R}^p \to [0,1]$ is a function that maps the encoder outputs to an *anomaly score*, which is used to classify the log message as either normal or anomalous.

In order to ensure compactness of the normal data representations, Nedelkoski et al. set $l(\cdot)$ to be the Gaussian radial basis function $l(\boldsymbol{z}) := \exp(-\|\boldsymbol{z}\|^2)$ such that Eq. (2.7) becomes

$$L_{HS}(\theta) = \frac{1}{n}\sum_{i=1}^{n}(1-y_i)\left\|\phi\left(\boldsymbol{x_i};\theta\right)\right\|^2 - y_i \log\left(1 - \exp\left(-\left\|\phi\left(\boldsymbol{x_i};\theta\right)\right\|^2\right)\right). \tag{2.8}$$

The objective is for the model to minimize the loss function in Eq. (2.8) with respect to the model parameters θ. The minimization procedure is performed using an optimization algorithm. One common such algorithm is gradient descent which iteratively changes the parameter values in the opposite direction of the gradient

w.r.t. the model parameters until the loss function is minimized. The model parameters are then updated to these optimal parameter values.

Gradient descent is motivated by the reason that given a multi-variable function $F(\boldsymbol{x})$ that is defined and differentiable in a neighborhood of a point $\boldsymbol{x}^*$, the most rapid decrease of $F(\boldsymbol{x}^*)$ is in the negative direction of the gradient w.r.t. $\boldsymbol{x}$ at $\boldsymbol{x} = \boldsymbol{x}^*$.

As a result of the aforementioned motivation and given a small enough learning rate $\alpha_n \in \mathbb{R}_+$, it follows that if

$$\theta_{n+1} = \theta_n - \alpha_n \nabla F(\theta_n), \tag{2.9}$$

then $F(\theta_n) \geq F(\theta_{n+1})$, where n is the current step of the gradient descent iteration. Furthermore, let $d_n := -\nabla F(\theta_n)$. Then, an optimal learning rate $\alpha_{n,opt}$ can be found through the following minimization

$$\alpha_{n,opt} = \arg\min_{\alpha_n} F(\theta_n + \alpha_n d_n) \tag{2.10}$$

An extension to gradient descent, and a method that is commonly used in many deep learning applications, is the Adaptive Moment Estimation (Adam) optimization algorithm [32]. This method, as opposed to normal gradient descent, leverages individual and adaptive learning rates for different parameters given by estimates of the first- and second order moments of the parameter gradients. Note that all above mentioned optimization methods are local and as such, convergence of a model is sensitive to the initial values.

2.6 Evaluation Metrics

This section will present the two evaluation metrics used in this thesis, namely confusion matrices and F1-scores.

2.6.1 Confusion Matrix

In machine learning, a confusion matrix is a two by two matrix used to evalute a model's performance. The matrix shows the true positive (TP), false negative (FN), false positive (FP), and true negative (TN) values for a model. Compared to only observing the accuracy of a model, a confusion matrix yields a more complete way to evaluate a model. This is due to the fact that only observing the accuracy can be misleading when the data set is imbalanced.

The structure of a confusion matrix can be seen in Fig. 2.4. The rows consist of the actual class labels, i.e. the ground-truth labels. The columns consist of the predicted class labels. Ideally, one would want to achieve as high TP and TN values as possible.

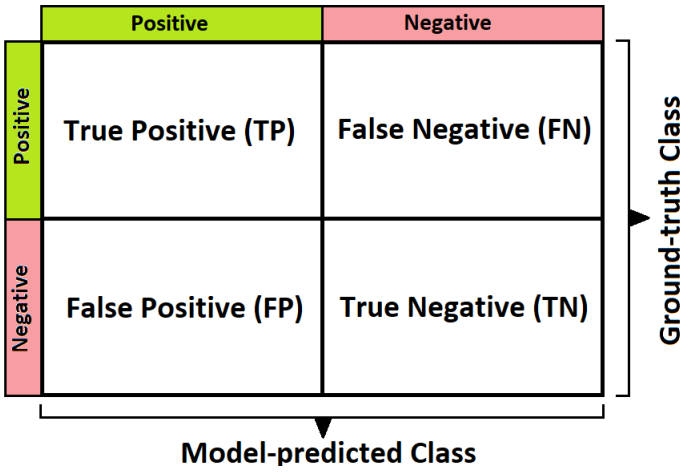


Figure 2.4: The structure of a confusion matrix.

2.6.2 F1-Score

In this thesis, F1-score is used as the main metric. To understand the F1-metric, we will first define precision and recall. Precision is a metric defined as TP divided by the sum of TP and FP, i.e.

$$\text{Precision} = \frac{TP}{TP + FP}.$$

Recall is defined as TP divided by the sum of TP and FN, according to

$$\text{Recall} = \frac{TP}{TP + FN}.$$

The precision and recall, ranging between 0 and 1, gives an indication of the number of FP and FN for a model. The optimal value to achieve for the precision or the recall is 1, since this would imply a FP or FN value close to 0. Furthermore, the F1-score is a harmonic mean between the precision and the recall, given by

$$\text{F1} = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}.$$

As for the precision and the recall, the F1-score ranges between 0 and 1, with a value of 1 being the optimal value.

3
Methods

This section describes the most central methods that were used in the thesis. The setup in regards to hard- and software will be presented as well as a model overview and the used evaluation methods.

3.1 Setup

In this section, the specifications about the soft- and hardware will be presented. Also, the data sets used in the thesis will be discussed.

3.1.1 Soft- and hardware

The code for this thesis is written in Python. The machine learning part of this thesis was implemented using the open source framework PyTorch[1]. Also, Hugging Face[2] was used as basis for all implementations regarding transformer models. All training and evaluation of the different models were performed on Ericsson's remote server, equipped with two Tesla P100-PCIE-16GB GPUs.

3.1.2 Data Sets

In this thesis, we use three different log data sets; HDFS, BG/L and, the Ericsson data set. The former two data sets are fully annotated and are publically available on the open-source AI platform for automated log analysis, LogPAI[3]. The HDFS data set has log messages divided into traces according to block IDs, with each trace belonging to a certain block ID being given a label of either normal or anomalous. The BG/L data set is annotated linewise according to alert category tags. Non-alert messages are labeled whereas alert messages are not. The Ericsson data set is mainly unlabeled with a few log files being labeled linewise.

Since only a small portion of the Ericsson data is labeled, we want to use it in the evaluation. The training data will thus consist of the unlabeled counterpart. In order to have the different data sets be as similarly structured as possible, the labels for the HDFS and BG/L data used in training are removed.

[1] `https://pytorch.org/`
[2] `https://huggingface.co/`
[3] `https://github.com/logpai`

During evaluation on the test data sets, the HDFS and BG/L data sets have been annotated in similar fashion as the Ericsson data set, i.e. linewise with a label of being either a normal or an anomalous log message. As for the HDFS test data, all lines associated with a specific block ID are given the same label as the one that was given to the block id itself. In the BG/L test data, non-alert messages are labeled as normal and alert messages are labeled as anomalous.

In Tab. 3.1, information about the data sets is shown. The table includes the total number of annotated log files available as well as the number of log files used for training, validation and testing.

Table 3.1: A table showing information about the different data sets. The total number of annotated log messages is shown in the first row. The train, validation and test split is shown in the remaining rows. Note that we want to use all of the annotated Ericsson data for evaluation. Also noted should be that the training and validation data for the Ericsson data set is provided separately from the test data set. As for the BG/L and HDFS data sets, the training and validation sets are the label-stripped log messages and are extracted from the full (annotated) data sets. The proportion of training and validation data (of the full data sets) used for the BG/L and HDFS data sets are shown in parentheses.

Number of total annotated log messages as well as train-, validation-, test split.			
	Ericsson	**BG/L**	**HDFS**
Total (annotated)	456 680	4 795 379	11 175 600
Training	1 000 267	1 281 933 (27%)	1 229 316 (11%)
Validation	218 106	284 874 (6%)	335 268 (3%)
Test	456 680 (100%)	474 790 (10%)	1 117 560 (10%)

3.2 Model Overview

The overall framework for the thesis can be seen in Fig. 3.1. First, the log data will be extracted with a basic context extraction algorithm. The basic context extraction selects the most rare word in a log file, and extracts the context thereafter. The selected context will then be fed to the model. Furthermore, the work in this thesis is divided in two main parts: training from scratch and anomaly detection with a hyperspherical loss. All models and the tokenizer used are trained from scratch on log data. The anomaly detection was done by utilizing a hyperspherical loss function.

