university of groningen / faculty of science and engineering

Multimodal Fusion for System-Wide Anomaly Detection through Multiple Log Files

Bachelor's Project Thesis

Nathan Bosch, s3475344, n.g.bosch@student.rug.nl,
Supervisor: Prof Dr Lambert Schomaker

Abstract: Complex software-intensive systems write information about their runtime behavior into log files, which are frequently used for both post-mortem and real-time analyses. These analyses can be used to determine anomalous occurrences in the log files. Machine learning techniques have previously been used to automate these analyses on single log files. However, as complex systems often produce many log files, each representative of some differing abstraction level or subsystem, it is difficult to use the existing techniques for system-wide anomaly detection. In this thesis, we aim to fill this gap by approaching multi-log anomaly detection from a multimodal perspective, designing and evaluating two early fusion models and one late fusion model to detect anomalies in log files. The models were evaluated on internal base station test data at Ericsson AB and, for the presentation of results, on open-source log file data. On the open-source data, the early fusion models both achieved an F1-Score greater than 0.95 when detecting system anomalies, whereas the late fusion model achieved an F1-Score of 0.88. However, we found that the performance of the models is dependent on the dependencies and relations between the log data, indicating that different fusion strategies may perform better in different situations.

1 Introduction

Many companies develop and operate complex, software-intensive systems that produce log files of system behaviour. These logs are often the primary measure by which system behaviour can be analysed, both in run-time or as part of a post-mortem analysis. These analyses can serve a multitude of purposes, such as debugging system errors and faults, detecting anomalous system behaviour, and building predictive models. As the complexity of systems grow, the need for large-scale logging efforts are great and efforts concerning the generation and analysis of log files have grown significantly in both industry and research over the last few decades.

With complex systems running for significant periods of time, log data grows unmanageably large for performing the traditional, manual analysis techniques in reasonable time. It is for this reason that statistical techniques and machine learning have been employed to perform the detection of anomalies and the classification of the state or future state of the system. The benefits of this are numerous. By delegating analysis tasks to a machine learning model, the great effort of manual analysis can be avoided and sped up considerably. It also allows for real-time detection of anomalies or errors, which may be used to raise alarms and employ graceful degradation.

To manage complexity, large systems are made up of subsystems, each with its own responsibilities and tasks. Furthermore, it is common to run a single product on multiple servers, with numerous applications running on each server. In such cases, it is unreasonable to record holistic system behaviour in a single log file, as this is both impractical and less useful for manual analysis, given the separation and differing levels of abstraction between processes. Therefore, determining cross-log anomalies in a system requires a model which aims to detect anomalies in the system as a whole.

Despite the increasing use of machine learning in the logging analytics community, there is limited research on models or protocols to address an environment in which multiple logs are present. In

cases where there are many underlying processes writing to multiple logs during run-time, it can be more effective to detect anomalous behaviour and predict future system behaviour when incorporating information from multiple log files. Any holistic anomaly detection model which aims to detect anomalies across a system likely needs to take multiple logs into account before it can provide satisfiable results.

In this paper, we propose three models for detecting anomalies in multiple event-based log files. The first two models approach the task through an **early** multimodal fusion approach, where the data from separate logs are integrated into a single input for a machine learning algorithm. The second model approaches the task through a **late** multimodal fusion approach, where machine learning algorithms are first trained on unimodal features before integrating the resultant data afterwards (Snoek, Worring, and Smeulders, 2005).

This paper addresses the following research question: **Does the performance of late fusion anomaly detection models outperform early fusion anomaly detection models when applied to multimodal system events gathered through log files?** The contribution of this paper, therefore, is to define and validate approaches by which multi-log anomaly detection can be done and determine whether early fusion or late fusion of data is the best approach for this task.

The models were evaluated on internal base station test data at a case company, Ericsson AB. For the purpose of the presentation of results, we evaluate the models on open source system log files collected from the loghub data repository (Zhu, He, Liu, He, Xie, Zheng, and Lyu, 2019). Specifically, we evaluate the models on the BlueGene/L dataset, which contains annotated alerts and non-alerts. These logs were generated by a BlueGene supercomputer at Lawrence Livermore National Labs (Oliner and Stearley, 2007).

The remainder of this paper is organized as follows. In the next section, we present the background and related work. Subsequently, in section 3, we present the early and late multimodal fusion approaches to the multi-log problem in greater detail. The data and specifics regarding the implementation and evaluation of the model are outlined in section 4. We provide our results and an analysis thereof in section 5. Finally, we provide a discussion and conclusion in section 6.

2 Background

2.1 System Log Anomaly Detection Models

The quantity of research on the analysis of logs through machine learning and process mining has grown significantly over the last few decades, no doubt related to the increase in system log file size and complexity. As systems and their resultant logs become more complex, the traditional, grep (regex) based analysis of logs struggles to capture faulty sequences of system events or changes in parameter values of certain log statements.

The vast majority of the existing log anomaly detection techniques can be organised in the following taxonomy:

Process Mining: The aim of process mining is to discover and act on real processes extracted from event logs (van der Aalst et al., 2012). In process mining, there are a host of methods which can be applied to an event log to extract processes, such as the HeuristicsMiner Algorithm (Weijters, van der Aalst, and de Medeiros;, 2006). In general terms, process mining techniques attempt to effectively extract the real dependencies between events in logs, which can, subsequently, be used to detect when deviations from normal system behaviour occur.

Workflow Methods: Finite state automata or other dependency representations between log events are employed to determine normal workflows in logs. Both Fu, et al. (2009) and Yu, et al. (2016) use workflow methods to represent normal runtime behaviour, but use additional information such as the time between log events to improve their models.

Rule-Based Methods: These methods define a rule-based approach to detect software failures or intrusions in log files (Lazarevic, Kumar, and Srivastava, 2005). This approach is often very effective, but requires significant effort and domain knowledge to build. Cinque, et al. (2013) achieved strong results when applying a rule-based approach on detecting faults in Apache webserver and TAO Open DDS logs, but this approach requires access to the source code and the ability to access log events before these are written to a log file.

