Advancing Log Anomaly Detection by Deep Log Modeling

by

Yifei Lin

A thesis submitted in partial fulfillment of the requirements for the degree of

Master of Science

in

Software Engineering and Intelligent Systems

Department of Electrical and Computer Engineering

University of Alberta

©Yifei Lin, 2024

Abstract

Nowadays systems logs are crucial for ensuring the reliability and security of modern computer systems. Effective log anomaly detection is essential for identifying potential threats and maintaining system integrity. Many existing unsupervised methods depend on additional abnormal data for hyperparameter selection or auxiliary datasets for discriminative model optimization, limiting their practical application. Moreover, current log anomaly detection methods are often static, offline, and struggle with the dynamic and evolving nature of real-world environments. They require extensive preprocessing and frequent retraining, which is resource-intensive and inefficient in handling non-stationary data distributions. This thesis addresses the challenges of log anomaly detection with two novel approaches: FastLogAD and LogREAD, which focus on offline and online anomaly detection, respectively.

FastLogAD is designed for offline log sequence anomaly detection, emphasizing both speed and accuracy. By employing a Mask-Guided Anomaly Generator (MGAG) to produce pseudo-anomalies and a Discriminative Abnormality Separation (DAS) network to distinguish these from normal logs, FastLogAD effectively mitigates the dependency on additional training data. Experimental results on HDFS, BGL, and Thunderbird datasets demonstrate that FastLogAD not only achieves superior F1-Scores but also significantly enhances detection speed compared to existing methods.

Conversely, LogREAD focuses on online continual log instance anomaly detection, adapting to evolving data distributions without extensive preprocessing. This parsing-free method utilizes an adaptively reduced memory bank and a continuously evolved feature extractor to manage dynamic log patterns. Evaluations reveal that LogREAD performs comparably to,

or better than current offline methods and outperforms all online methods, showcasing its robustness in both static and dynamic environments.

Together, FastLogAD and LogREAD offer comprehensive solutions for diverse log anomaly detection scenarios. FastLogAD excels in high-throughput offline detection, while LogREAD provides adaptive real-time monitoring. This thesis contributes to advancing the field of log anomaly detection by addressing key challenges such as non-stationarity, data imbalance, and the need for rapid anomaly identification. Future developments in creating a generalized model adapting to a broad range of systems following our work would extend its applicability in real-world scenarios.

Preface

This thesis is an original work by Yifei Lin. No part of this thesis has been previously published. The work presented in Chapter 3, titled "FastLogAD: Log Anomaly Detection with Mask-Guided Pseudo Anomaly Generation and Discrimination," authored by Y. Lin, H. Deng, and X. Li, has been submitted to *IEEE Transactions on Knowledge and Data Engineering* and is currently under review. The work discussed in Chapter 4, authored by Y. Lin and X. Li, titled "LogREAD: Log-based Real-time Embedding for Anomaly Detection," is planned for submission to *The 39th Annual AAAI Conference on Artificial Intelligence (AAAI) 2025*.

Acknowledgements

I would like to express my gratitude to my supervisor, Professor Xingyu Li, for her invaluable guidance, support and encouragement throughout my research. Professor Li's insightful feedback, expertise, and patience have been instrumental in shaping this thesis and guiding my research.

I am also profoundly grateful to my family. Their unwavering support and unconditional love have been deeply rooted in my heart. Their immense sacrifices in raising me and their constant encouragement have shaped who I am today.

A special thanks to my girlfriend, Yan. Her belief in me has been a great source of strength and motivation during the challenging times of my research.

Thank you all for believing in me and for providing the necessary support to achieve this milestone. Without your support, this journey would not have been possible.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Overview and Motivation

Anomaly Detection is a crucial topic across various research areas and applications, targeting discovering data instances that deviate from the expected behaviours [6, 7]. In the era of big data, large-scale computer systems generate extensive logs that record system activities. These logs serve as a vital source of information for identifying anomalies such as system failures and performance issues. Failure to detect anomalies can lead to severe consequences, including system downtime, data breaches, and significant financial losses [11]. Thus, quickly and accurately identifying anomalies is paramount to ensure the smooth operation of computer systems and respond proactively to potential threats. Moreover, human-based anomaly methods require domain knowledge and extensive labour. As the sheer volume of log data is continually produced in a large-scale system, at an example general rate of 30 to 50 gigabytes per hour (approximately 100-120 million lines of logs) [46], data-driven approaches become a necessity to be leveraged to automate the process of detecting log anomalies.

Despite its importance, log anomaly detection poses several key challenges. Logs are often unstructured and highly heterogeneous text data [69], with different formats and structures across various systems and applications. This diversity hinders the design of a universal one-fits-all solution to log anomaly detection. In terms of data availability, the uneven ratio of normal and anomaly data introduces data imbalance. Supervised approaches therefore are less favourable to be considered. Additionally, the streaming update of log data demonstrates its dynamic nature where its normal behaviour can evolve over time. With this layer of complexity, traditional methods relying on keyword searches struggle to adapt to new normal logs due to keyword mismatches, leading to high false positive rates [36]. This non-stationarity is a significant challenge in real-world applications, as system logs can exhibit diverse and evolving patterns due to software updates, configuration changes, and

varying usage patterns. Continual learning is also required for the model to adapt new log patterns without forgetting previously learnt patterns. This is known as catastrophic forgetting [19, 44] and requires additional mechanisms to both retain historical knowledge and integrate new information [13]. It should also be noted that in the realm of system maintenance and security, the speed of log anomaly detection plays a pivotal role. As systems grow increasingly complex and voluminous, the ability to swiftly process and analyze logs at high throughput becomes critical. Fast detection not only ensures timely identification of potential issues, thereby reducing downtime and mitigating risks but also enhances the overall resilience of systems against emerging threats.

Existing approaches are proposed to resolve several aforementioned challenges. Most studies primarily focus on the unsupervised scenario where the training data consists only of normal instances and can be categorized into those based on discriminative tasks [47, 52, 66, 67, 70, 75] and log language generative tasks [15, 21, 28, 35, 45, 51]. The former directly optimizes a binary classifier for log anomaly detection and its fast inference speed is a significant advantage. However, with only task-specific normal data available for model training, previous methods often used extra data from other sources to act as abnormal logs for discriminator training [47, 66, 70]. However, these extra data may not be representative of the target domain, leading to degraded performance in model deployment. The latter based on log language generative models does not rely on additional data and achieves anomaly detection by modeling the sequential pattern of normal logs in model training. Such solutions usually first train a generative model to predict the next or masked entries in normal log sequences. Then, anomalies are detected by examining if the target log entry is in the top-K list predicted by the generative model. Due to the gap in the objective between the training and testing phases under the paradigm of log language generative modelling [51], extensive abnormal data are usually required for the optimal hyperparameter selection in down streaming anomaly detection strategies. In addition, these generative models, while powerful, tend to be complex and computationally intensive due to their inherent regression property, leading to slower inference speeds. On the other hand, to respond to the challenges of non-stationarity, online continual learning (OCL) methods have emerged as a promising solution [5,40,50,58] to continuously learn and adapt to real-time new data, thereby enhancing their ability to handle evolving data distributions without the need for manual intervention. In the domain of log anomaly detection, existing online approaches often fall short due to their reliance on extensive preprocessing and fixed model architectures. [13, 64]. The diversity of log formats poses challenges during parsing, potentially leading to parsing errors that hinder model performance [33]. Furthermore, fixed model architectures restrict the flexibility of online methods to adapt to new and unforeseen types of log data.

To address the current limitations discussed above, this thesis proposes FastLogAD for fast log anomaly detection and LogREAD for online continual learning scenarios. FastLogAD incorporates a Mask-Guided Anomaly Generator (MGAG) and a discriminative network for efficient anomaly detection. It introduces novel sampling strategies and employs a transformer-based generative model to create tailored pseudo-anomalies, enhancing its ability to separate anomalies from normal log sequences. The Discriminative Abnormality Separation (DAS) then leverages these pseudo-anomalies to train a robust discriminator, enabling efficient real-time detection without reliance on anomalous data during training. Experiments on datasets HDFS [68], BGL [49], and Thunderbird [49] demonstrate FastLogAD's superior performance, achieving the highest F1-scores and a significant speed increase in detection. LogREAD, on the other hand, is designed for online learning in dynamic environments. It starts with offline training on a small dataset using contrastive learning to capture semantic embeddings and maintains an adaptive memory bank of representative normal logs. During online operation, LogREAD compares incoming log embeddings with those in the memory bank, updating its memory and feature extractor as needed. This parsing-free approach reduces preprocessing overhead and adapts to evolving log patterns. Evaluations on the BGL [49], Thunderbird [49] and Spirit [49] dataset show that LogREAD outperforms existing methods in both offline and online scenarios, demonstrating robustness and adaptability. By combining FastLogAD's real-time capabilities with LogREAD's online learning approach, this thesis offers a comprehensive solution for the multifaceted challenges of log anomaly detection in modern computer systems.

1.2 Contributions

The novelty of our research lies in developing an offline unsupervised log anomaly detection model and an online continual log anomaly detection model. Specifically, our contributions are:

- To address slow inference speed and the need for auxiliary data in offline unsupervised log anomaly detection, we propose FastLogAD, featuring Mask-Guided Anomaly Generation (MGAG) and Discriminative Abnormality Separation (DAS). MGAG generates pseudo-anomalies, eliminating the need for auxiliary data for training. DAS separates normal and anomalous logs and is the only mechanism required for inference. We validate our approach on three common log datasets and achieve the best F1-Scores and inference speed.

- To reduce preprocessing overhead from log parsing and overcome the limitations of

fixed model architectures in existing online continual log anomaly detection methods, we introduce LogREAD. This parsing-free approach, featuring an adaptively reduced memory bank and evolving feature extractor, dynamically captures log patterns and adjusts to data distribution. Experimental results on three log datasets applicable to online continual learning (OCL) demonstrate LogREAD's superior performance over existing online methods and comparable or better performance than existing offline methods in offline settings.

1.3 Thesis Structure

This thesis is organized as follows:

Chapter 2 provides a general background in anomaly detection, followed by a specific introduction to log anomaly detection. It covers various log preprocessing techniques, including log parsing, and introduces several commonly used datasets. A review of current unsupervised log anomaly detection methods and online log anomaly detection methods is also discussed.

Chapter 3 details my paper, "FastLogAD: Log Anomaly Detection with Mask-Guided Pseudo Anomaly Generation and Discrimination". This chapter focuses on the methodologies, experiments, and contributions of FastLogAD to unsupervised log anomaly detection.

Chapter 4 delves into my paper, "LogREAD: Log-based Real-time Embedding for Anomaly Detection". This chapter focuses on online continual log anomaly detection, presenting the LogREAD methodology and its contributions.

Chapter 5 concludes the thesis and outlines our insights for future work.

Chapter 2

Background and Literature Review

2.1 Anomaly Detection

2.1.1 Definition and Importance

Anomalies, also known as abnormalities, deviants or outliers in statistical analysis, are data points that deviate significantly from the majority of observations [1]. Figure 2.1 illustrates this concept, where N_1 and N_2 represent the majority of normal observations, while O_1, O_2, and the small region O_3 are anomalies. These anomalies can indicate critical incidents or errors within a dataset. The process of identifying these anomalies is known as anomaly detection [1, 6]. It has been widely applied in real-world applications, including:

- Identifying unusual patterns in network traffic to detect cyber-attacks or intrusions [3].
- Identifying irregularities in financial transactions, which can indicate fraudulent activities [2].
- Detecting abnormal patterns in medical records to identify potential health crises [16].
- Detecting anomalies in computer logs to identify system performance issues, potential bottlenecks, or failures [32, 34].

Failing to detect anomalies can have severe consequences. In healthcare, it can lead to missed diagnoses, delayed treatment, and potentially life-threatening situations. In computer systems, undetected anomalies can cause system failures, service disruptions, and financial losses. Therefore, quick and accurate anomaly detection is essential for maintaining system integrity and reliability.

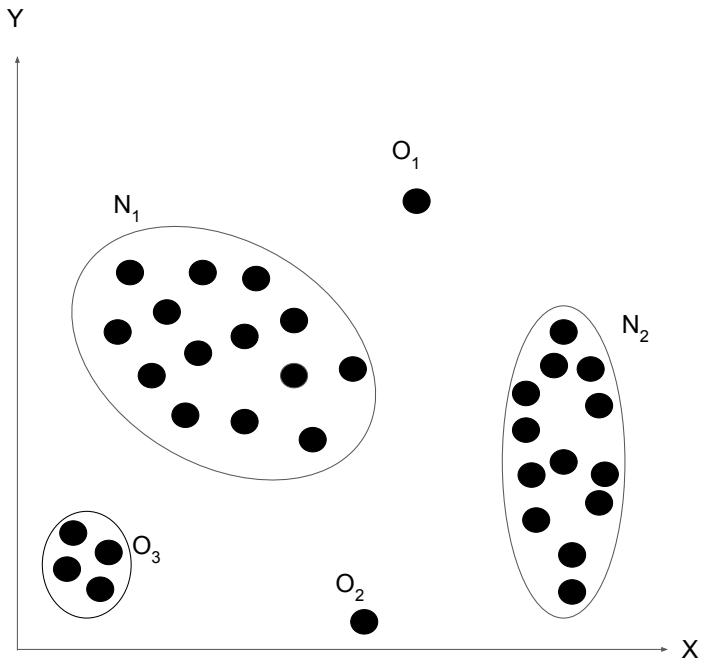

Figure 2.1: An example of anomalies presented in a two-dimensional dataset.