In this report, we focus on two sparse transformer models; Big Bird and Longformer, along with the BERT model, for log anomaly detection. These have all been trained from scratch with masked language modeling on six log files (two log files for each data set). The number of log files used was mainly determined by the time it took for the training.

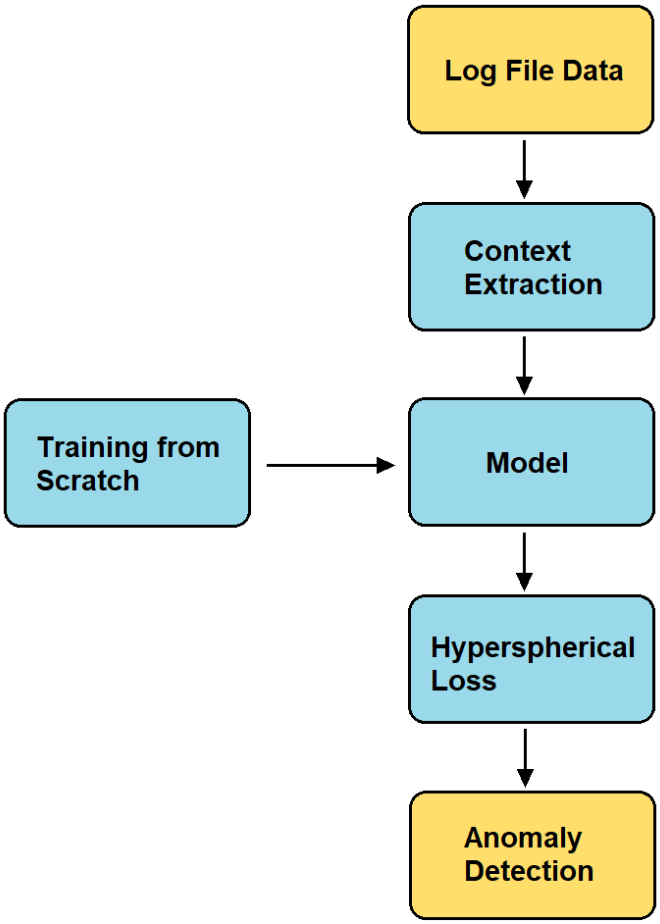


Figure 3.1: Overall framework for log anomaly detection.

The motivation behind choosing to use sparse transformers in this project is due to the fact that we are then able to include more log messages as context, which we hope will lead to a good performance in log anomaly detection. This is due to the sub-quadratic sequence length dependency of the input data.

3.2.1 Context Extraction

The input to the model used for anomaly detection consists of one query log message and several other log messages. The set of log messages used in the input is often described as the *context*. Since the sequence length of both the BERT model and the sparse transformer models are limited, it is desired to catch the ”most relevant” log messages as the input. With the ”most relevant”, one means the log messages with the highest probability to yield satisfying results, in terms of correct predictions. Particularly, one would want to reduce the amount of noisy log messages in the context. Therefore, a satisfying context is necessary for good model performance, why this step is of high importance for the results.

The model used for context extraction in this thesis is built upon a basic keyword

search methodology as proposed by Wall et al. [20]. The key idea behind the keyword search is to find the most rare word in the query line (w.r.t. a pre-defined vocabulary), search it through the preceding log messages in the training set, and to pick the most recent log messages containing the rare word. These lines will then be used as context for the model. The process continues until the maximum number of tokens allowed is reached. If no other lines contain the rare token, or if all preceding log messages have been searched through without reaching the token limit, the lines preceding the query will be used as context instead.

The motivation behind the context extraction algorithm is the limitation of input tokens to the models used. The BERT model has a limitation of 512 tokens. However, this limitation is increased in the sparse models. The sparse models have a maximum input length of 4096 tokens, why a larger context can fit in them. The goal would be to fit at least 10 000 lines of log messages in the context. However, even if one line would correspond to one token, this would still not be possible. This means that the context must be chosen wisely, preferably so that information of at least 10 000 lines would be captured.

3.2.2 Training from Scratch

The pretrained versions of the Big Bird and Longformer models are very large (1.275×10^8 parameters for Hugging Face's `bigbird-roberta-base` and 1.487×10^8 parameters for Hugging Face's `allenai/longformer-base-4096`) which leads to long training times. The main motivation behind training the models from scratch with a custom tokenizer is that if trained on log data, it would allow for longer sequences to be represented by fewer tokens as compared to a tokenizer trained on other corpuses. Training from scratch also allows for scaling of the model architecture, which can be used to reduce the number of model parameters by, for instance, reducing the number of hidden layers and/or reducing the hidden layer size. In this project, the training from scratch includes training a tokenizer on the log data as well as training the two models on masked language modeling, similar to how BERT was pretrained but without next sentence prediction. The reason for not using next sentence prediction was that the model architectures for the sparse transformer models from the Transformers[4] Python library did not have any model heads for training on next sentence prediction available.

In this project, the number of hidden layers is reduced to 3 (from the default 12 layers for both Big Bird and Longformer), which effectively reduces the number of network parameters by a factor of 10 for both Big Bird and for Longformer. The jump from 12 to 3 layers is quite large but the hope is that 3 layers is enough to yield good performance for the somewhat unique structure and language of the log message data. To keep the models consistent and for more suitable comparisons, we also train a BERT model from scratch with three hidden layers even though pretrained versions (such as Hugging Face's `bert-base-uncased`) would fit. The model configurations were set up using the classes `BertConfig, BigBirdConfig`,

[4] `https://github.com/huggingface/transformers`

and `LongformerConfig` from the Transformers library. A summary of the number of parameters for different models are shown in Tab. 3.2. Continuing, we refer to these three, from scratch trained models, as 3L-BERT, 3L-Big Bird and 3L-Longformer.

Still, the model sizes for the sparse transformer in particular, are quite large so in order to make them fit on the GPU:s, gradient checkpointing had to be implemented which decreases the memory usage in exchange for longer training times. The gradient checkpointing was only used for the sparse models as the BERT-based model fit on the GPU without having to use further memory usage reduction. In this project, the `checkpoint` function from PyTorch was used. Other alternatives exist, and the use of this implementation along with its effect on the results will be discussed later in the thesis.

Table 3.2: A table showing the number of model parameters for Hugging Face models `bert-base-uncased`, `bigbird-roberta-base`, and `allenai/longformer-base-4096` and for the 3L-models. This decrease in hidden layers reduce the number of model parameters by a factor 10 for all three models.

Number of Model Parameters			
Model:	**BERT**	**Big Bird**	**Longformer**
# of params. (Hugging Face):	1.095×10^8	1.275×10^8	1.487×10^8
# of params. (3L):	2.854×10^7	3.661×10^7	3.129×10^7

In this step, all models were trained for six epochs on a total of six log files (two per data set). A learning rate of 5×10^{-5} was used. The training loss is saved at every training step and the validation loss is recorded at the end of each epoch.

The tokenizer used in this project is a subword tokenizer trained from scratch on the log files. More specifically, the `ByteLevelBPETokenizer` from the Tokenizers[5] Python library was used. The vocabulary size for the tokenizer was set to 8192 which is rather small compared to the vocabulary sizes in many pretrained models' tokenizers. The reason for this is that the vocabulary for log files is most likely not as diverse as vocabularies required for other, more general, NLP tasks.

3.2.3 Hyperspherical Loss

The model for log anomaly detection is trained using a hyperspherical loss as described in Section 2.5. The target system is set to be the data set the model is currently being trained on. The auxiliary data set is set to be the remaining two data sets. The procedure for generating true and false samples is as follows; a query line is sampled from the target system with probability rate p, or the auxiliary data set with probability rate $1 - p$. Context lines are then, in both cases, generated from the target system using the basic keyword search context method. The pairs

[5] `https://github.com/huggingface/tokenizers`

of query-context are then labeled as true if both query and context are from the target system and false if the query is from the auxiliary data set. In this project, we use a probability rate $p = 0.9$.

Furthermore, the hyperspherical loss is applied to the `[CLS]` token of the model output as it contains information about the entire token sequence. The goal is then for the model to be able to separate true and false samples. Here, an assumption that the three data sets fulfill the assumptions of being informative enough for the model to be able to distinguish between auxiliary and normal data, while being diverse enough to reduce the risk of overfitting, is made.