Clustering Methods: One of the most widely used techniques in anomaly detection is clustering. LogCluster (Vaarandi and Pihelgas, 2015) and Beehive (Yen, Oprea, Onarlioglu, Leetham, Robertson, Juels, and Kirda, 2013) use hierarchical clustering and an adapted version of k-means clustering, respectively, to detect anomalies in system events. Anomalies are detected either by determining whether the distance to existing clusters exceeds some threshold, or if it is part of an anomalous cluster. These techniques often require significant amounts of preprocessing, as a single log event does not contain enough contextual information to be effectively clustered.

LSTM Language Modelling: Long Short Term Memory neural networks have seen great success in language modelling, an example of which being sequence to sequence models (Sutskever, Vinyals, and Le, 2014). When applied to log files, LSTM modelling has seen similar success in anomaly detection (Du et al., 2017), (Brown et al., 2018). In 2017, Du, Li, Zheng, and Srikumar proposed DeepLog, an anomaly detection framework which used LSTMs for both system event workflow and system parameter anomaly detection. They achieved state of the art performance when applied to HDFS (Hadoop File System) and OpenStack logs. The self-supervised nature of their training steps does not require labelled data or access to the source code, which sets it apart from rule-based and data mining methods.

Given that sequences in log files are generated through the execution of structured source code, the log file sequences can be interpreted similarly to natural language, where there are complex rules underlying in the relations between log events in sequences. For further explanation of using LSTMs for log anomaly detection, see section 3.

2.2 Multi-log Anomaly Detection

As outlined by Abad, et al. in 2003, there are certain types of anomalies which are not deemed statistically relevant enough by single-log anomaly detection methods. Abad, et al. found that many anomaly detection approaches lead to significant numbers of false positives.

Despite these observations, most existing log anomaly detection research is limited to single log analysis. There have been several cases where multiple logs have been used to detect intrusion or faults in systems. Jia, et al. (2018) developed a network anomaly detection algorithm, which used four logs present in servers. Lu, et al. (2019) combined traffic and multiple log data to develop TLCD (Traffic-Log Combined Detection). While both of these methods operate effectively in their respective domains, the methods used to combine logs require extensive domain knowledge and prior knowledge about the interaction between logs.

In our research, we found that there are few, if any, generalizable approaches to address a multi-log environment, despite a surge of generalizable, machine learning models for single-log anomaly detection.

2.3 Multimodal Fusion

To form a generalizable approach for anomaly detection in multiple logs, it is necessary to fuse information or models derived from multiple logs. These logs present different representations, or modalities, of partially overlapping subsets of the same underlying system. The process by which multiple logs can be incorporated in one model is through multimodal fusion (Atrey, Hossain, El Saddik, and Kankanhalli, 2010).

There are three approaches to multimodal fusion:

1. **Early (or Recognition-based) Fusion** is achieved by fusing the features of the different modalities to generate an input for a model. There are several benefits to this. In early fusion, it is only necessary to train one model for the input data and the early correlations between inputs are recognised in the model, which often improves task performance. However, an early fusion approach can be difficult to apply in some tasks, as significant effort may be required to find an effective strategy to merge features early on. In the case of multi-log anomaly detection, differing average times between events between logs can make the fusion process of log events difficult.

2. **Late (or Decision-based) Fusion** is achieved by forming models for different modalities and merging the outputs of each of the models through a decision function (referred to as fusion at the 'semantic level' by

Atrey, et al.). It is often far easier to perform fusion at the semantic level, as the format of each model's output will be very similar. Late fusion is not necessarily appropriate when there are strong dependencies between the features of the modalities of input, because these dependencies may be lost in the modelling process. Also, more time is frequently required to train the models (Snoek et al., 2005).

3. **Hybrid Multi-Level Fusion** is a combination of early and late fusion. For example, two very similar logs can be fused together at a feature level (early fusion), which can then be fused at the semantic level with other logs (late fusion). This can take advantage of the benefits of both approaches, but may require a greater degree of domain knowledge or exploratory analysis to build an effective model.

Because of the additional requirements that a hybrid multi-level fusion approach contains, this paper will only discuss the early and late fusion approaches to building a multi-log anomaly detection model. Further explanation of how we implement an early and late fusion approach in our task is presented in section 3.

3 Models

In this section, we outline how the early and late fusion models are implemented. The base predictor in these models is the Long Short Term Memory Network. A further explanation of LSTMs is provided in the following subsection, after which we outline how LSTMs can be used for anomaly detection, after which the early fusion and late fusion models are introduced.

3.1 Long Short Term Memory Networks

A Long Short Term Memory network is a type of recurrent neural network introduced by Hochreiter and Schmidhuber (1997). A standard feed-forward neural network architecture has static input to output matching, where activation propagates in one direction. These are used as universal function approximators and update their weights through backpropagation. The limitation of feed-forward neural networks is there is no memory of previous states, causing it to be unsuitable to model dynamic systems, such as sequential input. A recurrent neural network keeps an internal memory of previous states. Because of this, RNNs can serve as function approximators for dynamic systems. This makes them well suited to time-series, sequences and language modelling, as there are relations in each 'step' of input information.

RNNs often suffer from exploding or vanishing gradients. Because of the nature of sequential data, the back-propagation of RNNs is done through time. The *Back-Propagation Through Time* (BPPT) and *Real-Time Recurrent Learning* (RTRL) algorithms achieve this by propagating activity gradients backwards or forwards, respectively (Williams and Zipser, 1995). As discussed by Hochreiter, these methods propagate far backwards or forwards. To calculate the errors for previous time steps, the recurring weight (used to connect the hidden layers to previous time steps) will be multiplied by itself (through gradient descent) multiple times. If the recurring weight is greater than 1 this can lead to exploding gradients and if the recurring weight is less than 1 this can lead to vanishing gradients. This leads to difficulty in modelling long term dependencies. For a more detailed discussion of this, we refer to Hochreiter (1998) and Bengio, Simard, and Frasconi (1994).