2.1.2 Types of Anomalies

Anomalies can be broadly classified into three categories [7]: point anomalies, contextual anomalies and collective anomalies. Understanding the type of target anomaly is essential for designing effective anomaly detection methods. In Section 2.2, we will associate log anomaly detection tasks with their corresponding anomaly types in more detail.

- **Point Anomalies.** If an individual instance deviates from the rest of the data and presents to be anomalous, then this instance is considered a point anomaly. The major anomaly detection work focuses on detecting point anomalies, including log instance anomaly detection.

- **Contextual Anomalies.** An instance is termed a contextual anomaly if it can be considered anomalous under certain contexts, otherwise, it is normal. This type of anomaly is also known as conditional anomaly [57] and a real-world example can be found in time series data showing the monthly temperatures. A sudden low temperature reaching 35°**F** is normal in winter but is considered an anomaly in summer.

- **Collective Anomalies.** If a group of data instances appears to be anomalous with respect to the entire dataset, then it is known as a collective anomaly. Each isolated

point from this collection appears as a normal instance but exhibits unusual patterns when viewed together. An example of a collective anomaly can be found in credit card transactions. Suppose multiple transactions occur simultaneously from different geographic locations using the same credit card. Individually, each transaction might appear legitimate, but collectively, these transactions could indicate a compromised card being used fraudulently in multiple locations at the same time.

2.1.3 Running Modes

- **Supervised Anomaly Detection.** When operating an anomaly detection method in supervised mode, the training data has labelled instances for both normal and anomaly classes.

- **Fully Unsupervised Anomaly Detection.** With the assumption that normal instances appear more frequently than anomalies in the test data, fully unsupervised anomaly detection methods do not require training data. Their capability of detecting anomalies is solely based on the intrinsic properties of the data instances. Failing the implicit assumption could result in high false positive rates.

- **Semi-supervised or Unsupervised Anomaly Detection.** Due to the scarcity of abnormal instances for training, semi-supervised anomaly detection methods operate with only normal instances as training data. Since they do not require instances for the anomaly class, they are more widely applicable than supervised techniques. Nowadays the terms semi-supervised and unsupervised are used interchangeably in the domain of image [39] and log anomaly detection [32]. In this thesis, our approach is technically semi-supervised but aligns closely with what is commonly described as unsupervised in the literature [32, 39].

Unsupervised anomaly detection methods are preferred in the literature [18] to obtain a robust model. On the other hand, supervised models are generally limited by the known anomalies from training. While acknowledging that our methods may also be referred to as semi-supervised, we consistently use the term "unsupervised" throughout this work to maintain clarity and consistency with the broader literature.

2.1.4 Evaluation Metrics

Evaluating the performance of anomaly detection models requires specific metrics. Accuracy alone can be misleading in imbalanced datasets, as predicting all instances as normal can

result in high accuracy. The following metrics are more informative, where TP, FP, and FN denote True Positive, False Positive, and False Negative, respectively:

- **Precision:** The ratio of true anomaly predictions to the total predicted anomalies. It mainly measures the accuracy of the positive predictions.

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

- **Recall:** The ratio of true anomaly predictions to the total actual anomalies. It measures the ability of a model to identify all the anomalies.

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

- **F1-Score:** The harmonic mean of precision and recall, a single metric that balances both.

$$\text{F1-Score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

- **Area Under the Receiver Operating Characteristic Curve (AUROC):** A single scalar value that calculates the area under the curve of True Positive Rate (TPR) against False Positive Rate (FPR) at various thresholds. It indicates the model's ability to distinguish between classes when the costs of false positives and false negatives are similar.

$$\text{TPR} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

$$\text{FPR} = \frac{\text{FP}}{\text{FP} + \text{TN}}$$

- **Area Under the Precision-Recall Curve (AUPR):** A similar single scalar value to AUROC, except that the curve is by plotting Precision against Recall across various thresholds. This is useful for imbalanced datasets where the number of positive instances is much smaller than the number of negative instances, such as anomaly detection tasks.

2.2 Log Anomaly Detection

Log anomaly detection [26] is a crucial task of maintaining and ensuring the reliability and performance of software systems. Logs, generated by various system components, capture detailed information about system states, events, and activities. Anomalies in these logs

can indicate potential issues, such as system failures, security breaches, or performance bottlenecks. Given the volume and complexity of log data, automated methods for detecting anomalies are essential for timely and accurate identification.

Log anomaly detection can be categorized into log sequence anomaly detection and log instance anomaly detection. They are mainly different in the anomaly type and supported datasets. Log sequence anomaly detection focuses on identifying anomalies within a sequence of log entries. It examines the order and structure of log events to detect abnormal patterns or sequences that deviate from the expected behavior. The anomaly type is considered collective where a sequence of log entries collectively appears anomalous compared to the expected pattern. Log instance anomaly detection in contrast treats each log entry independently, making it well-suited for identifying isolated abnormal events. Thus, the anomaly type in this task is point anomaly, where each log is evaluated on abnormality.

In the subsequent subsections, we will explore the main components of a log anomaly detection framework, including log data processing techniques and several representative log anomaly detection methods. We will also discuss advancements and challenges specific to this field, providing a transition to the later chapters on our proposed methods.

2.2.1 Log Processing

Each log entry records a specific system activity, containing multiple fields such as timestamp, severity level, logger name and the actual log event. These fields are often separated by delimiters like white spaces and colons. To favour downstream anomaly detection tasks, it is crucial to process raw log data by extracting the most relevant information and features for model input.

Log Parsing

Log parsing aims to structure raw logs by extracting event templates. Specifically, each log message can be parsed into an event template (constant part) and some parameter fields (variable parts). Figure 2.2 shows an example of log messages from the HDFS dataset parsed into log events and a list of extracted parameters for each message. Although this thesis does not focus on log parsing, it is a vital component of log sequence anomaly detection. Most approaches [15, 21, 45, 73] use indices to represent the unique log event that each log entry corresponds to.

The current log parsing methods can be generally classified into four categories [74]:

- **Clustering:** By the name, the clustering approach mainly clusters the log messages into a set of K clusters based on the string matching. For each cluster, a log template

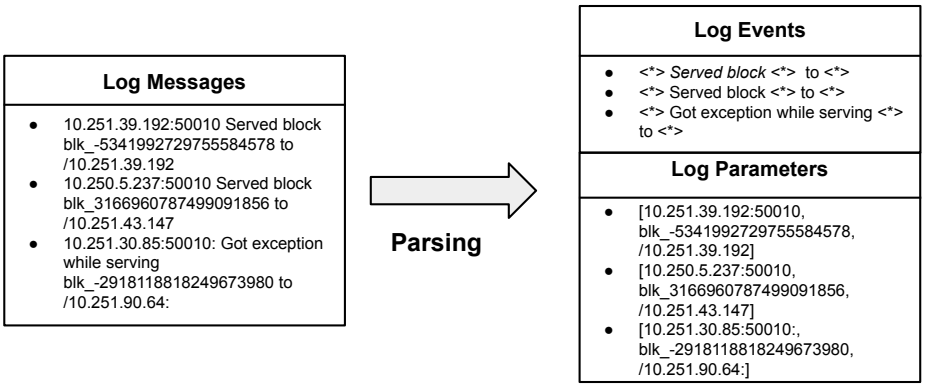

Figure 2.2: The process of transforming raw log messages from the HDFS dataset [68] into structured log events and parameters. The log messages are initially parsed into log events, where the constant parts are identified as event templates and the variable parts are extracted as parameters.

represents the log pattern for all entries. LogMine [23] selects the first message in each initial cluster as its common log pattern which forms the initial hierarchical level. It then iteratively merges the patterns by relaxing the distances until the hierarchy is fully established. In addition to hierarchical clustering, other clustering methods such as K-Means and density-based clustering are also applied in log parsing [29, 59].

- **Frequent pattern mining:** This method groups logs sharing the same patterns by counting words from a frequent word set. SLCT [60] builds the foundation of this category. It iterates the whole dataset with three passes, creating a frequent word set, building clusters by matching the frequent words and extracting patterns for the logs in the same cluster.

- **Heuristics:** This type of log parsing method provides the best overall parsing accuracy among all the categories. Spell [14] uses a longest common subsequence (LCS) algorithm to parse log entries. It maintains a map of parsed log entries with their LCS sequences. For each new log entry, Spell searches for the existing entry with the longest matching LCS sequence. If the match is significant, it updates the sequence. Otherwise, it creates a new entry. Drain [25] is the most widely employed parsing method in log anomaly detection methods due to its high and robust parsing accuracy across different datasets. It uses a fixed-depth parsing tree to cluster raw log messages. Each leaf node represents a log group, and the tree structure limits the number of groups a new log message must traverse. The top layer of nodes matches messages by length and the subsequent layers match by the first few tokens. When a log message reaches a leaf node, it is assigned to the group with the highest per-token similarity.

- **Program analysis:** The parsing methods [68, 71] in this category usually rely on regular expressions and need access to the source code, which might be unavailable due to licensing issues. Therefore, they become less practical than the aforementioned data-driven approaches.

As mentioned above, Drain is a solid option for log parsing in the domain of log anomaly detection. In our first paper on FastLogAD, we adopt this parser following existing approaches. In the second paper on LogREAD, we remove the parsing step to reduce processing overhead and prevent parsing errors for real-time online anomaly detection.

Log Sequence Grouping

Log sequence anomaly detection operates on sequences of log data, but datasets typically consist of individual log entries. To address this, log sequence grouping techniques are used to group log entries into meaningful sequences. There are three common methods for grouping log sequences:

- **Fixed window:** This method groups logs within a fixed time interval or a fixed number of logs. The logs in contiguous windows are non-overlapping, ensuring no contextual dependency between them. It is suitable for datasets without a clear grouping mechanism but may not accurately reflect real-world situations. For instance, datasets with labels for each log might not be appropriate for grouping when log instance anomaly detection is preferred.

- **Sliding window:** An extension of the fixed window approach, this method introduces an overlap between contiguous windows, controlled by a step size in time or number of logs. While it offers additional connections between log sequences, it shares the fixed window method's limitations and requires more computational resources due to the increased number of windows.

- **Session window:** This method is applicable to log datasets with an identifier parameter in each log entry, representing the group ID or process thread. It provides the most accurate grouping and reflects real-world scenarios where a sequence of log entries indicates an activity's progress. However, datasets suitable for this strategy are rare, forcing the use of the fixed and sliding window methods on the rest of the datasets for broader comparison.

2.2.2 Log Datasets

To compare our methods to the existing approaches, we select four public datasets [76] in total, namely HDFS [68], BGL [49], Thunderbird [49], and Spirit [49]. The details of each dataset are as follows:

- **HDFS (Hadoop Distributed File System)** [68]: This dataset contains 11,175,629 log messages collected from the Amazon EC2 platform. Each log has an identifier ID that records block operations such as allocation, writing, replication, and deletion. This identifier makes the logs naturally appropriate for grouping into session windows and log sequence anomaly detection. We evaluate our offline method, FastLogAD, on this dataset.

- **BGL (Blue Gene/L)** [49]: This dataset contains system logs generated by the Blue Gene/L supercomputer at Lawrence Livermore National Labs (LLNL), with 4,747,963 logs, including 348,460 manually labelled anomalies. Unlike HDFS, BGL logs do not have identifiers for each block of activities, so we apply sliding window methods for FastLogAD comparison. For the online log instance anomaly detection approach, LogREAD, no windowing method is used as BGL provides labels for each log.

- **Thunderbird** [49]: Collected from the Thunderbird supercomputer at Sandia National Labs (SNL), this large dataset has over 200 million log lines, usually subsetted for experimental evaluations. It has a format similar to BGL, so we use the sliding window method for FastLogAD comparison and keep it intact for LogREAD comparison.

- **Spirit** [49]: This dataset was collected from a Spirit supercomputer at Sandia National Labs (SNL). It consists of over 270 million logs that capture detailed operational behaviors. Only a subset is used for experimental comparison. Spirit is not included as a benchmark in FastLogAD comparisons since we already use the above three datasets but is experimented on in LogREAD since the HDFS dataset, with only group anomaly labels, cannot support log instance anomaly detection.