3.2.4 Anomaly Detection

We interpret the hyperspherical loss as an anomaly score since it tries to separate normal and auxiliary data points. Normal data points are expected to lie close to zero, see Eq. (2.8), whereas auxiliary data points are expected to be forced away from zero. Thus, the anomaly score can be used in conjunction with a threshold value to separate data points into two discrete labels.

All models are trained for two epochs and a batch size of 16 is used for all models and data sets. The Adam optimizer was used with learning rate 1×10^{-6} and weight decay 1×10^{-7}. After each training epoch, a validation epoch is initialized. Training and validation losses are stored after each respective epoch. During validation, an anomaly score threshold is chosen dynamically in order to maximize F1-score, allowing us to construct a confusion matrix. Gradient checkpointing was used only for the Longformer- and Big Bird-based models.

3.3 Evaluation Method

The evaluation in the log anomaly detection step is performed on the 3L-models. All of them use the basic keyword search context extraction method and are trained with the hyperspherical loss function on all three data sets.

When training and validation is done, a model evaluation is performed on the test data sets. Here, similarly to the validation epoch, an anomaly score threshold is chosen dynamically in order to maximize the F1-score and to obtain a confusion matrix. More specifically, for each log message, an anomaly score is computed and the threshold is selected to separate normal messages from anomalous ones in such a way that the F1-score is maximized for the model predictions.

The models are compared to a baseline model that always predicts a positive result. More specifically, an untrained model is directly evaluated on and modified such that only anomalies are predicted. Thus, any model obtaining lower F1-score than

the baseline model would be a model performing worse than an always-positive predicting model. Since the model predictions are set to 1 (anomaly) for all data points, it is independent of which model is used. In this project, an untrained version of the BERT-based model was used as base for the baseline model. In order to deal with class imbalance in the data sets, the F1-score is looked at rather than accuracy as a measure of performance.

The results for the log anomaly detection step will be presented in the form of obtained maximum F1-scores as well as estimated anomaly score probability density functions for each model on each data set. The estimated probability density functions looked at will illustrate the distribution of the anomaly scores colored according to the model predicted label for a specific data point, giving some indication of the model's ability to separate normal and anomalous log messages. These distributions are computed using the `kdeplot` function from the Seaborn[6] Python library. Moreover, confusion matrices for the different models will be provided in the appendix.

[6] `https://seaborn.pydata.org/`

4
Results

In this section, the central results obtained in this project are presented. It is divided into two sections; one training from scratch section and one log anomaly detection section. First, the results obtained from training the models from scratch will be presented, followed by the log anomaly detection results.

4.1 Training from Scratch

In this section, the training- and validation loss curves are illustrated for the training-from-scratch step for all 3L-models. First, 3L-BERT, will be presented, followed by 3L-Big Bird and 3L-Longformer.

4.1.1 3L-BERT Model

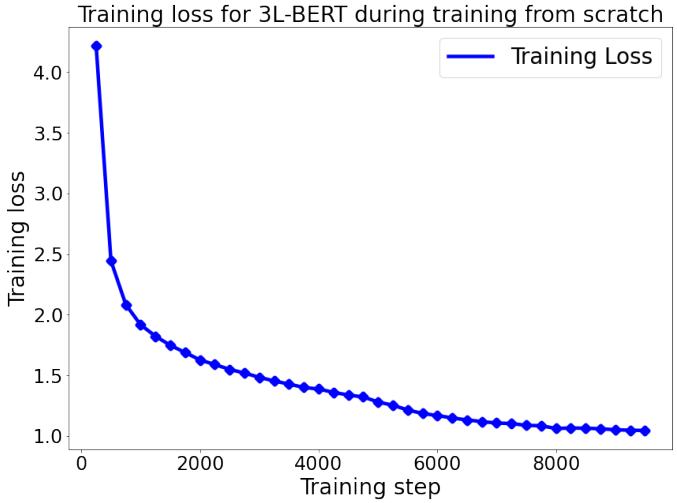
Training- and validation losses for the 3L-BERT model during training from scratch is shown in Fig. 4.1. Both losses seem to converge at the end of the run with a stronger convergence for the training loss compared to the validation loss. The model was trained for six epochs and losses were stored every 250 steps and after every epoch for training- and validation loss respectively.

4.1.2 3L-Big Bird Model

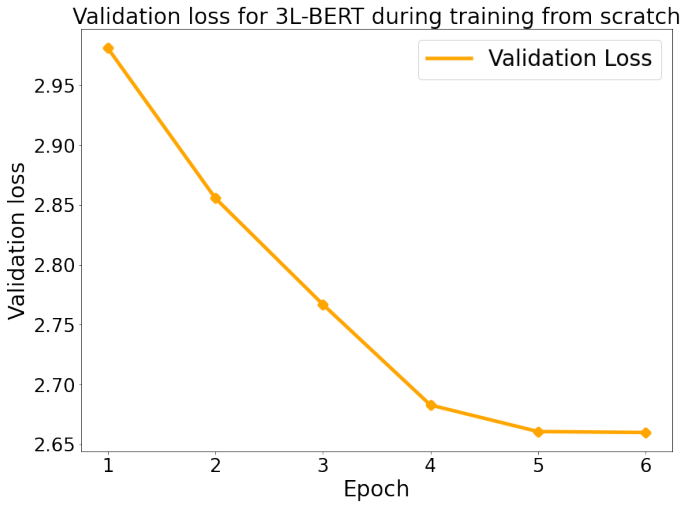
In Fig. 4.2, the training- and validation losses for the 3L-Big Bird model during training from scratch can be observed. In similarity to the 3L-BERT model, both losses seem to converge with a stronger convergence for the training loss than for the validation loss in terms of change in loss from start to finish. The model was trained for six epochs and losses were stored every 250 steps and after every epoch for training- and validation loss respectively.

4.1.3 3L-Longformer Model

The training- and validation losses for the 3L-Longformer model converge, as for the two other models, in similar fashion, see Fig. 4.3. For all models, it is also observed that the scales on the y-axis are on the same order for both the training- and validation losses. The model was trained for six epochs and losses were stored every 250 steps and after every epoch for training- and validation loss respectively.



(a) 3L-BERT model training loss during training from scratch.

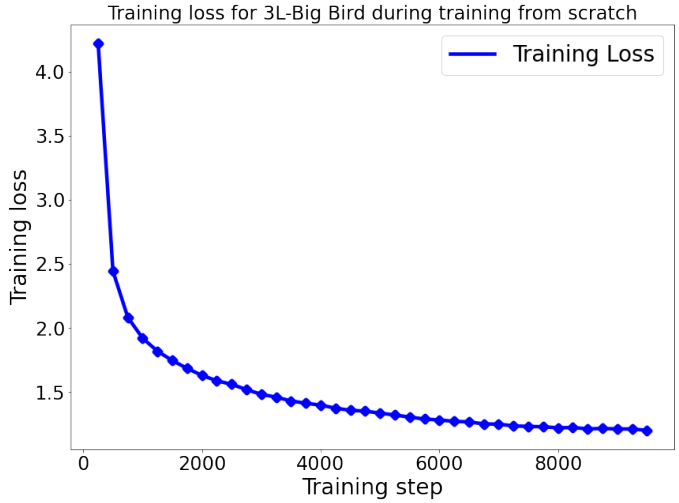


(b) 3L-BERT model validation loss during training from scratch.

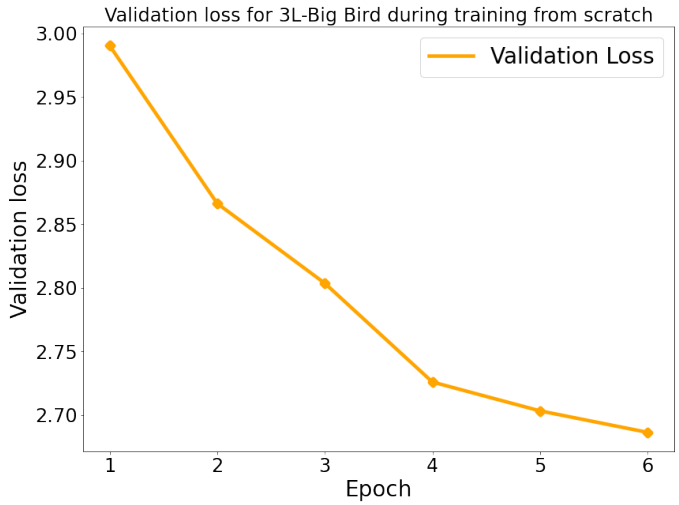
Figure 4.1: These figures, **(a)** and **(b)** respectively, illustrate the training- and validation losses for the 3L-BERT model during the training-from-scratch step. The training loss seems to have converged at the end of the training cycle, whereas the validation loss seems to converge albeit not as strongly as the training loss. Note that training loss was stored every 250 training steps and the validation loss was stored after each epoch.