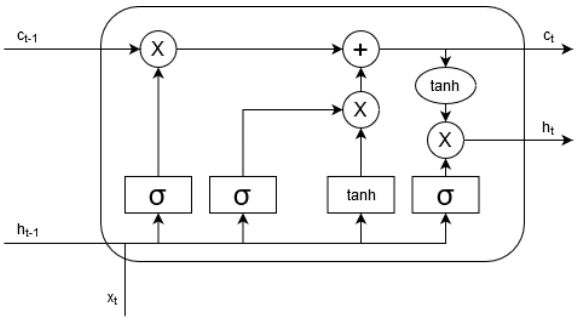

Figure 3.1: LSTM cell (rectangular boxes represent layers, whereas circular elements represent pointwise operators)

LSTMs counteract this vanishing and exploding gradient problems by allowing constant error flow through the LSTM cell (Hochreiter and Schmidhuber, 1997). The architecture is shown in Figure 3.1. Each LSTM cell has three inputs, h_{t-1}, C_{t-1} and

x_t, where x_t is the input at this time step and h_{t-1} and C_{t-1} are passed in from a previous cell. The cell state, C_{t-1} only goes through minor, linear transformations, allowing error to flow through. The output of the cell, h_t, is dependent on the cell state, which allows it to model problems without suffering long term dependency problems to the same degree.

3.2 LSTMs for System Log Anomaly Detection

LSTMs have seen success in modelling time-series (Malhotra, Vig, Shroff, and Agarwal, 2015), languages, and sequences (Sutskever et al., 2014). As mentioned in section 2, system events in logs contain a structure similar to natural language grammar (Malhotra et al., 2015) and the sequential nature of log events make them a prime candidate for LSTM modelling.

In the case of system logs, we define an anomaly as a *deviation from normal workflow behaviour*. An LSTM can, therefore, be trained to predict the next log event given a sequence of previous log events. When exposed to a new log after having been trained on logs expressing normal workflow behaviour, a log event prediction can be compared to the actual, observed log event. In the case that the prediction is substantially different from the observed log event, the observation can be considered anomalous.

There may be cases where there are multiple potential events which could occur from a given input sequence of events. In such cases, there is no deviation from normal workflow behaviour, but the predicted log event may be different from the observed log event. To account for this, we follow an approach outline by Du et al. (2017), where we consider an event anomalous if it is not in the top k predicted log events. This approach is somewhat naive and a better method to account for this is outlined by Malhotra et al. (2015).

3.3 Early Fusion Model

In an early fusion model, fusion is done at the feature level. In this paper, we aim to detect anomalies at the event level. Therefore, we must merge logs at the event level to generate input and output vectors.

In Figure 3.2, the early fusion model is described. The logs are first merged at an event level into a representation where single-log anomaly detection techniques can be applied. Then, sequences and the event which occurs after the sequence are extracted from the merged log. The sequence is served as input to the LSTM model, which predicts the subsequent log event. The predicted event is compared with the observed event. If there is a substantial difference, then we consider the predicted event anomalous.

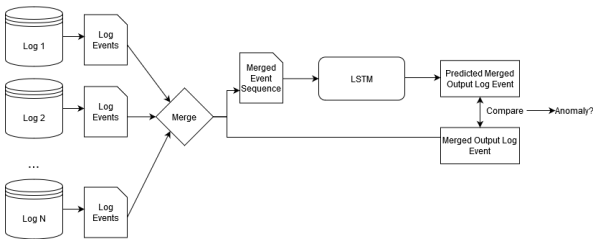

Figure 3.2: Early fusion model architecture for multi-log anomaly detection

There are two methods we identified by which this merge can be done:

(a) **Match events in logs by timestamp.** First, a log is selected to serve as the base log. For each log entry in the base log, we find the nearest entry by time in the other logs. The matched log is therefore a list of tuples, each associated to an entry in the base log and some entries in the other logs. One of the drawbacks of this method is that it does not predict log events, but rather tuples of log events.

(b) **Merge logs into one by timestamp.** We treat the logs as if they are a single log. We add all the log entries into a single merged log sorted by the time when the entry occurred.

The merging strategies are explained in more detail in section 4.

3.4 Late Fusion Model

In a late fusion model, fusion is done at the semantic level. The semantic level, in this case, is a prediction of whether an anomalous event has occurred. To achieve this, independent predictors are

trained on each individual log, where the result of each is combined into a single prediction.

The late fusion model is described in Figure 3.3. Event sequences are extracted from logs, each serving as input to an LSTM. The LSTM is trained to predict the next event in their log. When the observed next event is substantially different from the predicted event, the predictor returns an anomaly. The fusion of the separate predictions is done through leaky integrator fuzzy matching, an explanation of which is defined in the following subsection.

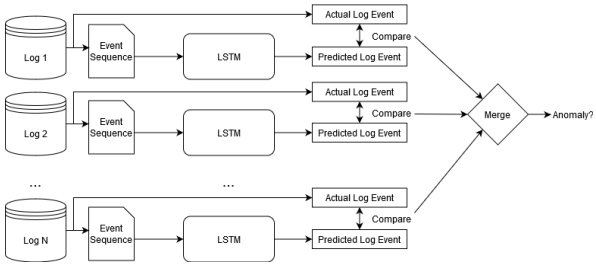

Figure 3.3: Late fusion model architecture for multi-log anomaly detection

As logs can describe separate system processes and are generally event-driven, log entries will often not occur at the same time. The results of an independent predictor will contain the time of the observed deviation from normal workflow, but, even if there is a system-wide anomaly, these times will only infrequently match between logs. The final prediction must, therefore, not merge simply by an exact timestamp, but must fuse the results of the individual predictors based on time.

3.4.1 Leaky Integrator Anomaly Detection

To catch system-wide anomalies we need to merge the resultant predictions of the individual predictive models. One could imagine a case in which one anomalous event occurs and another occurs very quickly afterwards. This scenario could indicate a system-wide anomaly. In practice, if an anomaly is predicted and another anomaly is predicted quickly afterwards, then we can consider these events to be more anomalous than other events in the log files, as there may be some large-scale system fault occurring.

To take advantage of this property of logs, we can use a leaky integrator with an adjustable decay rate to place a greater weight on anomalies which occur in the presence of other anomalies. The process by which these weights can be applied requires a forwards and backwards pass over the logs. In each pass, the weights can be assigned as outlined in pseudocode (see Appendix A.1).