2.2.3 Methods

Log anomaly detection methods can be classified into unsupervised and supervised approaches, similar to general anomaly detection techniques. This section focuses primarily on unsupervised approaches, further categorized according to the backbone model each approach is based on. Additionally, some online log anomaly detection methods will be introduced.

Traditional Machine Learning Methods

Figure 2.3 illustrates an overview flowchart of traditional log sequence anomaly detection methods. These methods require a fixed-sized input feature vector, typically achieved by converting the parsed log events within a sequence into a count vector that records the frequency of each log event in the sequence. This approach is lightweight due to the small model complexity. However, using a count vector loses the sequential dependencies between the log events, as the order of events in the sequence is not captured. Additionally, it suffers from the Out-Of-Vocabulary (OOV) issue [33]. During inference, if an unknown log event appears in the sequence due to software updates, the count vector cannot include it, potentially causing a potential decline in performance.

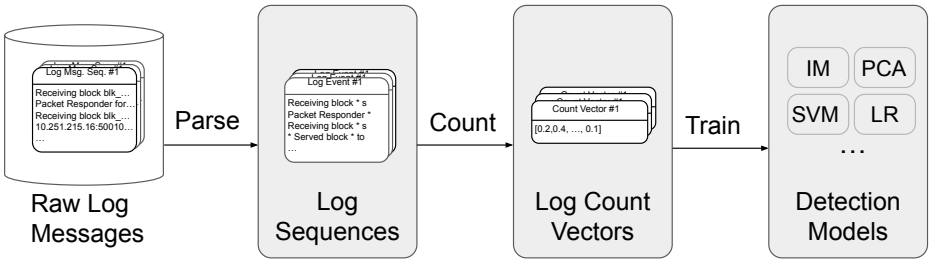

Figure 2.3: Overview of traditional log sequence anomaly detection methods [75]. The process begins with raw log messages, which are parsed into log sequences to identify log events. These sequences are then converted into log count vectors, which serve as features for training various detection models.

- **PCA** [68]: Employing the property of dimensionality reduction of PCA, the first k principal components of the training data are used to construct a normal subspace, while the remaining $(n-k)$ components create an abnormal subspace. A testing sample is identified as an anomaly if the projection of its event count vector onto the abnormal subspace has a large magnitude beyond a preset threshold. PCA is sensitive to outliers, which can skew principal components.

- **Invariant Mining** [42]: This method detects anomalies by identifying and validating invariant relationships between log events. Invariants are constant linear relationships that always hold true in normal system executions. First, this approach applies Singular Value Decomposition (SVD) to the event count matrix, where each row represents an event count vector of a log sequence. Invariants are then found using a brute-force algorithm to search for relationships that contain k non-zero coefficients. If a new log

message breaks a known invariant, it indicates an anomaly in the system. Though this method provides meaningful interpretations of the specific variants that an anomaly breaks, the process of invariant mining is time-consuming, and the computational cost increases with the number of non-zero coefficients in the invariant vectors.

- **LogCluster** [36]: As the name suggests, LogCluster is a clustering-based method that generates sets of clusters containing log event count vectors. The centroid of each cluster is selected as the representative vector for that cluster. During inference, the distance between the count vector of a new log sequence and each representative vector is computed. The log sequence is identified as an anomaly if the distances from all the clusters exceed a preset threshold. This method also adopts an online mechanism that allows new count vectors to be added to the existing cluster with the shortest distance or assigned to a new cluster. Nonetheless, we do not include this method in the comparison with our LogREAD method for online log anomaly detection. Our focus in LogREAD is online log instance anomaly detection whereas LogCluster focuses on sequence anomaly detection and cannot be converted to an instance anomaly detection version.

Deep Learning Methods

Deep learning methods have emerged to address the limitations of traditional log anomaly detection approaches. Instead of modelling log sequences into fixed-sized vectors, deep learning methods offer flexible input lengths that more accurately describe the temporal patterns of each sequence. Additionally, most deep learning-based methods use sequences of tokens, where each token represents a unique log event parsed from each log entry within the sequence. This step is known as tokenization. Compared to using a log count vector, this representation excels in capturing the order of log events within in a sequence. However, it still faces the limitation of the Out-Of-Vocabulary (OOV) [33], where unknown log events are assigned to the same index representing the unknown event, impeding the model's ability to leverage knowledge transfer.

Deep learning methods primarily rely on two backbones: Recurrent Neural Networks (RNN) [27] and transformers [61]. RNN-based methods have the advantage of fast processing speed than transformers for short sequences but struggle with long sequences, even with enhancements Long Short-Term Memory [20] and Gated recurrent units (GRUs) [8]. In addition, they have to break down long sequences into several shorter ones to sustain good performance. The model is biased towards the most recent tokens, and the tokens that appear early would practically have no effect on the output for a long sequence. Transformer

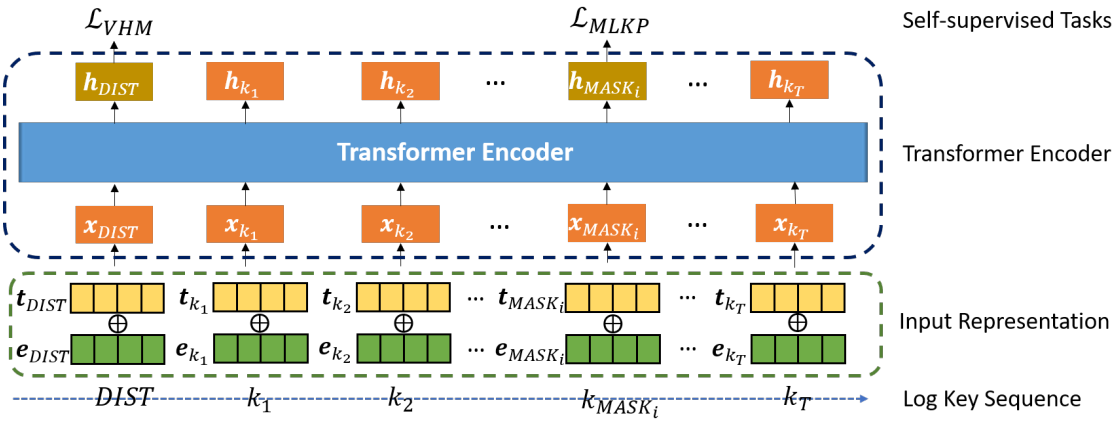

Figure 2.4: An overview of LogBERT, $\mathcal{L}_{MLKP}$ and $\mathcal{L}_{VHM}$ denote the loss functions of Masked Log Key Prediction (MLKP) and Volume of Hypersphere Minimization (VHM).

based-models, on the other hand, leave no room for information loss due to the self-attention mechanism [61]. They process a sequence in parallel to ensure no loss of past information, reducing the performance drop when processing long sequential data. This parallel computation also ensures fast training and inference, even though transformer models naively have a high computational complexity of $O(T^2)$ compared to $O(T)$ for RNN-based models where T stands for the input length. In recent years, Graph Neural Network (GNN) approaches have gained attention in log anomaly detection. They treat each sequence as a separate graph and leverage predictions on the whole graph. Despite being innovative, GNN-based approaches rely on their graph design, where the connections between log events require domain knowledge. This is not as effective as transformer-based approaches that learn dependencies through training.

Current deep learning approaches can be classified into two categories based on their training techniques: **generative** and **discriminative** models.

For **Log language generative models**, the model generates a log event token as the prediction for the next token following the given sequence or a masked token.

DeepLog [15] is considered the pioneer model in deep learning approaches. It leverages an LSTM [20] model to predict the next token following the given sequence. This method regards each log sequence as tokens representing the log events. If the ground true next token is not in the model's top-k prediction, then the given sequence is identified as an anomaly.

LogAnomaly [45] further incorporates semantic and quantitative information from log sequences as an RNN-based approach. Instead of encoding the log sequence into tokens, it leverages a distributional lexical-contrast embedding model (dLCE) [48] to represent log event templates with embedding vectors. Compared to tokenization, this embedding method can generate new embeddings for unknown log events based on the preserved information.

LogAnomaly also includes log event count vectors as additional information. Despite extracting a significant amount of information, LogAnomaly does not avoid the shortage of RNN-based methods compared to transformer-based models as discussed above.

LogBERT [21] stands out as an iconic transformer-based model designed to capture long sequence dependencies between tokens. It adopts Bidirectional Encoder Representations from Transformers (BERT) [12] as the backbone and optimizes the model with two self-supervised tasks: Masked Log Key Prediction (MLKP) and Volume of Hypersphere Minimization (VHM), as shown in the Figure 2.4. MLKP unleashes BERT's potential by randomly masking several input tokens with a pre-determined masking ratio and training the model to predict the original tokens. VHM task as an auxiliary is inspired by Deep SVDD [54]. Its objective operates on the embeddings of the log sequences, generated from the prepended [CLS] token. This ensures a data-enclosing hypersphere to regulate the distribution of normal log sequences. During inference, LogBERT determines anomalous log keys by leveraging top-k predictions on randomly masked tokens. The entire sequence is considered an anomaly if the number of anomalous log keys exceeds a threshold. Though LogBERT achieves better performance than RNN-based methods, it requires manual tuning of two threshold hyperparameters with the presence of additional abnormal data.

Glad-PAW [62] is the first GNN-based approach in log anomaly detection. It creates a single graph for each sequence. The vertices of the graph represent the log events and the edges represent their sequential execution order. A top-k prediction on the next token is applied to identify anomalies.

Despite the various designs in generative backbones and learning targets, during inference, these methods usually define a normal sequence based on the predicted output falling within a specified range of possible k logs. Nonetheless, the limitation of these approaches is prominent: The value of hyperparameter k cannot be easily selected unless abnormal data is provided. In addition, these generative models, while powerful, tend to be complex and computationally intensive due to their inherent regression property, leading to slower inference speeds.

Discriminative models, on the other hand, directly optimize the target model with a binary classification-style objective. Some of these models process an entire log sequence in parallel, enabling faster inference than log generative models. This type of model also excels in detection performance as training is often supervised, or unsupervised with extra data to simulate the abnormal behaviours.

LogRobust [75] is a log sequence anomaly detection model based on Bidirectional LSTM (Bi-LSTM). It leverages the off-the-shelf algorithm FastText [4] to represent each word with a semantically meaningful vector. TF-IDF [56] is adopted to weight aggregate the word

vectors of each log message to obtain one single vector representation. Each log sequence is fed to the Bi-LSTM as a list of vectors representing each log event. This supervised approach uses a binary classification head on the top of the Bi-LSTM to determine the normality of the sequence based on the bidirectional hidden representations.

Logsy [47] works on log instance anomaly detection. This parsing-free method treats each word in a log message as an input token to a transformer model, training unsupervised with only normal log instances and using auxiliary data from non-target datasets as pseudo-abnormal instances. Although the auxiliary data is easily accessible from the internet, it may deviate from true abnormal data of the target dataset domain, such as the difference in the token words. The transformer model is trained to separate the normal and abnormal logs by the distance between the norm of the transformer's [CLS] output embedding. This method inspires our FastLogAD on log sequence anomaly detection by generating more domain-related pseudo-abnormal data on its own during training.

LogGD [67] leverages Graph Transformer [72] to perform log sequence-level detection through graph classification as a supervised method. In addition to the semantic embedding of each log event, LogGD incorporates three structural attributes of the graph, the degree matrix, the distance matrix, and the edge weight matrix, to generate structure-aware encoding to enhance the discrimination of graph representation. The embedding of the entire graph representing each sequence is eventually read out from a concatenation of node representations. Despite its sophisticated design, the performance does not seem to reflect this, and it does not overcome the limitations of GNN-based methods.

PLELog [70] is a unique log sequence anomaly detection method trained on labelled normal data and unlabelled data including anomalies. It utilizes clustering to group log sequences by their semantic meanings, assigning probabilistic labels to unlabelled sequences for supervised anomaly detection on a GRU network [8]. While this approach relaxes the condition of using labelled anomaly to train a discriminative model, it assumes the presence of anomalous instances in the unlabelled data, which may not always be realistic. Additionally, using an RNN-based network still struggles with capturing dependencies in long sequences.

A2Log [66] is similar to Logsy [47] but focuses on log sequence anomaly detection instead. It creates pseudo-abnormal sequences by randomly masking tokens, using these sequences for validation to determine the decision boundary. However, it still requires non-target domain data as abnormal data during training, facing the same limitation as Logsy.