4.2 Anomaly Detection

In this section, the results of the log anomaly detection step of the project are presented. F1-scores and plots over estimated anomaly score probability density functions will be presented below. It should be noted that the distribution plots are smoothed out by the use of the `kdeplot` function in the `seaborn` Python library. Furthermore, as the training times of the sparse transformer-based models in particular were an important factor in this thesis, the total training and evaluation



(a) 3L-Big Bird model training loss during training from scratch.



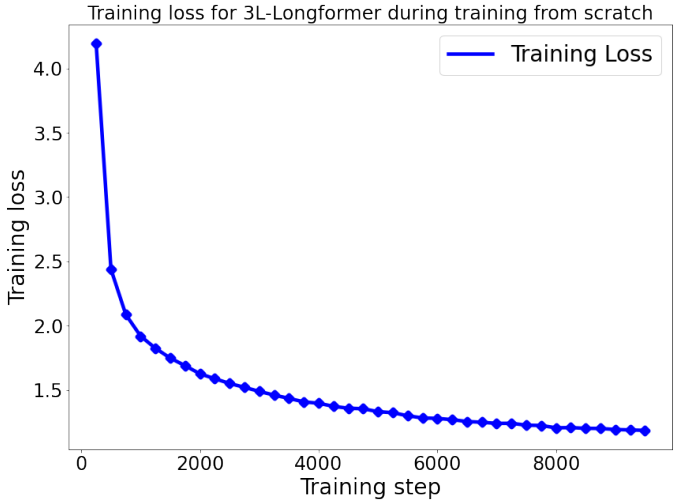
(b) 3L-Big Bird model validation loss during training from scratch.

Figure 4.2: These figures illustrate the training- and validation losses (**(a)** and **(b)** respectively) for the 3L-Big Bird model during the training-from-scratch step. Similar to the 3L-BERT model, the losses show indications of converging. Note that training loss was stored every 250 training steps and the validation loss was stored after each epoch.

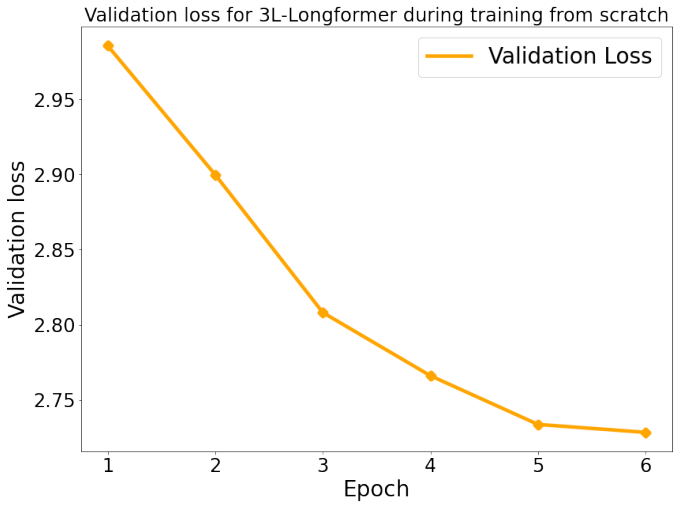
times will be presented for each model. In App. A.1, the confusion matrices will be provided for each model on the three data sets.

4.2.1 F1-scores

In Tab. 4.1, the F1-scores obtained by maximization w.r.t. the anomaly threshold are shown. Also, the baseline F1-scores, in which a model was made to only predict positive results (i.e. only predict anomalies) are shown. All F1-scores are computed on the test data sets.



(a) 3L-Longformer model training loss during training from scratch.



(b) 3L-Longformer model validation loss during training from scratch.

Figure 4.3: These figures illustrate the training- and validation losses (**(a)** and **(b)** respectively) for the 3L-Longformer model during the training-from-scratch step. Similar to the previous model, the losses show indications of converging. Note that training loss was stored every 250 training steps and the validation loss was stored after each epoch.

It is observed that many of the maximum F1-scores lie close to or are in level with the baseline scores for all data sets. There are a few stand-out scores, primarily that of the Big Bird-based model, achieving a maximum F1-score of 0.4980 for the BG/L data set. Also, the Longformer-based model achieves relatively high maximum F1-scores for the HDFS and BG/L data sets. For the Ericsson data set, all models achieve maxmimum F1-scores comparable to that of the baseline, with a small increase achieved for the Longformer-based model at a value of 0.1378.

Table 4.1: A table showing the F1-scores obtained by maximization w.r.t. the anomaly threshold for the different models. All F1-scores are computed on the test data sets.

Maximum F1-scores for the different models.			
	Ericsson	**BG/L**	**HDFS**
Baseline	0.1328	0.1145	0.0140
3L-BERT	0.1332	0.1145	0.0168
3L-Big Bird	0.1328	**0.4980**	0.0174
3L-Longformer	**0.1378**	0.3426	**0.1551**

4.2.2 BERT-based Model

In Fig. 4.4, the estimated anomaly score probability density functions are shown for the BERT-based model. The anomaly score probability density functions are shown for all data sets and are computed on the test data sets.

The anomaly score probability density functions for the BERT-based model in Fig. 4.4 shows a tendency of the model being unable to distinguish normal and anomalous log messages, for all data sets. This is indicated by the fact that the two distributions (for the true labels *normal* and *anomaly*) overlap.

4.2.3 Big Bird-based Model

The estimated anomaly score probability density functions for the Big Bird-based model are shown in Fig. 4.5. These are computed on the test data sets for all three data sets.

The estimated anomaly score probability density functions for the Big Bird-based model, as seen in Fig. 4.5, generally contain a lot of overlap between the two distributions which, as earlier discussed, indicates that the model fails to distinguish between normal and anomalous log messages. For the BG/L data set, we see some indication of the normal/anomalous separation as most of the normal log messages are distributed to the left of the threshold and a large part of the anomalous messages being distributed to the right. Also noteworthy is the location of the threshold (between 27.7-27.8) for all data sets which is significantly larger than for the BERT-based model.

4.2.4 Longformer-based Model

In Fig. 4.6, the estimated anomaly score probability density functions are shown for the Longformer-based model. These are computed on the test data sets for all three data sets.

The Longformer-based model's anomaly score distributions, as shown in Fig. 4.6, show a general tendency of being unable to distinguish normal and anomalous log

messages. There is a slight indication of separation for the BG/L data set, in which most of the normal log messages are distributed to the left of the threshold. The distribution for the HDFS data set differs from most of the previous distributions in that it has three distinct peaks around which the anomaly scores are distributed. Notable is that these peaks lie very close to one another, on the order of 10^{-6} more specifically. Similarly to the Big Bird-based model, the anomaly scores are distributed at or around 27.7.

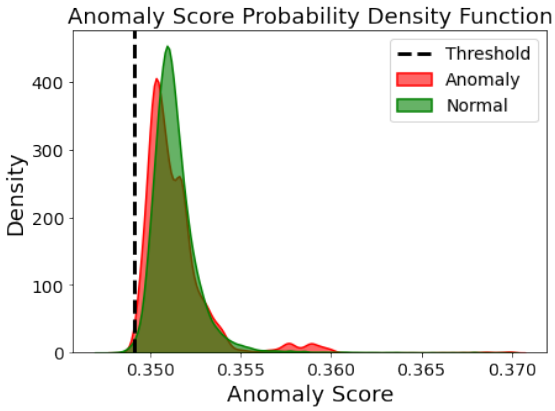
4.2.5 Model Training Times

As previously mentioned the training times for the sparse models in particular were a significant factor in this thesis. In Tab. 4.2, the total time for training, validation and evaluation is shown for each model. Note that these times are only for the models used to produce the results shown in Sec. 4.2.1-4.2.4. Furthermore, the times shown here are only for the anomaly detection step.