To explain this intuitively, a predicted anomalous event in the log causes an activation of g, whereas a predicted non-anomalous event in the log causes an activation of 0. For each event in the log, its activation as well as the time-decaying activation of previous log events are calculated and summed together, the sum representing the total activation or weight for that log event.

The activation values of each log event in both the forward pass and the backwards pass are summed together, leading to a list of the total activations or weights of each event in the logs combined by time. A threshold is then applied to predict anomalies in the log, leading to the final prediction of the late fusion model.

4 Methods

This section will detail the general preprocessing required to transform a set of logs into input for the models, it will outline the implementation of the two early fusion models and the late fusion model in more detail, and it also outlines what data is used to evaluate the models and the parameters used for each model.

4.1 Preprocessing of Data

4.1.1 General Preprocessing

We have previously defined an anomaly as a deviation from normal workflow behaviour. To be able to detect this deviation in log events, each log event needs to be represented numerically. This is done by representing log events through log keys, which are unique mnemonics in the set of process tokens that will not conflict with other text strings in the log output. A log key can be determined through the extraction of generalizable non-parameter properties of log entries, which can then be mapped numerically. An entire log can, therefore, be represented as a sequence of log keys, which can serve as input to the defined models.

As outlined in section 3, the further matching or merging of log events will require the extraction of timestamps from log files. A preprocessed log can be conceptualized as a list of tuples, containing both the log key and the timestamp of the event. The approaches outlined in this paper, therefore, assume that the timestamp of a log event is extractable and that a log event can be effectively mapped to a log key.

For the defined models, we require that each log file has been preprocessed in the outlined manner.

4.1.2 Early Fusion Match Preprocessing

The preprocessing algorithm is outlined in the pseudocode (see Appendix B.1).

To explain the pseudocode intuitively: First define a base log. For each tuple event in the base log, the timestamps are used to find the closest tuple log event in all other logs. A new event is formed using the log keys of the original tuple event and the closest tuple log events. The list of all new events can be considered a merged log of all input logs.

After this matching is done, each new tuple event of log keys is one-hot encoded and a window is run over the list to generate input sequences and output events, as described in section 3.

4.1.3 Early Fusion Merge Preprocessing

The preprocessing algorithm is outlined in the pseudocode (see Appendix C.1).

To explain the pseudocode intuitively: While we have not reached the end of each log, we loop over the logs and find the log event with the earliest timestamp. The found log event is added to a list which can be considered the new log. It is important to note that overlapping key values for logs are problematic in this case as there will be several log events with the same log key. Therefore, it is advisable to account for this by ensuring there is no overlap.

4.1.4 Late Fusion Postprocessing

For late fusion, no preprocessing other than the outlined general preprocessing is required. However, there is a need to process the outputs of the individual predictors in the late fusion model. This process was outlined in subsection 3.4.1.

4.2 LSTM Log Key Prediction Model

As outlined in section 3, both the early fusion model and the late fusion model use an LSTM to predict a subsequent log event given a sequence of log events.

In figure D.1 in the appendix, we outline how the model is implemented. First, we reduce the dimensionality of the one-hot encoded log events using a dense feed-forward layer. This is done to avoid the LSTM simply predicting the previous log event (Okafor and Schomaker, 2018). The reduced dimensional sequences are then used as input to an LSTM, which predicts a low dimensionality representation of the predicted log event. Then, another dense feed-forward layer is used to scale the reduced representation of the predicted log event to a vector the size of the input vector.

To train the model, the Adam optimizer (Kingma and Ba, 2015) is used. The Adam optimizer leverages the benefits of the AdaGrad and RMSProp optimizers and is generally considered the default optimizer to use in practice.

As is common for most categorical classification tasks, we use cross-entropy loss as the criterion of our model.

4.3 Data and Experimental Setup

We use BGL log data gathered from the loghub repository (Zhu et al., 2019) to evaluate our models. This data was generated by Lawrence Livermore National Labs through a BlueGene/L supercomputer system and has annotated alert and non-alert messages (Oliner and Stearley, 2007). At the case company, Ericsson, we evaluated the models on two logs, denoted in tables and figures as ELog A and ELog B, respectively.

The open-source log data had a tendency to repeat the same event many times, so we removed all repeating lines. This change was made as it may be difficult for the LSTM to learn the underlying sequential behavior of the log files when the input sequences do not exhibit this progression. We believe that this is a reasonable change to make, as this sort of preprocessing can be done even during runtime and the progression of events remains the same. This reduced the number of log entries from 4,747,963 to 660,810.

As the open-source data is in a single log format, we split the log in two by considering all events from the kernel as part of log A and all other events as part of log B. We believe that this split is reasonable as there is a clear difference in the source of both log A and log B. Of the 660,810 log entries, log A consists of 572,417 log entries, whereas log B consists of 88,393 log entries. In log A, 2.46% of log entries are anomalous, whereas 8.21% of log entries in log B are anomalous. While there is a clear imbalance in the number of log entries and anomalies per log, this is often the case in real, industry applications. Log files often operate at different levels of abstraction or operate on less used processes, so we believe this is common and representative of industry. It is important to mention that neither of these preprocessing steps were necessary for the industry log data at Ericsson, as log entries tended not to repeat frequently and there were multiple log files present in the original dataset, so no split had to be made.

For the early fusion match model, the first log was chosen as the base log. This was done to take advantage of the greater quantity of data present.

For each model, random partitions of test and train sequences were made. 80% of the data was used for training and 20% was used to test the model. For the Late Fusion log B predictor, the testing sets were determined by selecting all sequences of log entries which overlapped with sequences in the testing set of log A. The training set of log B was selected from all log sequences not added into the testing set. This means that the ratio between the length of the training and testing sets for log B were not always the same. All anomalous log sequences in the training sets were removed to ensure that the model was training on normal log data.

5 Evaluation

In Table 5.1, we outline the training and testing dataset splits for the open-source BlueGene/L log data. These were determined using a input sequence length of 2. For the Late Fusion Model predictor for Log B, the number of log entries used for training and testing differed for the reasons mentioned in the experimental setup.