From the existing discriminative approaches, the limitation is clear: they often require auxiliary abnormal data from the target domain or use non-target domain data as anomalies. The former as a supervised approach lowers the difficulty of training and the practicality of collecting a proper number of abnormal instances. The latter is conducted as an unsuper-

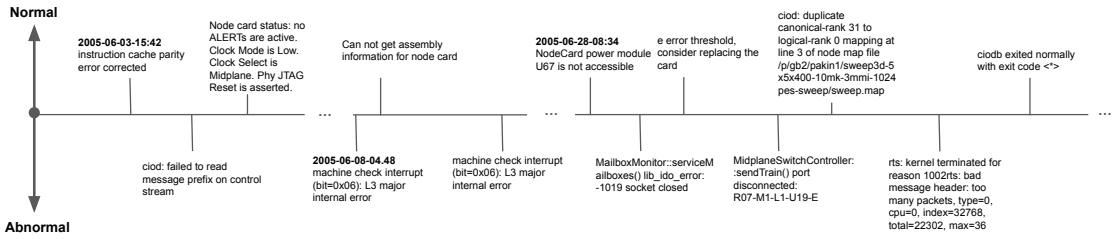

Figure 2.5: The timeline illustrates the dynamic and varying formats and lengths of logs in the BGL [49] dataset, highlighting the need for online log anomaly detection

vised method, their auxiliary data from the non-target dataset may not represent the target domain, leading to degraded performance in model deployment. By comparing both generative and discriminative models, we are inspired to propose FastLogAD, which maintains superior performance and fast inference speed as a discriminative model while alleviating the need to collect auxiliary data.

Online Methods

One issue with offline models applying to the industry is that they suffer from non-stationary data drift. This challenge is equally relevant to log anomaly detection methods. Offline methods are less practical in industrial settings due to the need for frequent retraining from scratch to maintain effectiveness with continuously incoming log data. As shown in Figure 2.5, logs can exhibit varying lengths and formats over time, regardless of their ground truth labels. This variability underscores the need for online log anomaly detection methods, which can adapt to these changes more effectively than offline methods. In addition, previously learnt log information such as log events or semantic information needs to be retained by the model. This contrasts with traditional time series models, which often do not incorporate large amounts of historical information. Our primary goal for log anomaly detection is to maintain and update knowledge from both past and new data while mitigating the issue of catastrophic forgetting, where the model forgets previously learned information when new data is introduced. This combined mechanism is known as online continual learning (OCL), which enhances pure online learning with continual learning's knowledge retention capabilities.

Robust Online Evolving Anomaly Detection (ROEAD) [24] offers an online version of SVM dealing with log sequence anomaly detection. Similar to traditional log anomaly detection methods, it uses event count vectors from each sequence as input. These vectors are multiplied with a feature weight vector to create a linear hyperplane that separates normal

and abnormal log sequences. By applying the Karush–Kuhn–Tucker (KKT) condition [31], the feature weight vector is adjusted according to the prediction result of each arriving log sequence. While this method provides a lightweight solution for enabling the online mechanism, it shares the same limitation as other traditional machine learning methods for log anomaly detection: the inability to record new log events. We omit ROEAD in comparison with our LogREAD because ROEAD is a log sequence anomaly detection method and cannot be easily adapted for log instance anomaly detection.

UnLearn [13] offers an online approach for log sequence anomaly detection. Built based on DeepLog [15], this approach transitions from top-k to probability prediction of the next log event. A sequence is flagged as an anomaly if its predicted probability falls below a given threshold. During online updates, an unlearning strategy is employed: the model is optimized to increase the training loss on instances with false negative predictions and to increase this loss for false positives. An L2 regularization is also applied to ensure the model's weights do not deviate significantly, thereby preventing performance degradation or catastrophic forgetting [44]. Despite its promise, the fixed prediction head limits the model's ability to incorporate new log events as prediction targets, resulting in a performance decline similar to offline RNN-based methods. By adapting UnLearn for log instance anomaly detection, we compare it with our LogREAD.

LogOnline [64] improves LSTM-based methods for online log sequence anomaly detection by allowing the prediction head to expand and include newly arrived log events. This approach adopts an auxiliary autoencoder to support online learning. The autoencoder is trained offline with the normal log headers, and log sequences with a reconstruction error below a predetermined threshold are classified as normal. During online learning, the identified normal log sequences by the frozen autoencoder are used for the incremental updating of the main LSTM model. The advantage of LogOnline is it does not require manual feedback or data labels for online updates, significantly reducing labour. However, LogOnline relies on the strong performance of the autoencoder, which is not continually trained during the online stage. Additionally, accurate log parsing is required to extract log headers, limiting the applicability of this approach to datasets with such log headers.

Existing online methods for log anomaly detection are designed to successfully implement incremental updates to the model with online log streams. However, several limitations still exist. Parsing is a mandatory processing step that induces more computational overhead. Fixed architectures, such as a fixed prediction head including a limited number of log events in UnLearn [13] and a frozen autoencoder in LogOnline [64], hinder the model from adapting to new log information. This is contrary to the idea of online continual learning. These current limitations inspire us to propose LogREAD, a parsing-free online log instance anomaly

detection method that incorporates a dynamic memory bank and a flexible feature extractor tuning mechanism.

Chapter 3

FastLogAD: Offline Log Sequence Anomaly Detection

3.1 Introduction

In this chapter, we present our approach to offline log sequence anomaly detection, FastLogAD. As modern computer systems become more complex and produce increasingly large volumes of data, the necessity for rapid and efficient log processing has become paramount. The timely detection of anomalies within these logs is critical to minimizing system downtime, preventing security breaches, and maintaining overall system integrity. Therefore, optimizing log anomaly detection for speed is not merely a technical challenge but a strategic imperative crucial to the reliability and security of contemporary computing environments. Beyond achieving high accuracy, our approach, FastLogAD, specifically targets the need for high-throughput log anomaly detection without sacrificing precision.

Current deep learning-based log anomaly detection methods can be categorized into discriminative models [47, 52, 66, 67, 70, 75] and log language generative models [15, 21, 28, 35, 45, 51]. Discriminative models, which directly train a binary classifier for anomaly detection, offer the advantage of rapid inference. However, they typically rely on supplementary data to simulate anomalous logs during training, which may not always accurately represent the target environment, leading to potential performance issues. In contrast, log language generative models focus on learning the sequential patterns of normal logs to identify deviations. While these models do not require additional data for training, they are often complex and computationally intensive, resulting in slower inference speeds.

To address these limitations, we introduce FastLogAD for fast log anomaly detection. Our model comprises two main components: the Mask-Guided Anomaly Generator (MGAG) and

the Discriminative Abnormality Separation (DAS) network. MGAG, built on a transformer-based architecture, captures the sequential patterns in normal logs and employs two novel sampling strategies—random replacing and masked token replacing—to generate pseudo-anomalies that deviate from these patterns. DAS is then trained to distinguish between normal and pseudo-anomalous sequences by leveraging the large discrepancy in output norms. This method enables efficient anomaly detection by setting thresholds based on the output norms of normal data, even without exposure to actual anomalies during training.

We validate FastLogAD on several benchmark datasets, including HDFS [68], BGL [49], and Thunderbird [49], demonstrating that our approach not only achieves superior F1-scores compared to existing methods but also significantly enhances detection speed by at least a factor of ten.

The contribution of this study can be summarized in the following points:

- We propose FastLogAD, a novel approach for unsupervised log anomaly detection characterized by Mask-Guided Anomaly Generation for generating anomalies and by Discriminative Abnormality Separation for discriminating anomalies.
- We introduce and analyze two strategies for synthesizing pseudo-anomalies: Random Generation and Masked Language Modeling generation.
- We propose Replaced Token Detection and Hyperspherical Separation Training to train an anomaly discriminator and a compact hyperspherical boundary of normal features.
- We evaluate the proposed FastLogAD on three common, yet real anomaly detection datasets. The experiments show that FastLogAD outperforms other methods and achieves real-time log anomaly detection.

3.2 Methodology

Problem formulation. Let $\boldsymbol{s} = \{s_1, s_2, \ldots, s_d\}$ be a sequence of d log events parsed from raw log messages, following the processing steps as shown in Figure 3.1. Our goal is to train a model with training data $\mathcal{D}_{train} = \{(\boldsymbol{s}_1, y_1 = 0), (\boldsymbol{s}_2, y_2 = 0), \ldots, (\boldsymbol{s}_N, y_N = 0)\}$, each $y_i = 0$ represents training set consisting of only normal sequences under the unsupervised anomaly detection setting. In the subsequent detection stage, the model is able to distinguish between normal and abnormal sequences in the test set $\mathcal{D}_{test} = \{(\boldsymbol{s}_{N+1}, y_{N+1}), \ldots, (\boldsymbol{s}_{N+M}, y_{N+M})\}$ with a total of M samples.

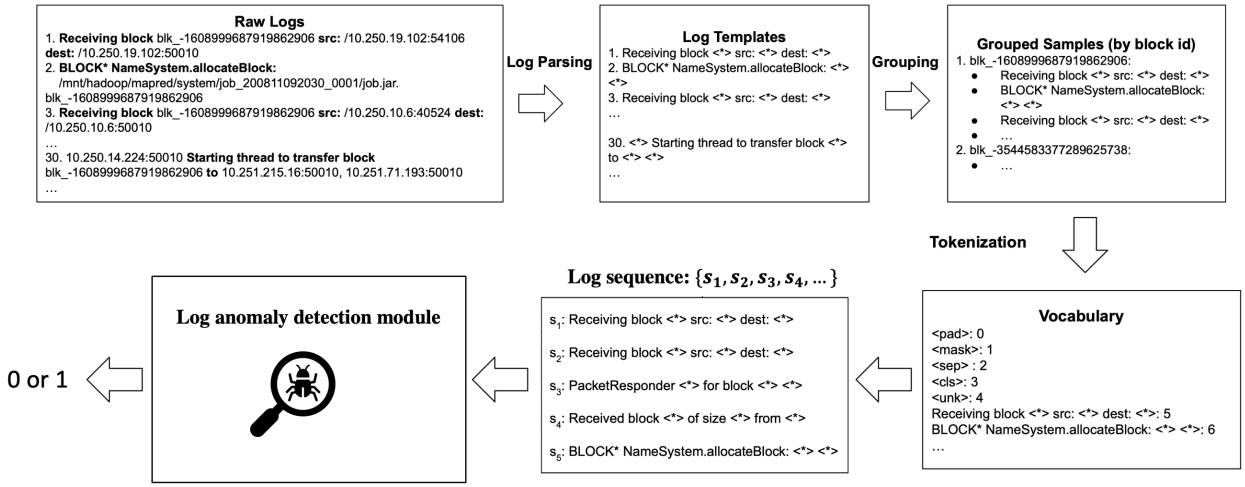

Figure 3.1: Illustration of the complete pipeline of the proposed log anomaly detection solution. Taking logs from the Hadoop Distributed File System (HDFS) dataset [68] as examples, log templates are extracted through log parsing and grouped into sequences based on the identifier block ids. A vocabulary is created to map the log events and special tokens (e.g., [cls], [mask]) to their corresponding unique indices during model training. Then the normal log training data is passed to the log anomaly detection module for model optimization. During inference, the vocabulary is static and used to construct query log sequences after log parsing and grouping. The log anomaly detection model provides a yes/no answer.

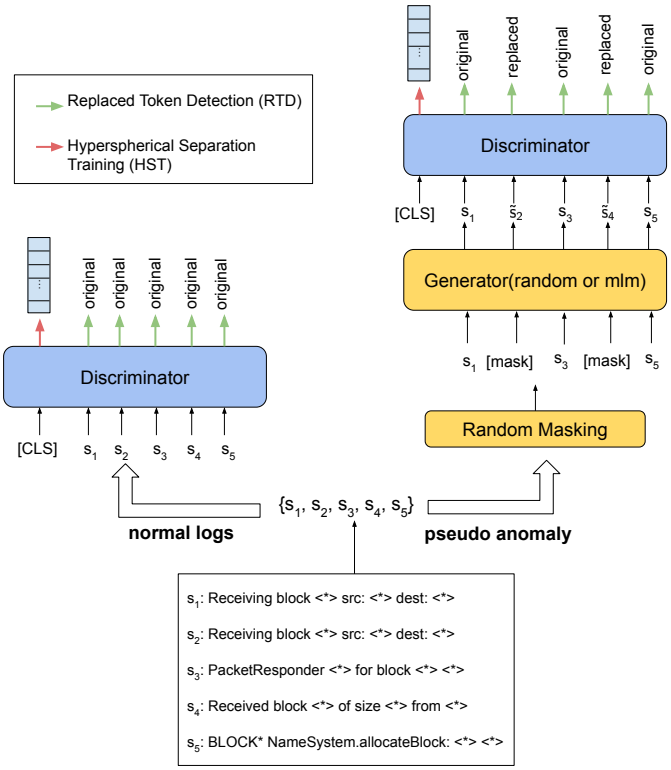

Figure 3.2: The training procedure of FastLogAD. For a given sequence of normal logs, we randomly mask the log tokens in a certain ratio, and then generate the corresponding log sequence through a generator. For the discriminator, we propose RTD and HST to learn to distinguish normal logs from pseudo-anomaly logs.