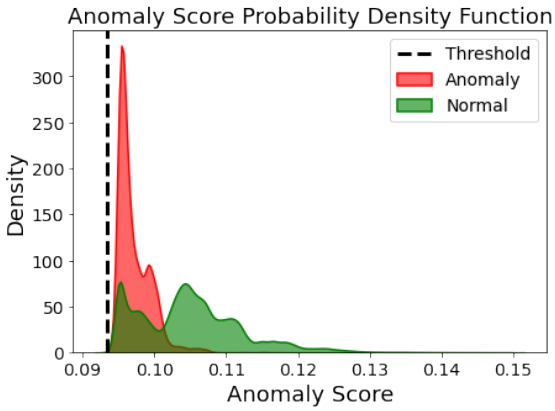
Table 4.2: A table showing the total time (in hours) for training, validation and evaluation for each model on every data set in the anomaly detection step.

Model	**Time [hrs]**
3L-BERT	123
3L-Big Bird	167
3L-Longformer	180

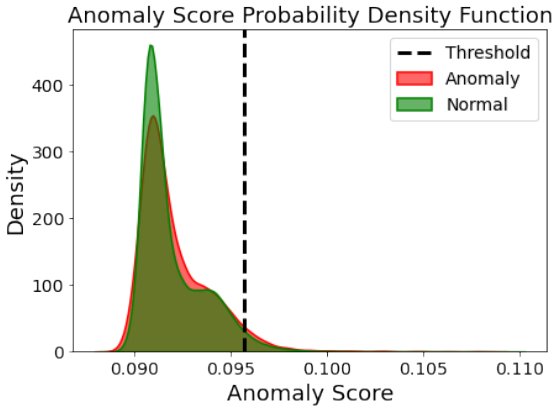
It can be seen that the total training and evaluation times for the various models span from roughly 5-8 days. These times were stored during model training and evaluation.



(a) Ericsson

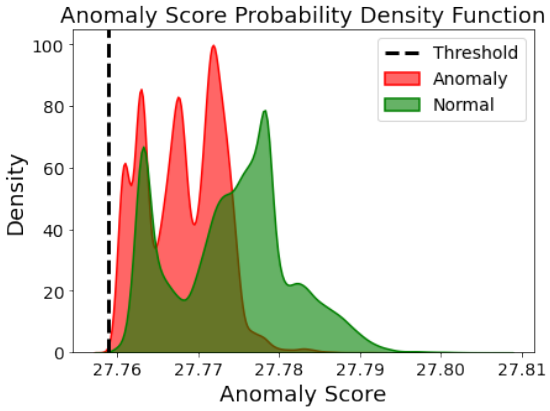


(b) BG/L

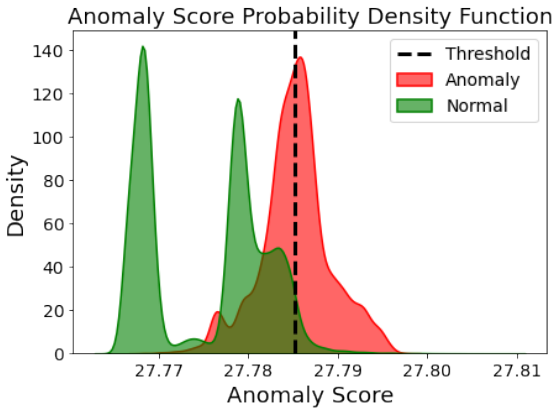


(c) HDFS

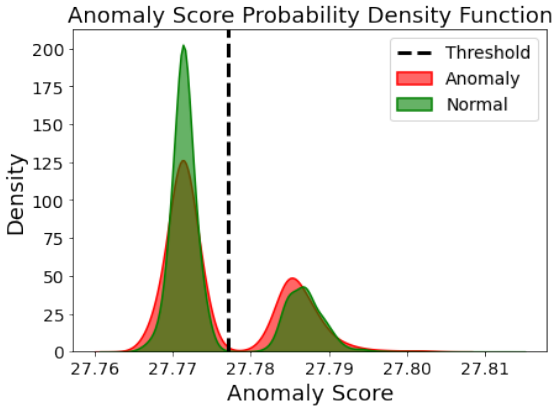
Figure 4.4: These plots illustrate the BERT-based model's estimated anomaly score probability density function for the different data sets. The two colors denote the true labels. Also shown is the anomaly score threshold which maximizes the F1-score. The strong overlap of the two distributions in all the data sets indicates that the model fails to separate the normal and anomalous messages in the hyperspherical sense. Note, these graphs do not display the full distributions but rather are focused on the parts where the distributions contain the most points.



(a) Ericsson

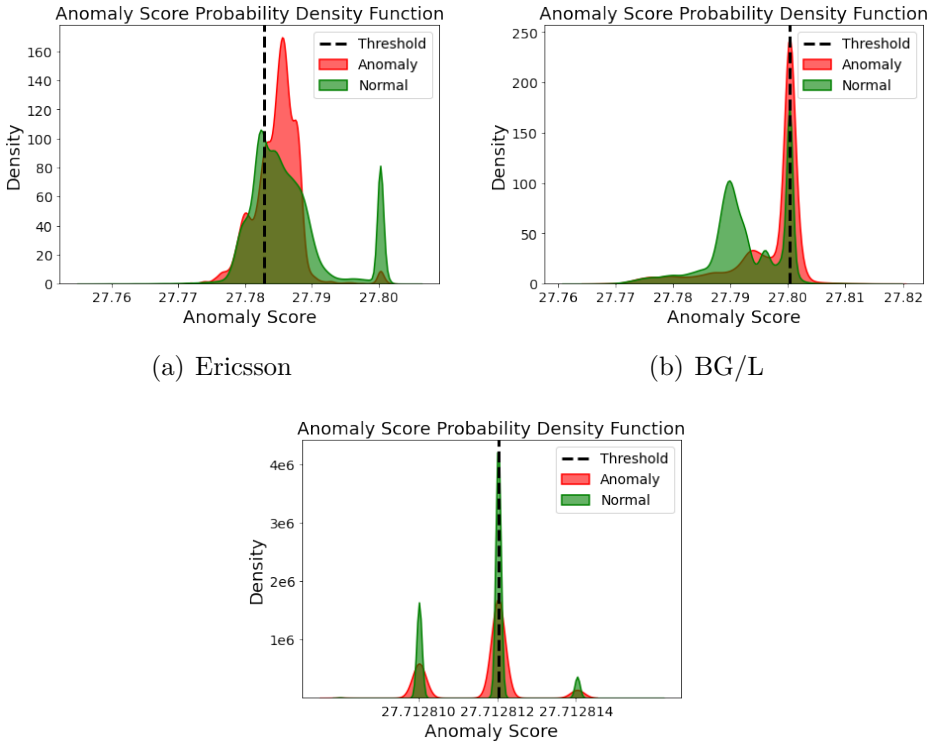


(b) BG/L



(c) HDFS

Figure 4.5: These plots show the anomaly score probability density for the Big Bird-based model for the different data sets. The colors indicate the true labels. Also shown is the threshold that maximizes the F1-score. The distributions of normal and anomalous log messages mostly overlap for the Ericsson and HDFS data sets. For the BG/L data set, the distribution indicates some form of separation of normal and anomalous log messages as seen by the normal messages being distributed mostly to the left of the threshold and a large part of the anomalous log messages being distributed to the right. Note the relatively high threshold value for all distributions as compared to the BERT-based model.



(a) Ericsson

(b) BG/L

(c) HDFS

Figure 4.6: This figure shoes the Longformer-based model's anomaly score probability density function. This plot illustrates the distribution of anomaly scores colored according the true labels (normal or anomalous log message). Also shown is the anomaly score threshold that yields the maximum F1-score. In general, the model seems unable to separate between normal and anomalous log messages, similar to that of the two previous models. The data set on which the model seems to have the best performance is BG/L. This is indicated by the large distribution of normal messages to the left of the threshold. The distribution for the HDFS data set has three distinct peak around which anomaly scores are distributed. Note that the distance between the centers of these peaks are at the order of 10^{-6}. Another noteworthy result is the value of the threshold, which, as with the Big Bird-based model lies between 27.7 and 27.8.

5
Discussion

This section will present a rigorous discussion of the results obtained in this project. Also, potential threats to validity of the results as well as possibilities for future work will be discussed.