To evaluate both the performance of each model in predicting the next log entry and in its ability to predict anomalies, we use four standard performance measures: Accuracy, Recall, Precision, and F1-Score. In standard binary classification problems, recall is the fraction of the number of true positives and the number of true positives plus the number of true negatives, this gives an indication of how complete the predictions of a model are. Precision is the fraction of the number of true positives and the number of true positives plus false positives, this gives an indication of how exact the model is in its predictions. The F1-Score is the harmonic mean between the precision and recall of a test.

Table 5.1: Number of sequences of length 2 in data sets

Model	Training Set	Test Set	Number of Log Keys
Early Fusion Match Model	457929	114483	4015
Early Fusion Merge Model	528643	132160	1299
Late Fusion Model Log A	457932	114483	570
Late Fusion Model Log B	-	-	729

As the prediction of a next log entry is a not a binary classification problem, but is instead a multiclass classification problem, we use the weighted variants of recall and precision. The weighted variants of recall and precision are a weighted average of each label's recall or precision, where the weight is determined by the frequency of the class in the dataset.

For the evaluation of each model regarding anomaly detection, the class bias between anomalous events and non-anomalous events will lead to inflated values for the performance measures. Therefore, we use the macro variants of recall and precision to find the unweighted mean between the classes. This should give a stronger indication of the performance of the model in detecting anomalies.

For the BlueGene/L dataset, all the models had the following properties. By default, the size of the hidden layer in the neural network model is 64, the size of each sequence is 2 entries long, and the number of epochs used to train the model is 20. For the anomaly detection section of our model, we use $k \approx 0.35 * nLogKeys$, where if the observed log event is in the top k predictions made by the

LSTM model, we don't consider the event anomalous. For the Late Fusion model, we use a decay rate of 0.8 and a threshold of 2 for detecting anomalies. All parameters were found by optimizing for model performance on randomly partitioned test sets.

For the Ericsson dataset, the size of the hidden layer in the neural network model is 512. The length of the sequences were also longer, at 10 entries long. We used 10 epochs to train the model. The same decay rate and threshold used for the BlueGene/L dataset were used.

The performance measures regarding how well each model is able to predict the next log entry (or, in the Early Fusion Match Model, a tuple of log entries) given a sequence are outlined in Table 5.2. The performance measures presented in Table 5.2 are the mean of 10 runs with different data partitions.

Table 5.2: Weighted performance measures (Accuracy, Recall, Precision, and F1-Score) for predicting the next log entry ± the standard deviation over 10 runs

Model	Accuracy	Recall	Precision	F1-Score
Early Fusion Match Model	0.911 ± 0.0014	0.911 ± 0.0014	0.938 ± 0.0009	0.924 ± 0.0011
Early Fusion Merge Model	0.902 ± 0.0008	0.902 ± 0.0008	0.919 ± 0.0011	0.910 ± 0.0009
Late Fusion Model Log A	0.909 ± 0.0011	0.909 ± 0.0011	0.921 ± 0.0013	0.915 ± 0.0011
Late Fusion Model Log B	0.776 ± 0.0427	0.776 ± 0.0427	0.853 ± 0.0250	0.813 ± 0.0349

As shown in Table 5.2, the performance of the early fusion models is quite strong. The early fusion match model outperforms the early fusion merge model marginally. The Late Fusion predictor for log A performs very well, but the log B predictor does not perform nearly as well as the other models. This is to be expected, as the log B predictor has far less data to train and test on than the other models.

Tables E.1 and E.2 in Appendix E contain information about the Ericsson log data and the performance of the models at predicting the next log entry given a sequence of log entries. The performance of the models is acceptable, achieving F1-Scores close to 0.8. It is also clear that the Early Fusion Match Model struggles to match the performance of the other models, as the number of possible input values grows exponentially with lower dependencies between the logs. As the industry data did not have labels, the anomaly detection performance is not quantitatively known. A basic qualitative description of our results is provided in the end of this section and in section 6. A more detailed description of the performance on the Ericsson log data can be found in Appendix E.

In Figure 5.1, we can see the performance of each of the models at detecting anomalies in the logs with growing k values. Both early fusion models achieve a macro F1-Score of around 0.95 and an accuracy over 99%. The early fusion merge model achieves slightly better performance than the early fusion match model but the difference is negligible, with Macro F1-Scores of 0.970 and 0.966, respectively. The late fusion model performs worse than the early fusion models in this case, peaking at a Macro F1-Score of 0.877.

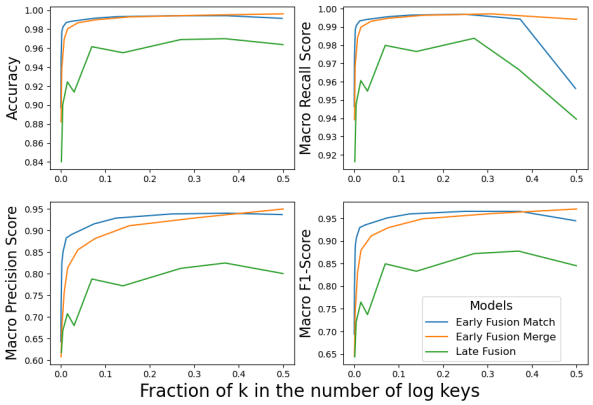

Figure 5.1: Anomaly Detection Performance of Early and Late Fusion Models

In Figure 5.2, we can see the influence of different sequence and hidden layer sizes on the performance of the models. We see that, despite the models struggling to retain information with very small hidden layers, a hidden layer size of 64 is sufficient for achieving strong performance, with no clear improvement when using higher dimensionalities. For the early fusion models, sequence length has no influence on the model performance. For the late fusion model, however, the performance of the models are much lower. The reason for this is because we partition the test and train datasets to ensure overlap between log A and log B test sequences in the late fusion models. With longer sequences, the log B test set sequences often contain log keys which were not present in the training set, leading to worse performance. It is important to note that we did not experience such a drop in performance with the Ericsson log data.