Overview of FastLogAD model. Figure 3.2 and Figure 3.3 provide an overview of FastLogAD's training and inference stage, respectively. FastLogAD utilizes a generator-discriminator architecture that features Masked-Guided Anomaly Generation and Discriminative Abnormality Separation tasks. In this architecture, only the discriminator corresponding to Discriminative Abnormality Separation task is employed during inference. The generator's role is to generate pseudo-abnormal samples for each normal sequence during training. Two variants of the generator are introduced: the training-free random generator and the masked language model requiring training. The discriminator undergoes initial adversarial training to distinguish between normal and abnormal tokens replaced by the generator within a sequence, focusing on token-level recognition. Ultimately, it learns the output embedding to make normal and anomalous log sequences separable from a sequence-level recognition, following a similar approach presented in Logsy [47].

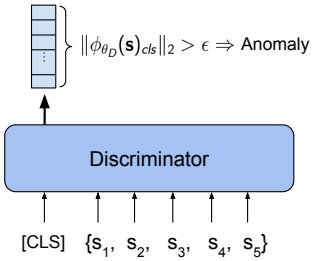

Figure 3.3: The inference procedure of FastLogAD. In inference, we directly use the anomaly discriminator for efficient diagnosis of logs.

3.2.1 Mask-Guided Anomaly Generation

The Mask-Guided Anomaly Generation (MGAG) task in FastLogAD's framework serves to craft pseudo-abnormal log sequences for the discriminator's training. Given a normal log event sequence $\boldsymbol{s} = \{s_1, s_2, \ldots, s_d\}$ from the training set, the tokens in the sequence undergo a random masking process with a masking ratio r, indicating what percentage of tokens gets replaced with $[mask]$ token. The resulting masking pattern $\boldsymbol{m}$ holding binary values indicates which tokens are replaced with the $[mask]$ token, and $\hat{\boldsymbol{s}} = \{\hat{s}_1, \hat{s}_2, \ldots, \hat{s}_d\}$ denotes the masked sequence.

$$\hat{s}_i = \begin{cases} s_i & m_i = 0 \\ [mask] & m_i = 1 \end{cases}. \tag{3.1}$$

By feeding the masked sequence to the generator, a pseudo-abnormal log sequence is constructed by replacing the masked tokens with random or unlikely tokens presented in a normal sequence. With this procedure, each training sample comes with corresponding abnormal versions of itself. Two variants of the generator are introduced in the following and using one of them can fulfill the MGAG task.

Random Generator

As we create a vocabulary of tokens that represent the log events from the training set, the idea of the random generator is to replace the masked token with a randomly sampled token from the vocabulary. While this simple approach does not undergo any training process thereby reducing the computational overhead, it may fall short in accurately introducing the anomalous property of the whole sequence. This limitation arises because the random generator does not learn the dependencies among tokens within a sequence.

Specifically, Let $\mathbf{V}$ represent the vocabulary, then the sequence generated by the random

generator is denoted as $\tilde{\boldsymbol{s}}^{rand} = \{\tilde{s}_1^{rand}, \tilde{s}_2^{rand}, \ldots, \tilde{s}_d^{rand}\}$, where

$$\tilde{s}_i^{rand} = \begin{cases} s_i & m_i = 0 \\ S \sim \mathbf{U}(\mathbf{V} \setminus \{s_i\}) & m_i = 1 \end{cases}. \tag{3.2}$$

Each $\tilde{s}_i^{rand}$ of masked input is sampled from a uniform distribution over the vocabulary $\mathbf{V}$ excluding s_i, the ground truth token from the original sequence.

MLM Generator

In contrast to the random generator, the Masked Language Model (MLM) generator tends to replace masked tokens with those less likely to appear in the given sequence, crafting more realistic abnormal sequences. This generator exhibits improved performance, especially on datasets with a smaller vocabulary. However, this variant requires a training process and its effectiveness may diminish as the vocabulary size increases as shown in the following experimental results.

Mathematically, the Masked Language Model (MLM) generator uses a BERT with a classification head, outputting a probability distribution P_G over the vocabulary $\mathbf{V}$. Unlike the random generator outputting a uniform distribution, this generator is trained using MLM loss:

$$\mathcal{L}_{MLM} = -\mathbb{E}\left[\sum_{\substack{i=1 \\ m_i=1}}^{d} \log P_G(s_i \mid \hat{\boldsymbol{s}}; \theta_G)\right]. \tag{3.3}$$

Although trained with this loss ensures the generator assigns higher probabilities to tokens likely to appear in the original normal sequence, our strategy for its output is to sample the tokens from the *complement* distribution $\bar{P}_G$ such that for each $s_i \in \mathbf{V}$,

$$\bar{P}_G(s_i \mid \hat{\boldsymbol{s}}; \theta_G) = \frac{1 - P_G(s_i \mid \hat{\boldsymbol{s}}; \theta_G)}{\sum_{j=1}^{|\mathbf{V}|}(1 - P_G(s_j \mid \hat{\boldsymbol{s}}; \theta_G))}, \tag{3.4}$$

where for each token the probability is normalized over the sum. Thus, the generated sequence by the MLM generator can be denoted as $\tilde{\boldsymbol{s}}^{mlm} = \{\tilde{s}_1^{mlm}, \tilde{s}_2^{mlm}, \ldots, \tilde{s}_d^{mlm}\}$, where

$$\tilde{s}_{mlm,i} = \begin{cases} s_i & m_i = 0 \\ S \sim \bar{P}_G(s \mid \hat{\boldsymbol{s}}; \theta_G) & m_i = 1 \end{cases}. \tag{3.5}$$

3.2.2 Discriminative Abnormality Separation

The log sequence is prepended with a [CLS] token before being fed to the discriminator. Such that, the input sequence is then $\tilde{\boldsymbol{s}} = \{\tilde{s}_{cls}, \tilde{s}_1, \tilde{s}_2, \dots \tilde{s}_d\}$ with $\{\tilde{s}_1, \tilde{s}_2, \dots, \tilde{s}_d\}$ being the given normal sequence with label $y = 0$ or the pseudo-abnormal version from the generator with label $y = 1$. In this Discriminative Abnormality Separation (DAS) task, we adopt a BERT [12] model as the discriminator for achieving anomaly detection through two separate training tasks: Replaced Token Detection (RTD) and Hyperspherical Separation Training (HST).

Replaced Token Detection (RTD)

Similar to the MLM generator, the discriminator follows a BERT model structure connected to a binary classification for each token in the input sequence except for the added [CLS] token. This classification head corresponds to the first training task, RTD being previously adopted effectively in ELECTRA's naive training [9] and its application DATE [43] of anomaly detection in NLP. In our approach, we apply this training as the warmup step for our discriminator to classify the replaced and non-replaced tokens for understanding the potential token-level anomalies before bringing them up to the overall anomaly detection on the whole sequence. Therefore, the training objective of RTD is to minimize the Binary Cross Entropy (BCE) loss across the training samples:

$$\mathcal{L}_{RTD} = -\mathbb{E}\left[\sum_{i=1}^{d} \log P_D(m_i \mid \tilde{\boldsymbol{s}}; \theta_D)\right] \tag{3.6}$$

Hyperspherical Separation Training (HST)

In one-class anomaly detection modelling, we want the normal features tightly clustered within the latent space. To this end, we adopt a hyperspherical function in Logsy [47] to separate normal and pseudo-abnormal sequences by controlling the norm of the feature embedding corresponding to the prepended [CLS] token to the discriminator. Let $f_D(\tilde{\boldsymbol{s}})_{cls}$ denote this feature embedding, the loss of training this objective, namely Hyperspherical Separation Training (HST), is

$$\begin{aligned}\mathcal{L}_{HST} =& \mathbb{E}\Big[(1-y)\|f_D(\tilde{\boldsymbol{s}})_{cls}\|_2 \\ & - \lambda y \log(1 - \exp(-\|f_D(\tilde{\boldsymbol{s}})_{cls}\|_2))\Big], \lambda > 0.\end{aligned} \tag{3.7}$$

The first term enables normal samples, with the feature embedding norm minimized to be close to the origin. In contrast, the anomalous samples are enforced away from the origin as presented in the second term, penalizing the small embedding norm. This also prevents convergence to trivial solutions [55], where the parameters of the networks are zeros to minimize the first term only. λ as a positive weight controls the emphasis between the two terms.

Algorithm 1 Two-Step Training of FastLogAD

Require: $\mathcal{D}_{train}$, Training epochs N and M, Discriminator f_D, Generator f_G

Step 1: Training MLM Generator and RTD
for epoch = 1 to N **do**
  for $\boldsymbol{s}$ in Batch($\mathcal{D}_{train}$) **do**
    $\hat{\boldsymbol{s}} \leftarrow \text{Mask}(\boldsymbol{s})$ ▷ Mask input
    $\tilde{\boldsymbol{s}} \leftarrow f_G(\hat{\boldsymbol{s}})$ ▷ Pseudo-abnormal samples
    if using MLM generator **then**
      Compute $\mathcal{L}_{MLM}(\tilde{\boldsymbol{s}})$
    end if
    Compute $\mathcal{L}_{RTD}(f_D(\tilde{\boldsymbol{s}}))$
    if using MLM generator **then**
      Optimize $\mathcal{L}_{MLM} + \mathcal{L}_{RTD}$
    else
      Optimize $\mathcal{L}_{RTD}$
    end if
  end for
end for
Step 2: HST Training
Freeze f_G
for epoch = 1 to M **do**
  for $\boldsymbol{s}$ in Batch($\mathcal{D}_{train}$) **do**
    $\hat{\boldsymbol{s}} \leftarrow \text{Mask}(\boldsymbol{s})$ ▷ Mask input
    $\tilde{\boldsymbol{s}} \leftarrow f_G(\hat{\boldsymbol{s}})$ ▷ Pseudo-abnormal samples
    Compute and optimize $\mathcal{L}_{HST}(f_D([\boldsymbol{s}, \tilde{\boldsymbol{s}}]))$
  end for
end for
return f_D

3.2.3 Overall Training

Our goal is to eventually detect anomalies based on the feature embedding norm of the [CLS] token, as presented in the following section. Thus, we decide to employ a two-step training for our discriminator and generator (or training-free random generator), and this whole training procedure is provided by Algorithm 1.

- Training the MLM generator together with the discriminator's RTD task for the first N epochs to start up: $\mathcal{L}_{MLM} + \mathcal{L}_{RTD}$ or $\mathcal{L}_{RTD}$ for the random generator.
- M epochs for HST to promote accurate anomaly scores in the detection stage: $\mathcal{L}_{HST}$

This two-step training approach offers several advantages compared to joint training and separate training of all tasks:

1. The joint training of RTD and MLM tasks for N epochs reduces training complexity, recognizing both tasks as auxiliary and not directly used in the subsequent detection stage. This streamlined approach avoids unnecessary computational overhead associated with separate training.
2. RTD task is trained before HST to give the discriminator attention to the crafted abnormal tokens first for the subsequent learning of the sequence-level recognition. If joint training of RTD and HST is employed, the discriminator might overfit the crafted tokens instead of considering anomalies as entire sequences. Log anomalies are typically determined by the sequences they are part of, rather than individual log events. For instance, the HDFS dataset provides anomaly labels for entire grouped sequences rather than assigning a label to each log. Thus, RTD is exclusively trained before HST to guide the warm-up of training and later we show RTD training does help with our anomaly detection task.

3.2.4 Detection

As the role of the generator is to craft the pseudo-anomalies for the training step, the anomaly detection for the incoming log sequences then only requires the discriminator to be performed. As shown in Algorithm 2, each log sequence $\boldsymbol{s}$ combined with a leading [CLS] token is directly fed to the discriminator. The discriminator evaluates each log sequence $\boldsymbol{s}$ based on the feature embedding norm $\|f_D(\boldsymbol{s})_{cls}\|_2$ as the anomaly score. Sequences are deemed abnormal if the score exceeds a threshold ϵ, determined using a validation set comprising normal sequences.

3.3 Experiment

3.3.1 Experimental Settings

Datasets

We use HDFS [68], BGL [49] and Thunderbird [49] datasets for experimental comparisons. For all datasets, a chronological order is maintained during the train-test split, ensuring that

Algorithm 2 FastLogAD Detection Stage

Require: $\mathcal{D}_{test}$, threshold ϵ, Trained discriminator f_D

$\boldsymbol{y}_{pred} \leftarrow [\,]$ ▷ Initialize the prediction list
for $\boldsymbol{s}$ in $\mathcal{D}_{test}$ **do**
    $y_{temp} \leftarrow \|f_D(\boldsymbol{s})_{cls}\|_2$
    if $y_{temp} > \epsilon$ **then**
        Append 1 to $\boldsymbol{y}_{pred}$
    else
        Append 0 to $\boldsymbol{y}_{pred}$
    end if
end for
return $\boldsymbol{y}_{pred}$

normal sequences are split without shuffling. Specifically, we follow the experiment setting in LogBERT [21] and take around the first 5000 log sequences following the given timestamp as training data to reflect practical scenarios. Subsequently, 10% of the split training data is set aside for validation, and all abnormal sequences are included in the test set for evaluation.