5.1 Key Results

5.1.1 RQ1: Usefulness of Sparse Transformers for Log Anomaly Detection

In general, none of the models do a great job of distinguishing normal log messages from anomalous ones as indicated by the maximum F1-scores and the anomaly score distributions as shown in Tab. 4.1 and Fig. A.1-4.6. The highest maximum F1-scores for each data set are, however, obtained by the sparse models with the Longformer-based model obtaining highest maximum F1-scores for the Ericsson and HDFS data sets and the Big Bird-based model obtaining the highest score for the BG/L data set. In regards to the distributions, some models on some data sets show indications of separation of normal and anomalous log messages. Most notably might be the Big Bird-based model on the BG/L data set in which it is observed that the model places most of the normal log messages to the left of the threshold and a large size of the anomalous ones to the right. This separation is by far not as clear as it would have optimally been as a large size of the anomalies are located to the left of the threshold. The Longformer-based model shows a similar distribution for the BG/L data set, i.e. hints of, but no clear, separation of normal and anomalous log messages. The BERT-based model on the other hand shows no indication of separation for any of the data sets, as indicated by the vast majority of normal and anomalous log messages being located on the same side of the threshold. This is also reflected by the maximum F1-scores obtained, being similar to those of the baseline on all data sets.

One peculiar result is how the Longformer-based model performed on the HDFS data set, obtaining a maxmimum F1-score nearly ten times that of the others. This peculiarity extends to the distribution plot (Fig. 4.6(c)) in which three distinct peaks are observed very close to each other and the threshold being located at the center of the middle peak. This suggests that the model outputs one of three discrete anomaly scores no matter the input log message. This result, along with the high anomaly score values for the Longformer- and Big Bird-based models, could be

a result of the way gradient checkpointing was implemented and will be discussed further in the threats to validity section.

The results obtained do suggest that the sparse models do in fact perform better than the non-sparse counter part. If this is directly due to the increased size of the input, however, is difficult to say as many other parameters had to be altered as well. Training the models from scratch, training a tokenizer from scratch as well as scaling the models down in order to fit on the GPU may have unforeseen effects on model performance, meaning that input size may not be the only relevant factor. Optimally, a proper ablation study should have been done in which the different factors were varied one at a time, but due to the time and resource constraints no such study could be performed.

5.1.2 RQ2: Possibility to extract explanatory information without use of parsing methods

In order to keep generalization across data sets as high as possible and per request by Ericsson, preprocessing of the raw log data was kept at a minimum. The generally poor ability of being able to separate normal and anomalous log messages, as presented in the estimated anomaly score probability density functions (Figs. 4.4, 4.5 and 4.6), might very well be a reflection of the minimal effort put in to preprocessing. Directly using the raw log data as input keeps the generalizability very high as new data sets could effectively be used without relying on any data set specific log parsing method. The benefit of log parsing however, is that it transforms the typically unstructured format of log files into a more structured one which in many cases might help the model in the anomaly detection procedure.

5.2 Threats to Validity

In general, due to large model sizes, long training times and time constraints, some decisions that were made in this project likely affected the results and could potentially be risks to their validity. Below are the areas which are believed to have had the largest effects.

5.2.1 Fixed Attention Pattern

The two sparse transformers explored in this thesis, Big Bird and Longformer, both use a fixed attention pattern. This means that which tokens attend to one another is pre-determined. This could be an issue since it effectively means that some prior knowledge is assumed about these token relations. Therefore, something that could be interesting to explore are sparse transformers with trainable attention patterns which might allow for the model to learn what token relations are important.

5.2.2 Context Extraction

Another area that most likely influenced the results is the choice of context extraction method, namely the basic keyword search context as described in Sec. 3.2.1. The goal of the context extraction is to provide the model with context meant to be informative enough to result in accurate predictions. Since the basic keyword search approach is based on a keyword search of the most rare word in the query line (w.r.t. the tokenizer vocabulary), a lot of weight is put on the fact that this word is informative enough. If the rare word, however, is not informative enough this approach most likely fails to include relevant context which in turn most likely leads to the models not being able to make any meaningful predictions.

The thought prior to acquiring the results was, however, that the sheer increase in number of tokens available in the context for the sparse models might be enough to see increases in performance over the traditional transformer-based model.

5.2.3 Hyperparameter Tuning

Within the field of machine learning and AI, hyperparameter tuning is often a very important procedure which could affect model performance. However, since the time it takes training the transformer models is quite long, no extensive study of the choice of hyperparameters was done. A couple of choices (for learning rate as well as weight decay) were explored without seeing any large difference in performance which lead to the conclusion that performance was most likely dictated by other factors in the model framework.

5.2.4 Model Architecture

One obstacle encountered in this project is the sheer size of the sparse transformer models. As mentioned previously, the pretrained models of Big Bird and Longformer that are on Hugging Face have $\sim 10^8$ model parameters which are too large to fit on the GPUs. Thus, a reduction of this number was needed in order to use these models. In this project, the reduction of the number of model parameters was done by reducing the number of hidden layers from 12 (in the respective base versions of the three models) to 3. This is quite a drastic change and likely has an effect on the models' performances. Optimally, a comparison in performance between the larger and smaller transformer models would be done.

5.2.5 Training from Scratch

As a result of having to change the model architectures, the benefits of using Hugging Face models trained on huge amounts of natural language corpuses is lost. A motivation for still proceeding with training the models from scratch was that due to the unique structure and language of log messages, only training on log data might suffice for the goal of this project. However, the number of log data to train on was quite small (6 log files in total) and increasing this number might affect the performances. Also, an increase in the number of training epochs might lead to an

improvement in the results. The final models used were trained from scratch on 6 epochs each which was mainly chosen with consideration to the long training times as well as the convergence of the training and validation losses.

Another factor which could have a positive effect on the outcome is the choice of proxy task. In this thesis, we chose to only use the masked language modeling proxy task as a result of there not being any implementations available online for the next sentence prediction proxy task. As such, if time was not an issue one might be able to implement their own next sentence prediction and use it in the training-from-scratch step.

5.2.6 Gradient Checkpointing

As mentioned in the method chapter, the `checkpoint` function from the PyTorch library was used for gradient checkpointing in this project. However, as opposed to the BERT-based model (which was trained without gradient checkpointing) the sparse transformer-based models obtained much higher loss values both during training and evaluation, without seeing any improvement over the two training epochs. This is most likely due to how gradient checkpointing was implemented, as the PyTorch implementation is sensitive to which tensors require gradients and which tensors do not. This was overlooked during the project which led to the model weights being unintentionally frozen during training. To see if this might have been the case, the automated gradient checkpointing implementation from the Transformers Python library was used on the Big Bird model for the HDFS data set, for which training loss along with anomaly score values seemed to decrease during training as had been wanted from the start. Due to the time constraint, the sparse models could not be trained using this, presumably correct, implementation of gradient checkpointing. As a result, this effectively meant that the sparse models gained little to no value from training, which would explain the large values for the anomaly scores for the sparse models.

5.3 Future Work

The work performed in this thesis leaves some interesting questions that could be investigated in future work. In this section, potentially interesting areas to explore in relation to this thesis will be discussed.

In general, for a more reliable indication on whether sparse transformer-based anomaly detection models perform better than their traditional counterparts, future work should look at performing a rigorous ablation study where different parameters are varied systematically. Otherwise one might risk overlooking potentially important parameters which might have an effect on the results.

As discussed in Sec. 5.2.1, the fixed attention pattern in both of the sparse models researched in this thesis might not be optimal. Interesting might be for future research to explore ways in which one could learn the transformer attention pattern

when looking at a specific query line. This would then likely have to be implemented in some form of a two-step model in which both attention pattern as well as the performance on log anomaly detection would have to be taken in to account when updating model parameters. This would then have the benefit of not having to rely on a context extraction step and might lead to the model being able to distinguish better between normal and anomalous log messages. This would however most likely see an even bigger emphasis on time and resource availability as training such a model likely strains these two factors heavily.

Training the models from scratch could also be a potential area of improvement in future work. In particular, one could explore the effect of directly implementing the hyperspherical loss function instead of masked language modeling and/or next sentence prediction. This might yield a positive result as the log anomaly detection step sees only the hyperspherical loss implemented.

6
Conclusion

This thesis has explored the viability of using sparse transformer-based models for anomaly detection in system log files. More specifically, two sparse transformer-based models were leveraged using the Big Bird and Longformer architectures. These were compared to a traditional transformer-based model with a BERT architecture. All models were trained from scratch with a custom tokenizer on system log data from three different sources; Ericsson, HDFS and BG/L. The models were then trained on log anomaly detection using a hyperspherical loss function aimed at separating normal log messages from anomalous ones in an embedding space.