In Figure 5.3, the effects of decay and threshold

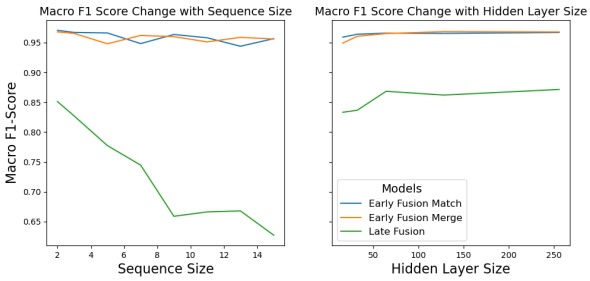

Figure 5.2: Anomaly Detection Performance of Early and Late Fusion Models with Differing Sequence Length and Hidden Layer Size

are shown on the late fusion models. It is clear that the performance of the model peaks at a threshold of 2, which is the base activation achieved for an anomaly detected by an individual predictor (after the forward and backward pass). For this dataset, there is rarely a similarity between the times anomalies occur between logs, therefore the late fusion approach was far less effective and could only leverage the results of the individual predictors. In the figure, higher decays performed better when the threshold was under 2 as this would ascribe less activation to events which were not detected as anomalous, but were close to an anomaly.

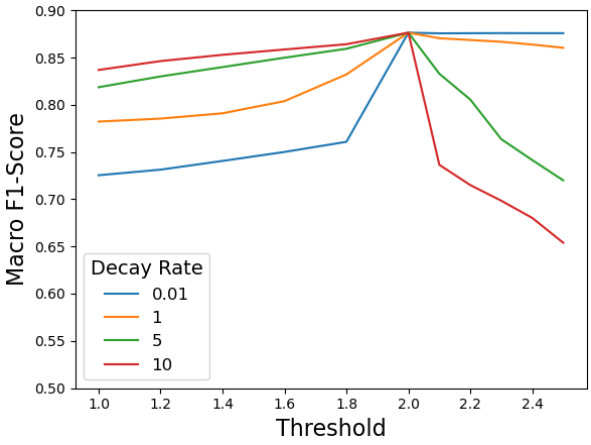

Figure 5.3: Anomaly Detection Performance of the Late Fusion Model with Differing Decay and Threshold Parameters. A high decay rate indicates lower time-based activations between nearby log events.

It is clear that the early fusion models outperform the late fusion models for this log data. As the original log data came from a single log, the time-based dependencies between the respective log entries are likely strong. In cases where log file data is more separate, the late fusion model may perform better. At the case company, we observed that the Late Fusion Model outperformed both Early Fusion Models when applied to log files with low entry-level dependencies over time. The Early Fusion Merge Model performed reasonably well but struggled to find cross-log anomalous events, whereas the Early Fusion Match Model did not perform nearly as well, mainly because with less dependency between the log files the dimensionality of the input grows dramatically. This growth means that there will be many inputs present in a test set not present in the training set, leading to worse performance.

6 Conclusion

Many companies develop and operate complex, software-intensive systems that produce log files of system behavior. In many cases, multiple log files are generated throughout the runtime of a system, each containing information at different levels of abstraction or about different subsystems.

Despite the extensive research which has been done in log file analysis and log file anomaly detection, there is a lack of existing approaches to effectively combine log files for improved results. In this paper, we outlined three generalizable strategies by which one could approach a multi-log file scenario.

We address the research question: **Does the performance of late fusion anomaly detection models outperform early fusion anomaly detection models when applied to multimodal system events gathered through log files?** We found that early fusion models perform better when there is a strong, causal dependency over time in the feature (event) level of the log. However, late fusion models may perform better when these dependencies are less strong, as was found in our evaluation of the Ericsson log data.

6.1 Future Work

While we achieved strong success in the early fusion approaches and moderate success in our late fusion approach, these approaches ignore the complex de-

pendencies and interactions between the systems underlying the log files. In addition to this, detecting system-wide anomalies could be improved using other performance measures present in the system. For instance, CPU or RAM usage, number or type of active connections, and other such performance measures could serve as additional parameters in models to improve their performance in this application.

Most importantly, there are many other possible combinations of log data possible. A hybrid fusion approach may serve better in more complex cases, where some logs are merged together at the event level, which are subsequently merged at the semantic level with other logs. The influence of the type predictors present in the models may also be investigated. For example, convolutional neural networks may also serve as more stable predictors than LSTMs and one-class classification methods (Tax, 2001) may serve as an interesting alternative to the existing approach. Future research will look into defining further approaches to detecting anomalies in a multi-log environment.

Acknowledgement

I would like to thank my academic supervisor, Prof Dr Lambert Schomaker, and my supervisor at Ericsson AB, Maria-Bianca Andersson, for their help and guidance with this bachelor's thesis.

References

Cristina Abad, Jed Taylor, Cigdem Sengul, William Yurcik, Yuanyuan Zhou, and Ken Rowe. Log correlation for intrusion detection: A proof of concept. In *Proceedings - Annual Computer Security Applications Conference, ACSAC*, 2003. ISBN 0769520413. doi: 10.1109/CSAC.2003.1254330.

Pradeep K. Atrey, M. Anwar Hossain, Abdulmotaleb El Saddik, and Mohan S. Kankanhalli. Multimodal fusion for multimedia analysis: A survey. *Multimedia Systems*, 2010. ISSN 09424962. doi: 10.1007/s00530-010-0182-0.

Yoshua Bengio, Patrice Simard, and Paolo Frasconi. Learning Long-Term Dependencies with Gradient Descent is Difficult. *IEEE Transactions on Neural Networks*, 1994. ISSN 19410093. doi: 10.1109/72.279181.

Andy Brown, Brian Hutchinson, Aaron Tuor, and Nicole Nichols. Recurrent neural network attention mechanisms for interpretable system log anomaly detection. In *Proceedings of the 1st Workshop on Machine Learning for Computing Systems, MLCS 2018 - In conjunction with HPDC*, 2018. ISBN 9781450358651. doi: 10.1145/3217871.3217872.

Marcello Cinque, Domenico Cotroneo, and Antonio Pecchia. Event logs for the analysis of software failures: A rule-based approach. *IEEE Transactions on Software Engineering*, 2013. ISSN 00985589. doi: 10.1109/TSE.2012.67.

Min Du, Feifei Li, Guineng Zheng, and Vivek Srikumar. DeepLog: Anomaly detection and diagnosis from system logs through deep learning. In *Proceedings of the ACM Conference on Computer and Communications Security*, 2017. ISBN 9781450349468. doi: 10.1145/3133956.3134015.