Baselines

Due to the limited available public implementations [34] of unsupervised log anomaly detection with deep learning, we compare our FastLogAD to log anomaly detection baseline models including DeepLog [15], LogAnomaly [45] and LogBERT [21].

Evaluation Metrics

Following the existing works, we adopt Precision, Recall and F1-Score as evaluation metrics:

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}};\ \text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}};\ \text{F1-Score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

TP: True Positive; FP: False Positive; FN: False Negative;

Experimental Setup

We conduct our experiments on a Linux Server with Intel i9-9940X @ 3.30GHz CPU and 64GB of RAM. One Nvidia RTX 2080Ti GPU is utilized for training under the Python 3.8.10 environment with Pytorch 2.1.0+cu118 installed.

Implementation Details

For FastLogAD, we take $r = 0.5$ mask ratio, $\lambda = 1$ for HST loss. The detection threshold ϵ is determined by taking the 99% quantile of embedding norms on the validation dataset to exclude the outliers. We adopt the architecture of ELECTRA [9] to construct both the MLM generator and discriminator for both variants. The single embedding layer is configured with a dimension of 256, followed by 4 transformer layers. Each transformer layer is equipped with 4 heads, and the feedforward layer has a dimension of 256. More detailed parameters can be found in our public implementations. For the baselines, DeepLog [15] and LogAnomaly [45] do not have their official implementations available to the public, therefore we experimented with others' replicated versions[1] instead. All the involved hyperparameters are selected according to the adopted implementations and the detection thresholds of these three baselines: k presented in DeepLog [15] and LogAnomaly [45] and anomalous tokens ratio r in LogBERT [21] are selected according to their best performance on the test set since these thresholds need to be determined with abnormal sequences involved. While this provides a natural advantage for these baselines in comparing performances to FastLogAD, the impracticality of selecting thresholds should be considered.

3.3.2 Results

Table 3.1 shows the experimental results of the baselines and FastLogAD with both two generator variants. The results are based on running with 3 random seeds and all metric values are expressed as percentages (%). FastLogAD outperforms the compared baselines in terms of F1-Score. Specifically, FastLogAD with the MLM generator achieves the best F1-Score across three datasets, while the random generator variant achieves the second best, with a slight lag behind the MLM generator approach on Thunderbird. This slight lag is caused by the large vocabulary size ($\approx$1000) of the training data in which training MLM is nearly equivalent to training a random generator by eventually sampling from a large number of candidates. In an overall comparison with the best-performing baselines on the three datasets, FastLogAD exhibits a 6.54% and 4.29% advantage in F1-Scores over DeepLog for HDFS and BGL, respectively. Concerning the Thunderbird dataset, FastLogAD achieves a 1.98% higher F1-Score than LogBERT.

[1] `https://github.com/donglee-afar/logdeep` and `https://github.com/LogIntelligence/LogADEmpirical`

HDFS			
Methods	Precision (%)	Recall (%)	F1 (%)
DeepLog [15]	79.37±4.24	92.06±0.28	85.23±3.67
LogAnomaly [45]	**93.46±2.67**	59.20±4.55	72.48±5.08
LogBERT [21]	85.20±0.58	75.39±1.29	80.00±0.68
FastLogAD-Random	83.56±0.75	99.51±0.12	90.84±0.40
FastLogAD-MLM	84.80±0.60	**99.99±0.01**	**91.77±0.36**
BGL			
Methods	Precision (%)	Recall (%)	F1 (%)
DeepLog [15]	86.89±1.89	90.91±4.13	88.85±2.51
LogAnomaly [45]	78.03±1.51	86.70 ±3.25	82.14±2.18
LogBERT [21]	74.79±0.96	91.68±1.45	82.38±0.78
FastLogAD-Random	82.72±3.14	98.02±0.95	89.72±1.49
FastLogAD-MLM	**88.28±3.58**	**98.66±0.86**	**93.14±1.66**
Thunderbird			
Methods	Precision (%)	Recall (%)	F1 (%)
DeepLog [15]	90.25±1.71	98.57±1.05	94.23±1.29
LogAnomaly [45]	79.48±0.00	99.55±0.00	88.39±0.00
LogBERT [21]	**96.27±0.40**	95.03±2.23	95.65±1.14
FastLogAD-Random	94.88±0.12	**99.95±0.03**	97.34±0.02
FastLogAD-MLM	95.45±0.14	99.91±0.08	**97.63±0.03**

Table 3.1: Experimental results (mean±std) on HDFS, BGL and Thunderbird datasets, the best metric values are **bolded** and the runner up is highlighted with undeline. Note that numerical results of the compared baselines are obtained with the best hyperparameters selected manually.

Method	HDFS		BGL		Thunderbird	
	Total (s)	Avg. (ms)	Total (s)	Avg. (ms)	Total (s)	Avg. (ms)
DeepLog [15]	46.85	0.08	177.91	6.89	1647.97	14.14
LogAnomaly [45]	83.42	0.15	320.28	12.42	2953.37	25.34
LogBERT [21]	791.12	1.39	132.05	5.11	936.40	8.04
FastLogAD-MLM	77.32	0.14	37.21	1.44	105.58	0.90

Table 3.2: Total inference time and the average inference time per log sequence.

Inference Time Analysis

As we discussed in the Introduction, the speed of log anomaly detection plays a pivotal role in the realm of system maintenance and security. So in this experiment, we evaluate the inference time of FastLogAD and other competitors to demonstrate the effectiveness of our proposed method. As shown in Table 3.2, we present the total inference time and average inference time per log sequence for the compared models across three datasets. LogBERT [21] achieves slower inference on the HDFS dataset, which is composed of short sequences. This is caused by the random masking step and multiple predictions performed on each sequence. DeepLog [15] and LogAnomaly [45] have the best efficiency with HDFS dataset, of which the log sequences are relatively shorter than the other two datasets. Their inference speed noticeably deteriorates with long sequences where each sequence needs to be further decomposed into multiple short sequences to maintain the performance. In contrast, FastLogAD showcases satisfactory overall inference speed across datasets. Its practicality for real-time inference is noteworthy, as it involves only the discriminator, requiring no additional operations on the input sequence. This characteristic makes FastLogAD suitable for both long and short-sequence datasets, presenting a distinct advantage over prior arts.

Distribution of Anomaly Score

Fig. 3.4 displays the distribution plot of anomaly scores for the three experimental datasets. The anomaly scores of normal and abnormal data are depicted in distinct colours, showing that almost exclusively normal data is concentrated near the origin. Moreover, a significant portion of abnormal data has an anomaly score exceeding 10. This substantial separation ensures a distinct separation between normal and abnormal data, contributing to FastLogAD-MLM's competitive performance. Notably, this achievement is noteworthy as the anomaly threshold is selected without prior exposure to abnormal data.

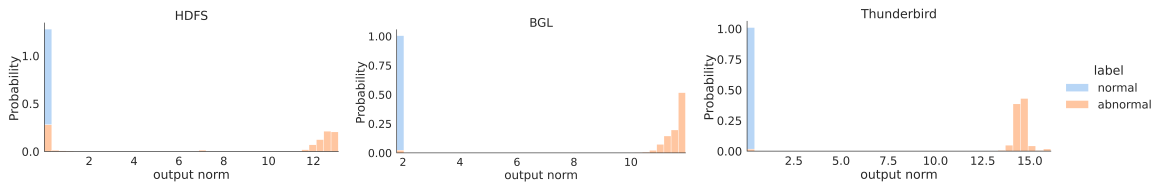

Figure 3.4: Visualization of anomaly probability distributions on HDFS, BGL and Thunderbird datasets.

Backbone	Random		MLM	
Setting	w/o RTD	w/ RTD	w/o RTD	w/ RTD
HDFS	90.01	90.55 (+0.54)	90.70	90.76 (+0.06)
BGL	88.34	89.58 (+1.24)	91.50	91.64 (+0.14)
Thunderbird	98.24	98.72 (+0.48)	98.42	98.61 (+0.19)

Table 3.3: Ablation studies in terms of F1-score. We denote w/ and w/o as with RTD and without RTD, respectively.

3.3.3 Ablation Studies

RTD training

To assess the impact of RTD training, we report the performance of FastLogAD-MLM and FastLogAD-Random without RTD training in Table 3.3. A comparison with the original FastLogADs demonstrates that RTD training proves effective, resulting in slightly improved performance across all three datasets. This enhancement can be attributed to RTD's support of the subsequent HST task.

Masking ratio

Additionally, we explore the influence of the masking ratio on FastLogAD's performance. Experimenting with a masking ratio ranging from 0 to 1, incremented by 0.1, we train FastLogAD-MLM on each dataset. Fig. 3.5 with plotted three evaluation metrics reveals a significant drop in performance at $r = 0$, as intact normal sequences are treated as pseudo-anomalies, leading the model to inflate the embedding norm of normal data. However, for $r > 0$, FastLogAD demonstrates effectiveness in creating pseudo-anomalies. A notable observation is a performance drop at higher masking ratios, starting from 0.7 on the HDFS dataset. This can be attributed to the fact that the HDFS dataset comprises relatively short log sequences, making the distinction between normal and abnormal sequences less obvious when considering all the tokens. This implies with a large masking ratio, the model may

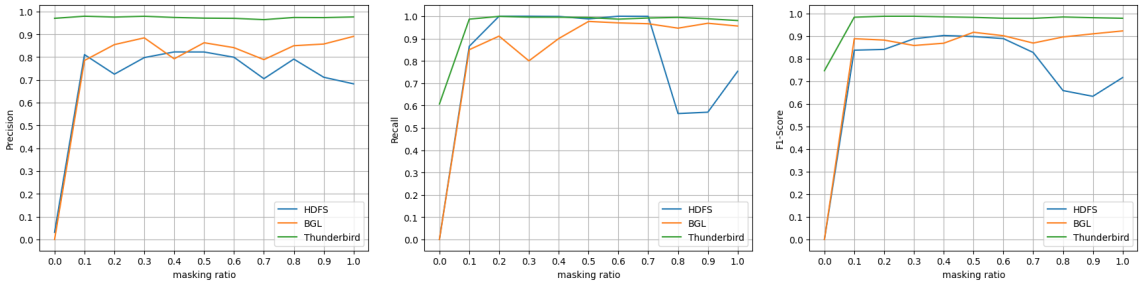

Figure 3.5: Performance comparison of FastLogAD-MLM across different masking ratios (0 to 1).

capture only the patterns of easy pseudo-anomalies, aligning with real anomalies in datasets with long log sequences like BGL and Thunderbird. However, these easy pseudo-anomalies may not align with the characteristics of the HDFS dataset, causing the model to misclassify real hard anomalies as normal. This corresponds to the increasing number of false negatives and a large drop in recall values as shown in the plot.

3.4 Conclusion

We introduced FastLogAD, a novel approach for fast and effective log anomaly detection. Leveraging the Mask-Guided Anomaly Generation (MGAG) and Discriminative Abnormality Separation (DAS) models, FastLogAD demonstrates significant improvements in both detection accuracy and speed compared to existing methods. The use of pseudo-abnormal logs, generated by replacing masked tokens in normal sequences, allowed us to train a highly discriminative model capable of distinguishing between normal and anomalous logs without the need for additional abnormal data.

Extensive experiments on benchmark datasets, including HDFS, BGL, and Thunderbird, confirmed the superiority of FastLogAD, achieving the highest F1-scores and at least a tenfold increase in anomaly detection speed over prior work. This performance highlights the effectiveness of our approach in real-world scenarios, where rapid and accurate detection is crucial for maintaining system reliability and security.

Chapter 4

LogREAD: Online Log Instance Anomaly Detection

4.1 Introduction

In this chapter, we delve into our approach to online log instance anomaly detection, LogREAD. As computer systems generate vast and dynamic log data streams, the ability to continuously learn and adapt to new log patterns in real-time becomes increasingly vital. Offline methods fall short in handling evolving data distributions, necessitating an approach that not only detects anomalies swiftly but also adapts to changing log patterns without retraining. The dynamic nature of log data introduces several challenges for anomaly detection. Logs are often unstructured and vary significantly across different systems, complicating the creation of a one-size-fits-all solution. Additionally, the non-stationarity of log data—where normal behavior evolves due to software updates, configuration changes, and varying usage patterns—requires models to continually learn and adapt to new patterns.

Existing approaches to online log anomaly detection often rely on extensive preprocessing and fixed model architectures, limiting their ability to adapt to new and unforeseen log formats. These methods also struggle with parsing errors and are typically unable to handle the evolving nature of log data effectively.