The metric looked at to determine model performance were F1-scores. Also, the anomaly score distribution was used to evaluate the models. The F1-score was chosen as a metric so as to account for class imbalance in the data sets. Furthermore, the anomaly score distribution illustrates the estimated probability density function of anomaly scores colored according to model predicted labels. No model shows a clear separation of normal and anomalous log messages but the two sparse models show indications separation on certain data sets.

Future work in log anomaly detection with sparse transformer-based models should be done with special consideration to time and resources and a proper ablation study should also be performed. The results shown in this report, however, do show some indications of the potential of the sparse transformer-based models.

Bibliography

[1] Sina Gholamian and Paul A. S. Ward. A Comprehensive Survey of Logging in Software: From Logging Statements Automation to Log Mining and Analysis. arXiv, 2021. doi: 10.48550/ARXIV.2110.12489. URL https://doi.org/10.48550/arXiv.2110.12489.

[2] Adam Oliner, Archana Ganapathi, and Wei Xu. Advances and Challenges in Log Analysis: Logs Contain a Wealth of Information for Help in Managing Systems. *Queue*, 9(12):30–40, 2011. ISSN 1542-7730. doi: 10.1145/2076796.2082137. URL https://doi.org/10.1145/2076796.2082137.

[3] Van-Hoang Le and Hongyu Zhang. Log-based Anomaly Detection Without Log Parsing. arXiv, 2021. doi: 10.48550/ARXIV.2108.01955. URL https://doi.org/10.48550/arXiv.2108.01955.

[4] Shilin He, Jieming Zhu, Pinjia He, and Michael R. Lyu. Experience Report: System Log Analysis for Anomaly Detection. In *2016 IEEE 27th International Symposium on Software Reliability Engineering (ISSRE)*, pages 207–218, 2016. doi: 10.1109/ISSRE.2016.21. URL https://doi.org/10.1109/ISSRE.2016.21.

[5] Weibin Meng, Ying Liu, Yichen Zhu, Shenglin Zhang, Dan Pei, Yuqing Liu, Yihao Chen, Ruizhi Zhang, Shimin Tao, Pei Sun, and Rong Zhou. LogAnomaly: Unsupervised Detection of Sequential and Quantitative Anomalies in Unstructured Logs. pages 4739–4745, 2019. doi: 10.24963/ijcai.2019/658. URL https://doi.org/10.24963/ijcai.2019/658.

[6] Min Du, Feifei Li, Guineng Zheng, and Vivek Srikumar. DeepLog: Anomaly Detection and Diagnosis from System Logs through Deep Learning. CCS '17, page 1285–1298, New York, NY, USA, 2017. Association for Computing Machinery. ISBN 9781450349468. doi: 10.1145/3133956.3134015. URL https://doi.org/10.1145/3133956.3134015.

[7] Xu Zhang, Yong Xu, Qingwei Lin, Bo Qiao, Hongyu Zhang, Yingnong Dang, Chunyu Xie, Xinsheng Yang, Qian Cheng, Ze Li, Junjie Chen, Xiaoting He, Randolph Yao, Jian-Guang Lou, Murali Chintalapati, Furao Shen, and Dongmei Zhang. Robust Log-Based Anomaly Detection on Unstable Log Data. In *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ESEC/FSE 2019, page 807–817, New York, NY, USA, 2019. Association for

Computing Machinery. ISBN 9781450355728. doi: 10.1145/3338906.3338931. URL https://doi.org/10.1145/3338906.3338931.

[8] Haixuan Guo, Shuhan Yuan, and Xintao Wu. LogBERT: Log Anomaly Detection via BERT. arXiv, 2021. doi: 10.48550/ARXIV.2103.04475. URL https://doi.org/10.48550/arXiv.2103.04475.

[9] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota, 2019. Association for Computational Linguistics. doi: 10.18653/v1/N19-1423. URL https://doi.org/10.48550/arXiv.1810.04805.

[10] Iz Beltagy, Matthew E. Peters, and Arman Cohan. Longformer: The Long-Document Transformer. arXiv, 2020. doi: 10.48550/ARXIV.2004.05150. URL https://doi.org/10.48550/arXiv.2004.05150.

[11] Rewon Child, Scott Gray, Alec Radford, and Ilya Sutskever. Generating Long Sequences with Sparse Transformers. arXiv, 2019. doi: 10.48550/ARXIV.1904.10509. URL https://doi.org/10.48550/arXiv.1904.10509.

[12] Jisha M. Jose and S. R. Reeja. Anomaly Detection on System Generated Logs—A Survey Study. In Subarna Shakya, Robert Bestak, Ram Palanisamy, and Khaled A. Kamel, editors, *Mobile Computing and Sustainable Informatics*, pages 779–793, Singapore, 2022. Springer Singapore. doi: 10.1007/978-981-16-1866-6_59. URL http://dx.doi.org/10.1007/978-981-16-1866-6_59.

[13] Rainer Gerhards. The Syslog Protocol. Number 5424 in Request for Comments. RFC Editor, 2009. doi: 10.17487/RFC5424. URL https://www.rfc-editor.org/info/rfc5424.

[14] Ming Zhang, Boyi Xu, and Jie Gong. An Anomaly Detection Model Based on One-Class SVM to Detect Network Intrusions. In *2015 11th International Conference on Mobile Ad-hoc and Sensor Networks (MSN)*, pages 102–107, 2015. doi: 10.1109/MSN.2015.40. URL https://doi.org/10.1109/MSN.2015.40.

[15] Wei Xu, Ling Huang, Armando Fox, David Patterson, and Michael I. Jordan. Detecting Large-Scale System Problems by Mining Console Logs. In *Proceedings of the ACM SIGOPS 22nd Symposium on Operating Systems Principles*, SOSP '09, page 117–132, New York, NY, USA, 2009. Association for Computing Machinery. ISBN 9781605587523. doi: 10.1145/1629575.1629587. URL https://doi.org/10.1145/1629575.1629587.

[16] Thomas Reidemeister, Miao Jiang, and Paul A.S. Ward. Mining unstructured log files for recurrent fault diagnosis. In *12th IFIP/IEEE International Symposium on Integrated Network Management (IM 2011) and Work-*

shops, pages 377–384, 2011. doi: 10.1109/INM.2011.5990536. URL https://doi.org/10.1109/INM.2011.5990536.

[17] Rakesh Yadav, P Kumar, and Sunita Dhavale. A Survey on Log Anomaly Detection using Deep Learning. pages 1215–1220, 2020. doi: 10.1109/ICRITO48877.2020.9197818. URL https://doi.org/10.1109/ICRITO48877.2020.9197818.

[18] Mengying Wang, Lele Xu, and Lili Guo. Anomaly Detection of System Logs Based on Natural Language Processing and Deep Learning. In *2018 4th International Conference on Frontiers of Signal Processing (ICFSP)*, pages 140–144, 2018. doi: 10.1109/ICFSP.2018.8552075. URL https://doi.org/10.1109/ICFSP.2018.8552075.

[19] Siyang Lu, Xiang Wei, Yandong Li, and Liqiang Wang. Detecting Anomaly in Big Data System Logs Using Convolutional Neural Network. In *2018 IEEE 16th Intl Conf on Dependable, Autonomic and Secure Computing, 16th Intl Conf on Pervasive Intelligence and Computing, 4th Intl Conf on Big Data Intelligence and Computing and Cyber Science and Technology Congress(DASC/PiCom/DataCom/CyberSciTech)*, pages 151–158, 2018. doi: 10.1109/DASC/PiCom/DataCom/CyberSciTec.2018.00037. URL https://doi.org/10.1109/DASC/PiCom/DataCom/CyberSciTec.2018.00037.

[20] Jakob Wall and Fredrik Viberg. Transformer-Based Anomaly Detection in Highly Unstructured System Logs. Master's thesis, Chalmers University of Technology, 2021.

[21] Manzil Zaheer, Guru Guruganesh, Avinava Dubey, Joshua Ainslie, Chris Alberti, Santiago Ontanon, Philip Pham, Anirudh Ravula, Qifan Wang, Li Yang, and Amr Ahmed. Big Bird: Transformers for Longer Sequences. arXiv, 2020. doi: 10.48550/ARXIV.2007.14062. URL https://doi.org/10.48550/arXiv.2007.14062.