Qiang Fu, Jian Guang Lou, Yi Wang, and Jiang Li. Execution anomaly detection in distributed systems through unstructured log analysis. In *Proceedings - IEEE International Conference on Data Mining, ICDM*, 2009. ISBN 9780769538952. doi: 10.1109/ICDM.2009.60.

Sepp Hochreiter. Recurrent neural net learning and vanishing gradient. *Int. Journal of Uncertainity, Fuzziness and Knowledge-Based Systems*, 1998. ISSN 0899-7667. doi: 10.1142/S0218488598000094.

Sepp Hochreiter and Jürgen Schmidhuber. Long Short-Term Memory. *Neural Computation*, 1997. ISSN 08997667. doi: 10.1162/neco.1997.9.8.1735.

Zhanpei Jia, Chao Shen, Xiao Yi, Yufei Chen, Tianwen Yu, and Xiaohong Guan. Big-data analysis of multi-source logs for anomaly detection on network-based system. In *IEEE International Conference on Automation Science and Engineering*, 2018. ISBN 9781509067800. doi: 10.1109/COASE.2017.8256257.

Diederik P. Kingma and Jimmy Lei Ba. Adam: A method for stochastic optimization. In *3rd International Conference on Learning Representa-*

tions, ICLR 2015 - Conference Track Proceedings, 2015.

Aleksandar Lazarevic, Vipin Kumar, and Jaideep Srivastava. Intrusion detection: A survey. In *Managing Cyber Threats*, pages 19–78. Springer, 2005.

Jiazhong Lu, Fengmao Lv, Zhongliu Zhuo, Xiaosong Zhang, Xiaolei Liu, Teng Hu, and Wei Deng. Integrating traffics with network device logs for anomaly detection. *Security and Communication Networks*, 2019. ISSN 19390122. doi: 10.1155/2019/5695021.

Pankaj Malhotra, Lovekesh Vig, Gautam Shroff, and Puneet Agarwal. Long Short Term Memory networks for anomaly detection in time series. In *23rd European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning, ESANN 2015 - Proceedings*, 2015. ISBN 9782875870148.

Emmanuel Okafor and Lambertus Schomaker. Integrated dimensionality reduction and sequence prediction using lstm. 2018. URL `http://www.ictopen2015.nl/`. ICT.Open 2018, Amersfoort, The Netherlands; ICT.Open : The Interface for Dutch ICT-Research, ICT.Open ; Conference date: 24-03-2015 Through 25-03-2015.

Adam Oliner and Jon Stearley. What supercomputers say: A study of five system logs. In *Proceedings of the International Conference on Dependable Systems and Networks*, 2007. ISBN 0769528554. doi: 10.1109/DSN.2007.103.

Cees G M Snoek, Marcel Worring, and Arnold W M Smeulders. Early versus late fusion in semantic video analysis. In *Proceedings of the 13th ACM International Conference on Multimedia, MM 2005*, 2005. ISBN 1595930442. doi: 10.1145/1101149.1101236.

Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. Sequence to sequence learning with neural networks. In *Advances in Neural Information Processing Systems*, 2014.

David Martinus Johannes Tax. One-class classification; Concept-learning in the absence of counter-examples. *Delft University of Technology*, 2001. ISSN 00218979. doi: 10.1063/1.3605545.

Risto Vaarandi and Mauno Pihelgas. LogCluster - A data clustering and pattern mining algorithm for event logs. In *Proceedings of the 11th International Conference on Network and Service Management, CNSM 2015*, 2015. ISBN 9783901882777. doi: 10.1109/CNSM.2015.7367331.

Wil Van Der Aalst et al. Process mining manifesto. In *Lecture Notes in Business Information Processing*, volume 99 LNBIP, pages 169–194, 2012. ISBN 9783642281075. doi: 10.1007/978-3-642-28108-2_19.

A.J.M.M. Weijters, W M P van der Aalst, and A K Alves de Medeiros;. Process Mining with the HeuristicsMiner Algorithm, 2006.

Ronald J. Williams and David Zipser. Gradient-based learning algorithms for recurrent networks and their computational complexity. *Back-propagation: Theory, architectures and Applications*, 1995. doi: 10.1080/02673039508720837.

Ting Fang Yen, Alina Oprea, Kaan Onarlioglu, Todd Leetham, William Robertson, Ari Juels, and Engin Kirda. Beehive: Large-scale log analysis for detecting suspicious activity in enterprise networks. In *ACM International Conference Proceeding Series*, 2013. ISBN 9781450320153. doi: 10.1145/2523649.2523670.

Xiao Yu, Pallavi Joshi, Jianwu Xu, Guoliang Jin, Hui Zhang, and Guofei Jiang. CloudSeer: Workflow monitoring of cloud infrastructures via interleaved logs. In *International Conference on Architectural Support for Programming Languages and Operating Systems - ASPLOS*, 2016. ISBN 9781450340915. doi: 10.1145/2872362.2872407.

Jieming Zhu, Shilin He, Jinyang Liu, Pinjia He, Qi Xie, Zibin Zheng, and Michael R. Lyu. Tools and Benchmarks for Automated Log Parsing. In *Proceedings - 2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice, ICSE-SEIP 2019*, 2019. ISBN 9781728117607. doi: 10.1109/ICSE-SEIP.2019.00021.