To address these challenges, we introduce LogREAD, a novel approach designed for online continual learning in dynamic environments. LogREAD begins with offline training on a small dataset using contrastive learning to capture semantic embeddings of log entries. It maintains an adaptive memory bank of representative normal logs, which helps in recognizing and adapting to new log patterns as they emerge. During online operation, LogREAD dynamically updates its memory and feature extractor to handle evolving log patterns.

We evaluate LogREAD on the BGL [49], Thunderbird [49], and Spirit [49] datasets under both offline and online scenarios with existing counterparts. The experimental results show that LogREAD performs on par with or even better than existing offline methods and outperforms all the online methods.

The contributions of this study can be summarized as follows:

- We introduce LogREAD, an innovative approach for online continual log instance anomaly detection that combines offline contrastive learning with a dynamic, parsing-free online framework.
- We develop an adaptive memory bank and an evolving feature extractor that enable LogREAD to continually learn and adapt to new log patterns.
- We demonstrate LogREAD's effectiveness on multiple real-world datasets, showing its superior performance in both offline and online settings.

4.2 Methodology

4.2.1 System Overview

Problem Formulation: Let $\mathcal{D}_{train} = \{\boldsymbol{s}_i \mid \forall y_i = 0\}_{i=1}^{N}$ represent the set of logs available for offline training, where each $y_i = 0$ indicates a normal log. The objective is to train a model using $\mathcal{D}_{train}$ so that it can detect anomalies and adapt to new log data during the online phase. This subsequent log data is denoted by $\mathcal{D}_{test} = \{\boldsymbol{s}_i \mid \forall y_i \in (0,1)\}_{i=N+1}^{N+M}$, which includes both normal and abnormal logs. Note, different from the canonical offline scenario which assumes $\mathcal{D}_{train}$ and $\mathcal{D}_{test}$ follow the same distribution, the problem tackled in this study allows data distribution drift during model deployment.

Schematic Scheme. The workflow of LogREAD designed for detecting anomalies in continuously evolving log data is depicted by Fig. 4.1. LogREAD begins with offline training on a small amount of log data collected during normal system operation. The feature extractor, BERT [12], employs contrastive learning to capture the semantic embeddings of normal logs. Subsequently, a lightweight memory bank is created by selectively including the embeddings that represent the most typical normal data. During the online phase, LogREAD makes predictions by measuring the distance between the embedding of each incoming log and its k-nearest neighbours within the memory bank. In the event of incorrect predictions, either memory bank update or BERT tuning will be performed.

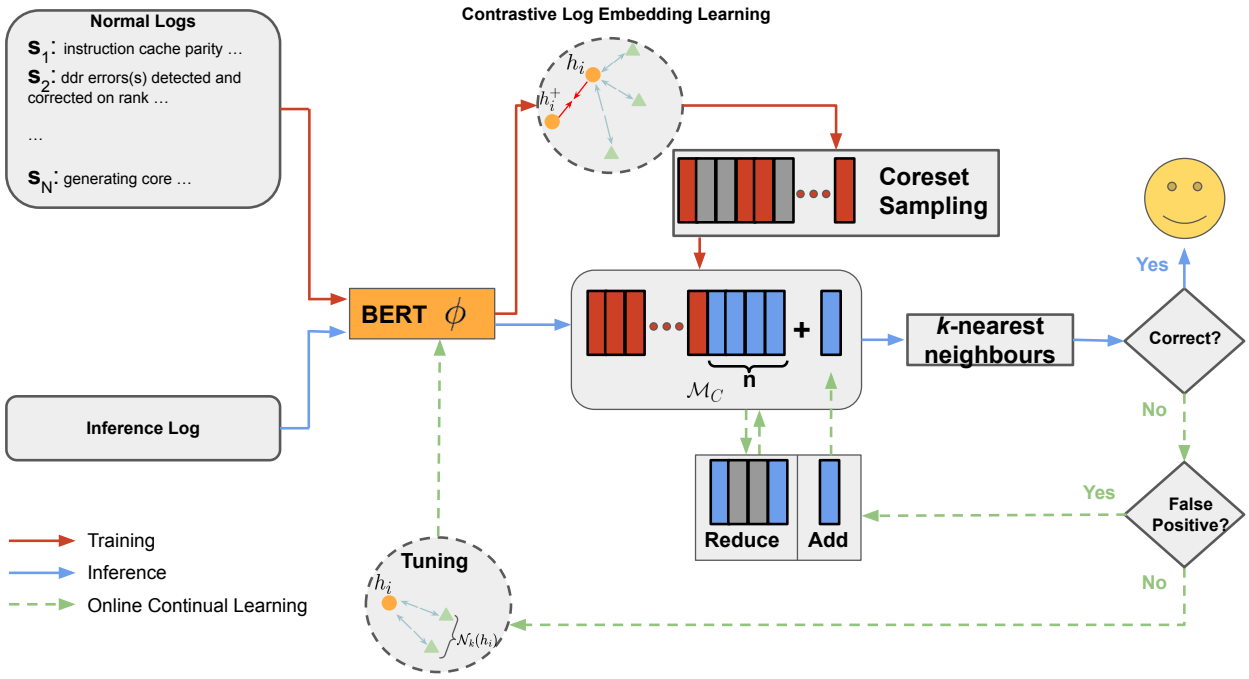

Figure 4.1: The LogREAD framework primarily consists of a BERT [12] feature extractor ϕ and a coreset-reduced memory bank $\mathcal{M}_C$. During offline training, the feature extractor employs contrastive learning to derive semantic embeddings of normal logs, while the memory bank selectively includes embeddings representing the most typical normal data. In the online phase, LogREAD predicts anomalies by measuring the distance between the embedding of each incoming log and its k-nearest neighbors in the memory bank. If incorrect predictions occur, the system will either expand the memory bank or tune the feature extractor.

4.2.2 Parsing-Free Offline Training

Logs often evolve as program development, system update, and dynamic configuration variation, leading to changes in log structures. To address such data drift, traditional log anomaly detection methods relying on log parsing require constant updates to templates or rules in the parsing step, which can be cumbersome and difficult to scale as log data grows in size and complexity. Parsing-free methods, however, are inherently adaptable to these changes. To achieve a parsing-free model scaling effectively with growing and evolving log data, our LogREAD essentially consists of a BERT model for log embedding and a compact memory bank for normal pattern distinction.

Contrastive log embedding learning:

Our LogREAD utilizes a pre-trained BERT [12] to convert a log into numerical semantic embedding. Due to the lack of large annotated log data, unsupervised contrastive learning is applied to the normal logs, pulling semantically close neighbours together and pushing apart non-neighbours [22]. In the domain of contrastive learning for natural language sentence embeddings, various data augmentation methods have been applied to construct both positive and negative pairs, including word deletion and synonym/antonym replacement [63, 65]. However, these methods cannot be directly transferred to log data due to the special acronyms and jargon in log messages. And there is often no corresponding synonym or antonym to replace the original word. Therefore, we adopt the simple yet effective augmentation method introduced in SimCSE [17] by utilizing the natural dropout mask from the transformer model [61]. Let $\boldsymbol{h}_i = \phi(\boldsymbol{s}_i)$ denote the output feature embedding corresponding to the [CLS] token prepended to the beginning of each log, the stochastic dropout mechanism allows us to feed the same log $\boldsymbol{s}_i$ to the feature extractor ϕ twice and create a positive pair $(\boldsymbol{h}_i, \boldsymbol{h}_i^+)$:

$$(\boldsymbol{h}_i, \boldsymbol{h}_i^+) = (\phi(\boldsymbol{s}_i), \phi(\boldsymbol{s}_i)). \tag{4.1}$$

Then the BERT model is updated by minimizing the following contrastive loss

$$l_i = -\log \frac{e^{\mathbf{sim}(\boldsymbol{h}_i, \boldsymbol{h}_i^+)/\tau}}{\sum_{j=1}^{N_{\text{batch}}} e^{\mathbf{sim}(\boldsymbol{h}_i, \boldsymbol{h}_j^+)/\tau}}, \tag{4.2}$$

where **sim** denotes the cosine similarity score, τ is a temperature hyperparameter, and N_{batch} is the training batch size.

Coreset-reduced memory bank:

With the preceding step of log semantic embedding learning, it is reasonable assume that normal logs are diversely, but compactly, distributed in the feature space. Anomalies can then be identified if the embedding of an incoming log is far from these normal embeddings. We create a memory bank $\mathcal{M}$ for storing the training log embeddings, essential for offline inference and online continual learning since the past-seen data is unavailable upon detection. In the OCL setting, new representative samples may merge for improved performance. Specifically, we adopt coreset sampling [53] on the memory bank $\mathcal{M}$ for efficient and scalable anomaly detection:

$$\mathcal{M}_C = \underset{\mathcal{M}_C \subset \mathcal{M}}{\arg\min} \max_{\boldsymbol{h} \in \mathcal{M}} \min_{\boldsymbol{h'} \in \mathcal{M}_C} \|\boldsymbol{h} - \boldsymbol{h'}\|^2, \tag{4.3}$$

where $\mathcal{M}_C$ is the coreset-reduced memory bank that covers an approximately similar region as the original $\mathcal{M}$ in the feature space. Unlike other methods that fill the memory bank with local patch features [38, 53], our approach directly utilizes the embedding of each log due to its highly discrete nature, where a corresponding spatial structure does not exist. The whole offline training process for LogREAD is detailed in Algorithm 3.

Algorithm 3 LogREAD offline training

Require: Training set $\mathcal{D}_{train}$, training epochs N_{epoch}, pre-trained BERT ϕ, batch size N_{batch}, coreset sampling ratio q.

Step 1: Contrastive Log Embedding Learning
Set ϕ to training mode ▷ To apply dropout for contrastive learning
for epoch $\in [1, \dots N_{epoch}]$ **do**
  for $\boldsymbol{s}_i \in \mathcal{D}_{train}$ **do**
    $(\boldsymbol{h}_i, \boldsymbol{h}_i^+) \leftarrow (\phi(\boldsymbol{s}_i), \phi(\boldsymbol{s}_i))$
    Compute l_i with (4.2)
    $\mathcal{L}_{batch} \leftarrow \sum_{i=1}^{N_{batch}} l_i / N_{batch}$
    Optimize $\mathcal{L}_{batch}$
  end for
end for
Step 2: Memory Bank Filling and Reduction
Set ϕ to test mode
$\mathcal{M} \leftarrow \{\}$
for $\boldsymbol{s}_i \in \mathcal{D}_{train}$ **do**
  $\mathcal{M} \leftarrow \mathcal{M} \cup \phi(\boldsymbol{s}_i)$
end for
$\mathcal{M}_C \leftarrow \{\}$
for $j \in [1, 2, \dots, q \times |\mathcal{M}|]$ **do** ▷ Coreset Sampling
  $\boldsymbol{h}_j \leftarrow \arg\max_{\boldsymbol{h}_m \in \mathcal{M} - \mathcal{M}_C} \min_{\boldsymbol{h}_n \in \mathcal{M}_C} \|\boldsymbol{h}_m - \boldsymbol{h}_n\|_2$
  $\mathcal{M}_C \leftarrow \mathcal{M}_C \cup \{\boldsymbol{h}_j\}$
end for
return $\phi, \mathcal{M}_C$

4.2.3 Anomaly Detection

With the ready memory bank $\mathcal{M}_C$, the anomaly score of each upcoming log can be computed as the average L2-distance between its embedding and k-nearest neighbours [10] from $\mathcal{M}_C$:

$$\text{score}(\boldsymbol{s}_i) = \frac{1}{k} \sum_{\boldsymbol{h} \in \mathcal{N}_k(\boldsymbol{h}_i)} \|\boldsymbol{h}_i - \boldsymbol{h}\|_2, \tag{4.4}$$

where $\mathcal{N}_k(\boldsymbol{h}_i)$ is the set of k-nearest neighbours of $\boldsymbol{h}_i$ from $\mathcal{M}_C$. A threshold ϵ is chosen based on a validation set such that an input log $\boldsymbol{s}_i$ is predicted to be anomalous if $\text{score}(\boldsymbol{s}_i) > \epsilon$, and normal otherwise.

4.2.4 Online Continual Learning

Due to the diversity in both the format and semantics of log data, pure offline approaches have limitations as new logs are easily missed. As suggested in [6], our approach offers a great capability of adaptively adjusting the model according to the feedback from real-time detection. The entire process is performed on $\mathcal{D}_{test}$ and the model processes one log at a time.