[22] Chuhan Wu, Fangzhao Wu, Tao Qi, Binxing Jiao, Daxin Jiang, Yongfeng Huang, and Xing Xie. Smart Bird: Learnable Sparse Attention for Efficient and Effective Transformer. arXiv, 2021. doi: 10.48550/ARXIV.2108.09193. URL https://arxiv.org/abs/2108.09193.

[23] Shilin He, Pinjia He, Zhuangbin Chen, Tianyi Yang, Yuxin Su, and Michael R. Lyu. A Survey on Automated Log Analysis for Reliability Engineering. *ACM Comput. Surv.*, 54(6), 2021. ISSN 0360-0300. doi: 10.1145/3460345. URL https://doi.org/10.1145/3460345.

[24] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffrey Dean. Distributed Representations of Words and Phrases and their Compositionality. arXiv, 2013. doi: 10.48550/ARXIV.1310.4546. URL https://doi.org/10.48550/arXiv.1310.4546.

[25] Yoav Goldberg. *Neural Network Methods in Natural Language Processing (Synthesis Lectures on Human Language Technologies)*, page 49. Morgan Claypool Publishers, 2017. ISBN 978-1627052986. doi: 10.2200/S00762ED1V01Y201703HLT037. URL https://doi.org/10.2200/S00762ED1V01Y201703HLT037.

[26] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient Estimation of Word Representations in Vector Space. arXiv, 2013. doi: 10.48550/ARXIV.1301.3781. URL https://doi.org/10.48550/arXiv.1301.3781.

[27] Jeffrey Pennington, Richard Socher, and Christopher Manning. GloVe: Global Vectors for Word Representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543. Association for Computational Linguistics, 2014. doi: 10.3115/v1/D14-1162. URL http://dx.doi.org/10.3115/v1/D14-1162.

[28] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention Is All You Need. arXiv, 2017. doi: 10.48550/ARXIV.1706.03762. URL https://doi.org/10.48550/arXiv.1706.03762.

[29] Aston Zhang, Zachary C. Lipton, Mu Li, and Alexander J. Smola. Dive into Deep Learning. arXiv, 2021. doi: 10.48550/ARXIV.2106.11342. URL https://doi.org/10.48550/arXiv.2106.11342.

[30] Tianyang Lin, Yuxin Wang, Xiangyang Liu, and Xipeng Qiu. A Survey of Transformers. arXiv, 2021. doi: 10.48550/ARXIV.2106.04554. URL https://doi.org/10.48550/arXiv.2106.04554.

[31] Sasho Nedelkoski, Jasmin Bogatinovski, Alexander Acker, Jorge Cardoso, and Odej Kao. Self-Attentive Classification-Based Anomaly Detection in Unstructured Logs. arXiv, 2020. doi: 10.48550/ARXIV.2008.09340. URL https://doi.org/10.48550/arXiv.2008.09340.

[32] Diederik P. Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization. 2014. doi: 10.48550/ARXIV.1412.6980. URL https://arxiv.org/abs/1412.6980.

List of Figures

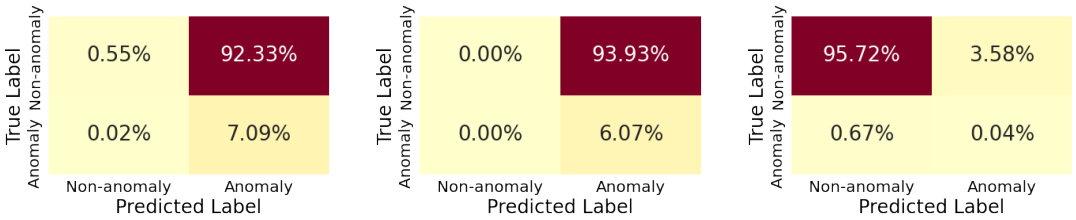
List of Tables

A
Appendix

A.1 Log Anomaly Detection Confusion Matrices

A.1.1 BERT-based Model

In Fig. A.1, the confusion matrices are shown for the BERT-based model. The confusion matrices are shown for all datasets and are computed on the test datasets.



(a) Ericsson (b) BG/L (c) HDFS

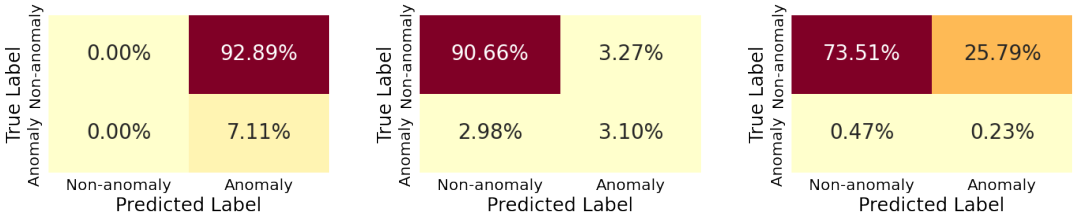
Figure A.1: Confusion matrices for the BERT-based model for the three datasets. For the Ericsson and BG/L datasets, the model seems to only predict anomalies as shown by the low true negative and false negative rates. For the HDFS dataset, the model achieves a high true negative rate, with the second highest being the false positive which at least suggests the model is making different predictions.

As seen in Fig. A.1, the model seems to only predict anomalies for the Ericsson and BG/L datasets as indicated by the low true- and false-negative rates. On the other hand, the model achieves a high true negative rate for the HDFS dataset, with the second highest rate being the false positive rate. As opposed to the two other datasets, this at least suggests that the model is giving different predictions.

A.1.2 Big Bird-based Model

The confusion matrices for the Big Bird-based model are shown in Fig. A.2. These are computed on the test datasets for all three datasets.

The confusion matrices in Fig. A.2 show some variety among the different datasets. For the Ericsson dataset, the zero-valued true negative and false negative rates indicate that the model only predicts anomalies. The model achieves a high true negative rate for the BG/L dataset, with the other rates being of similar size. For



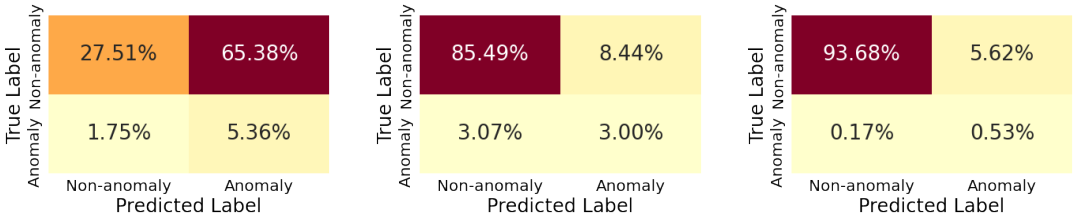
(a) Ericsson (b) BG/L (c) HDFS

Figure A.2: Confusion matrices for the Big Bird-based model for the three different datasets. For the Ericsson dataset, similar to the BERT-based case, the model seems to only predict anomalies as indicated by the zero-valued true- and false-negative rates. As for the BG/L dataset, the model achieves a high true negative rate, with the other rates being of relatively similar size. For the HDFS dataset, the true negative and false positive rates dominate.

the HDFS dataset, the highest rates are the true negative and false positive. As opposed to the two other datasets, here, the model seems to vary its predictions with about 1/4 of the predictions being anomalies and 3/4 being normal.

A.1.3 Longformer-based Model

In Fig. A.3, confusion matrices are shown for the Longformer-based model. These are computed on the test datasets for all three datasets.



(a) Ericsson (b) BG/L (c) HDFS

Figure A.3: These figures illustrate the confusion matrices for the Longformer-based model, for all datasets. The matrix for the Ericsson dataset shows a high value for the false positive rate with the second highest being the true negative rate. For the BG/L and HDFS datasets, the model has a high true negative rate. As opposed to the previous cases, the Longformer-based model spreads its predictions between true and false for all datasets to various extents.

The final model, the Longformer-based one, whose confusion matrices are illustrated in Fig. A.3 seems to spread its predictions more than the two previous models. This holds for all datasets, as opposed to the BERT-based model which only predicts anomalies for the BG/L dataset and the Big Bird-based model which only predicts anomalies for the Ericsson dataset. Furthermore, the Longformer-based model shows high true negative rates for the BG/L and HDFS datasets. For the Ericsson dataset,

the model has a high rate of false positive predictions with the second highest rate being that for the true negative predictions.

MATHEMATICAL SCIENCES
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden
www.chalmers.se

CHALMERS
UNIVERSITY OF TECHNOLOGY