A Appendix

Algorithm A.1 Calculates weights for each log event

Require: $decay > 0,\ g > 0,\ detectedTimes \neq emptylist,\ times \neq emptylist$
  $weights \Leftarrow emptylist$
  $memory \Leftarrow 0$
  $prevAnom \Leftarrow NULL$
  for all $time$ in $times$ **do**
    if $prevAnom \neq NULL$ **then**
      $timeDelta \Leftarrow abs(time - prevAnom[0])$
      $val \Leftarrow e^{-decay*timeDelta}$
    else
      $val \Leftarrow 0$
    end if
    $appended \Leftarrow$ **false**
    for $i \Leftarrow memory$ to $len(detectedTimes)$ **do**
      if $detectedTimes[i] = time$ **then**
        $prevAnom \Leftarrow (time, val + g)$
        $weights.append(val + g)$
        $memory \Leftarrow i$
        $appended \Leftarrow$ **true**
        break
      else if $time < detectedTimes[i]$ **then**
        $weights.append(val)$
        $appended \Leftarrow$ **true**
        break
      end if
    end for
    if $appended =$ **false then**
      $weights.append(val)$
    end if
  end for
  return $weights$

B Appendices

Algorithm B.1 Match Log Entries Preprocessing

Require: $logs \neq emptylist, baseLogIdx >= 0$
  $newLog \Leftarrow emptylist$
  for all $entry$ in $logs[baseLogIdx]$ **do**
    $newEntry = [entry[0]]$
    for $i \Leftarrow 0$ to $len(logs)$ **do**
      if $i = baseLogIdx$ **then**
        continue
      end if
      $ne, td \Leftarrow NULL$
      for all $entry2$ in $logs[i]$ **do**
        $timeDelta \Leftarrow abs(entry[1] - entry2[1])$
        if $td = NULL$ **or** $timeDelta < td$ **then**
          $td \Leftarrow timeDelta$
          $ne \Leftarrow entry2[0]$
        else
          break
        end if
      end for
      $newEntry.append(ne)$
    end for
    $newLog.append(newEntry)$
  end for
  return $newLog$

C Appendix

Algorithm C.1 Merge Logs into One Preprocessing

Require: $logs \neq emptylist$
  $logLengths \Leftarrow emptylist$
  $indices \Leftarrow emptylist$
  for $i \Leftarrow 0$ to $len(logs)$ **do**
    $logLengths.append(len(logs[i]) - 1)$
    $indices.append(0)$
  end for
  $newLog \Leftarrow emptylist$
  while $indices \neq logLengths$ **do**
    $lt, ce, wi \Leftarrow NULL$
    for $i \Leftarrow 0$ to $len(logs)$ **do**
      if $indices[i] = logLengths[i]$ **then**
        continue
      end if
      $entry \Leftarrow logs[i][indices[i]]$
      if $le = NULL$ **or** $entry[1] < le$ **then**
        $ce \Leftarrow entry[0]$
        $le \Leftarrow entry[1]$
        $wi \Leftarrow i$
      end if
    end for
    $indices[wi] \Leftarrow indices[wi] + 1$
    $newLog.append(ce)$
  end while
  return $newLog$

D Appendix

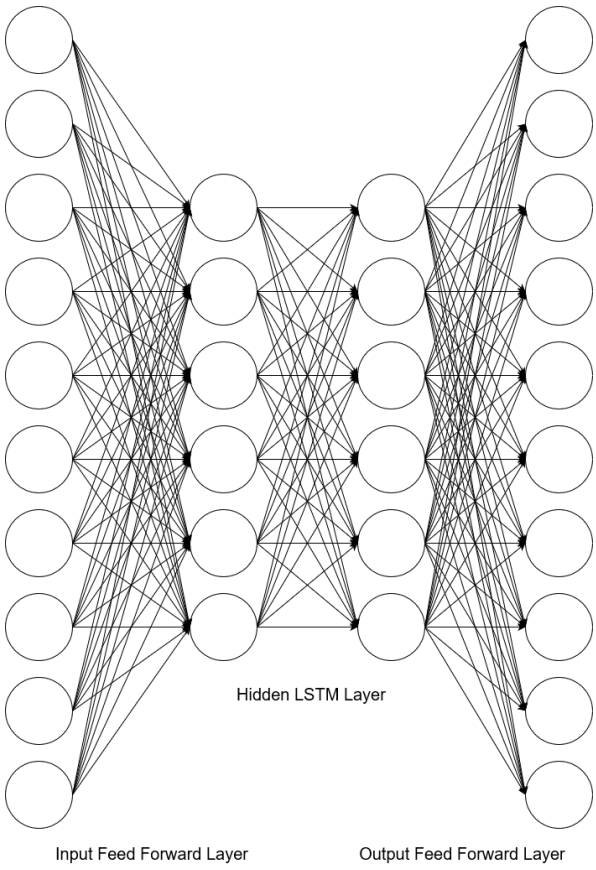

Figure D.1: The neural network model used to predict log events given a sequence of log events

E Appendix

Table E.1: Number of sequences of length 10 in the Ericsson log data set

Model	Training Set	Test Set	Number of Log Keys
Early Fusion Match Model	483845	162158	10543
Early Fusion Merge Model	2078600	913733	1614
Late Fusion Model ELog A	484034	201435	709
Late Fusion Model ELog B	1593874	711925	905

As shown in Table E.1, the number of log keys grows significantly in the Early Fusion Match Model. This is because the time-dependencies between the logs are not as strong, causing an explosion of possible log entry pairs between the logs.

Table E.2: Weighted performance measures (Accuracy, Recall, Precision, and F1-Score) for predicting the next log entry in the Ericsson log data

Model	Accuracy	Recall	Precision	F1-Score
Early Fusion Match Model	0.679	0.679	0.807	0.738
Early Fusion Merge Model	0.784	0.784	0.829	0.806
Late Fusion Model ELog A	0.782	0.782	0.806	0.794
Late Fusion Model ELog B	0.798	0.798	0.830	0.814

In Table E.2, the performance of the models for predicting the next log entry is shown on the Ericsson log data. While the performance on the data is not as strong as on the BlueGene/L dataset, this can, in large part, be attributed to the higher degree of complexity underlying the generation of the logs. This high degree of complexity is clear, as the models achieve the highest performance with a sequence length of 10 and a hidden layer size of 512. With better performance on higher sequence lengths, the longer-term dependencies in the logs are evident. Similarly, the large number of parameters in the hidden layer indicate that the underlying system behavior present in the logs is complicated.

As the Ericsson data set was not labelled, we cannot quantitatively report on the anomaly detection performance exhibited by the models on this data. As indicated by both Table E.2 and observations of model behavior, however, it is clear that the Early Fusion Match Model is not comparable to the other models unless there is a very strong relation between individual entries between the log files. The Early Fusion Merge Model and the Late Fusion Model initially performed equally well but, by tuning the decay and threshold parameters, the Late Fusion Model was found to perform better on the dataset.