Memory Bank Expansion

During real-time detection, new log events may appear due to system updates or development. Despite being normal, these new logs may deviate significantly from the existing space described by the embeddings in $\mathcal{M}_C$. In this case, the false alarm is triggered and our solution is to add $\boldsymbol{h}_i$ to the existing memory bank $\mathcal{M}_C$. To better manage the growing size of the memory bank, we track the most recent n feature samples added to $\mathcal{M}_C$ and denote this temporary set as $\mathcal{M}_{temp}$. Once it reaches its full capacity of n samples, we apply coreset sampling to this temporary set and replace the recently added n samples in $\mathcal{M}_C$ with the sampled feature embeddings. $\mathcal{M}_{temp}$ is cleared after each sampling and the iteration repeats.

Feature Extractor Tuning

Similarly, new anomalous logs may impact detection performance when our feature extractor fails to isolate them from the seen normal logs in their embedding space. This false-negative prediction can be attributed to the similar word appearance in both normal and abnormal logs which the feature extractor deems close together. To address this issue, we further tune the feature extractor to force the embedding $\boldsymbol{h}_i$ of such an abnormal log away from its

k-nearest neighbours:

$$\mathcal{L}_{knn} = \max(0, B - \frac{1}{k} \sum_{\boldsymbol{h} \in \mathcal{N}_k(\boldsymbol{h}_i)} \|\boldsymbol{h}_i - \boldsymbol{h}\|_2), \tag{4.5}$$

where B is an upper bound to determine how far the feature embedding needs to be pushed away. $\mathcal{L}_{knn}$ alone ensures a larger distance from the compared normal samples, but it has no control over the direction in which the log embedding gets pushed. The potential concerns include:

- The log embedding may get pushed close to the other embeddings within the memory bank $\mathcal{M}_C$ other than the k neighbours, i.e. $\boldsymbol{h} \in \mathcal{M}_C \setminus \mathcal{N}_k(\boldsymbol{h}_i)$.
- Directions ensure the diversity and uniformity of the semantic embeddings in the feature space. Moving a log embedding toward a random direction could alter the existing mapping of the normal logs, causing a mismatch between the new mapping of a previously seen normal log and its existing embedding in the memory bank. The stored embeddings are no longer representative for comparison and this can lead to performance collapse.

To address these concerns, we record the initial embedding $\boldsymbol{h}_i$ as $\boldsymbol{h}_i^*$ and apply an L2-regularization $\mathcal{L}_{dir}$ on the embedding to ensure directional alignment:

$$\mathcal{L}_{dir} = \|\boldsymbol{h}_i - \boldsymbol{h}_i^*\|_2. \tag{4.6}$$

We also apply an L2-regularization to the feature extractor's parameters, denoted by $\boldsymbol{\theta}$, to ensure minimal changes to the mapping of the rest of the logs:

$$\mathcal{L}_{\boldsymbol{\theta}} = \|\boldsymbol{\theta} - \boldsymbol{\theta}^*\|_2 \tag{4.7}$$

where $\boldsymbol{\theta}^*$ denotes the network parameters before online learning is applied to the current sample. This regularization method has been proven effective in continual learning [30, 37] and online log anomaly detection [13] approaches. To balance distance maximization and regularization, we further add a weight term λ to the regularization terms such that the total loss for false negative events can be represented as:

$$\mathcal{L}_{FN} = \mathcal{L}_{knn} + \lambda(\mathcal{L}_{dir} + \mathcal{L}_{\boldsymbol{\theta}}) \tag{4.8}$$

The complete online continual learning process is detailed in Algorithm 4.

Algorithm 4 LogRead online continual learning

Require: Test set $\mathcal{D}_{test}$, online epochs N_{online}, trained feature extractor ϕ, memory bank $\mathcal{M}_C$, threshold ϵ, number of samples n for re-scaling the memory bank, number of nearest neighbours k, upper bound hyperparameter τ, regularization weight λ.

for $(\boldsymbol{s}_i, y_i) \in \mathcal{D}_{test}$ **do**
  Step 1: Perform Detection
  Set ϕ to test mode
  $\boldsymbol{h}_i \leftarrow \phi(\boldsymbol{s}_i)$
  Compute score($\boldsymbol{s}_i$) with (4.4)
  if score($\boldsymbol{s}_i$) $> \epsilon$ **then**
    $y_i' \leftarrow 1$
  else
    $y_i' \leftarrow 0$
  end if
  Step 2: Online Learning
  Set ϕ to training mode
  counter $\leftarrow 0$
  if $y_i = 0$ and $y_i' = 1$ **then** ▷ False Positive
    $\mathcal{M}_C \leftarrow \mathcal{M}_C \cup \{\boldsymbol{h}_i\}$
    counter $\leftarrow$ counter $+1$
    if counter $= n$ **then**
      $\mathcal{M}_{temp} \leftarrow \mathcal{M}_C[-n:]$
      $\mathcal{M}_C \leftarrow \mathcal{M}_C[:-n]$
      $\mathcal{M}_C \leftarrow \mathcal{M}_C \cup \text{CoresetSample}(\mathcal{M}_{temp})$
      counter $\leftarrow 0$
    end if
  else if $y_i = 1$ and $y_i' = 0$ **then** ▷ False Negative
    $\boldsymbol{\theta}^* \leftarrow \boldsymbol{\theta}$
    $\boldsymbol{h}_i^* \leftarrow \boldsymbol{h}_i$
    for epoch $\in [1, \ldots, N_{online}]$ **do**
      Compute $\mathcal{L}_{FN}$ with (4.8)
      Optimize $\mathcal{L}_{FN}$
      Compute score($\boldsymbol{s}_i$) with (4.4)
      if score($\boldsymbol{s}_i$) $> \epsilon$ **then**
        Early Stopping
      end if
    end for
  end if
end for

4.3 Experiment

4.3.1 Experimental Settings

Datasets

We validate LogREAD on BGL [49], Thunderbird [49] and Spirit [49] datasets. A chronological order is maintained according to the timestamps to ensure the training dataset does not expose any distribution drift in advance. For offline baseline comparison, we take the first 60% of normal logs of each dataset as the training data. Using a large training ratio would help reduce the variation in results and these offline baselines are designed to be trained with more data. On the contrary, we utilize only 30% of normal logs for offline training and the rest of the data is for simulating the OCL scenario, as less training data can examine how well the methods evolve under potential distribution drifts.

We compare our LogREAD to several offline and online log anomaly detection baselines. Due to the limited availability of public implementations in log anomaly detection research [34], we either replicate non-public baselines based on the provided implementation details or adapt publicly available baselines to our experimental setting. The baselines are listed below, with an asterisk (*) indicating our replicated version.

Offline methods: LogBERT [21], Logsy* [47]

Online methods: UnLearn* [13], LogOnline [64]

Evaluation Metrics

Following the existing works, we adopt Precision, Recall, F1-Score and the average inference time per log as evaluation metrics:

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}};\ \text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}};\ \text{F1-Score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

TP: True Positive; FP: False Positive; FN: False Negative;

We use a batch size of 1 to infer all the compared methods, ensuring that the offline and online inference times are unaffected by the batch size. The average inference time is recorded in milliseconds per log.

Experimental Setup

We conduct our experiments on a Linux Server with Intel i9-9940X @ 3.30GHz CPU and 64GB of RAM. One Nvidia RTX 2080Ti GPU is used for training under the Python 3.8.10 environment with Pytorch 2.1.0+cu118 installed.

Implementation Details

For LogREAD, we use `bert_uncased_L-4_H-256_A-4` as the feature extractor ϕ, with 10 epochs of training on the contrastive log embeddings. The coreset sampling ratio is set to 50%, and $k = 2$ is the number of nearest neighbors for comparison. λ is set to 0.5 and the number of samples n for re-scaling the memory bank is 32 in the online continual learning algorithm. AdamW [41] optimizer is employed for both offline and online learning. The learning rate is set to 3e-5 for offline learning and 3e-6 for online continual learning to ensure stability. During online updates, we perform 10 backpropagations for each instance update to ensure robust continual learning.

4.3.2 Results

Offline

Table 4.1 shows the experimental results of the baselines compared to LogREAD in offline mode. The results are based on running with 3 random seeds. Logsy achieves nearly perfect detection on the Thunderbird dataset but lacks consistency on the other two datasets, particularly showing an average F1-Score of 67.61% on the Spirit dataset. This indicates that Logsy's performance heavily depends on the auxiliary datasets used to represent the abnormal class during training. If the auxiliary dataset fails to represent the abnormal class accurately, Logsy's performance will collapse significantly. Similar to LogREAD, LogBERT maintains consistent performance across all three datasets. However, its latency becomes an issue since LogBERT was originally built for sequence-level detection, and converting it to log-level detection slows it down substantially. Moreover, LogBERT requires both log parsing and a fixed-sized embedding layer representing the training logs' vocabulary, making it unsuitable for online learning scenarios. The performance of the two online baselines in offline modes lags LogREAD across all three datasets. Despite performing slightly behind several baselines on the Thunderbird dataset, LogREAD achieves the best offline F1-Score on the BGL and Spirit datasets. Regarding the inference speed, LogREAD is only about 0.3 milliseconds slower per log than Logsy and approximately 1.5 times faster than LogBERT.

BGL				
Methods	Precision (%)	Recall (%)	F1 (%)	Time (ms)
Logsy* [47]	75.78 ± 15.91	91.44 ± 4.70	81.08 ± 9.72	1.64 ± 0.02
LogBERT [21]	**93.65 ± 0.04**	80.80 ± 0.01	86.75 ± 0.01	3.80 ± 0.05
LogOnline [64]	30.26 ± 8.80	82.51 ± 9.57	42.61 ± 5.13	1.14 ± 0.01
UnLearn* [13]	82.67 ± 2.64	**95.32 ± 0.23**	88.53 ± 1.41	0.53 ± 0.05
LogREAD	91.44 ± 2.38	88.94 ± 5.38	**89.86 ± 4.51**	1.84 ± 0.00
Thunderbird				
Methods	Precision (%)	Recall (%)	F1 (%)	Time (ms)
Logsy* [47]	**100 ± 0.00**	**99.99 ± 0.00**	**99.99 ± 0.00**	1.68 ± 0.37
LogBERT [21]	87.77 ± 0.89	**99.99 ± 0.00**	93.48 ± 0.50	3.44 ± 0.04
LogOnline [64]	84.56 ± 7.82	97.03 ± 0.83	90.24 ± 4.53	1.47 ± 0.05
UnLearn* [13]	10.16 ± 5.34	26.95 ± 2.51	14.75 ± 3.98	0.50 ± 0.03
LogREAD	84.81 ± 1.64	98.56 ± 2.36	91.16 ± 1.52	1.98 ± 0.01
Spirit				
Methods	Precision (%)	Recall (%)	F1 (%)	Time (ms)
Logsy* [47]	56.23 ± 9.45	99.98 ± 0.01	67.61 ± 7.76	1.75 ± 0.11
LogBERT [21]	90.92 ± 0.29	**99.99 ± 0.00**	93.48 ± 0.51	3.27 ± 0.00
LogOnline [64]	24.19 ± 2.53	84.42 ± 10.63	37.36 ± 1.80	1.34 ± 0.01
UnLearn* [13]	45.43 ± 2.00	62.43 ± 10.22	52.31 ± 2.31	0.65 ± 0.05
LogREAD	**99.64 ± 0.18**	99.87 ± 0.14	**99.72 ± 0.13**	1.86 ± 0.00

Table 4.1: Offline results (mean ± std) on BGL, Thunderbird and Spirit datasets with a 0.6 training split ratio. The best metric values are **bolded** and the runner-up is highlighted with undeline.

Online

The comparison of online methods is shown in Table 4.2. We also include the offline baselines' experimental results under this setting. LogBERT's performance falls on the BGL and Thunderbird datasets compared to its offline result due to a smaller training ratio and no online learning mechanism. LogREAD outperforms the other baselines by a large margin on all three datasets. This demonstrates the effectiveness of its online mechanism. While requiring no manual feedback, LogOnline necessitates specific log headers in the datasets(e.g. BGL) for online learning. Furthermore, its reliance on a frozen auxiliary autoencoder results in a low precision score because the autoencoder which is trained only on the offline dataset, tends to predict all the unseen logs as abnormal. UnLearn, another online baseline, shows better performance than LogOnline on the BGL dataset. It achieves a Precision of 77.53% and an F1-Score of 85.58%, with a significant advantage in inference time, being the fastest among all methods with 0.39 milliseconds per log. However, UnLearn's performance is notably poor on the Thunderbird and Spirit datasets, with F1-Scores of 14.16% and 50.89%. This inconsistency highlights UnLearn's difficulty in generalizing across different datasets. LogREAD exhibits superior online detection accuracy. The increase in inference time compared to its offline performance is within an acceptable range. More representative log embeddings are stored in the memory bank to support strong online performance and account for the additional inference time.

Figure 4.2 provides a comprehensive comparison of F1-scores for all evaluated methods, presented through separate bar plots for offline and online modes. This visualization allows for a clearer assessment of performance across different techniques.

4.3.3 Ablation Study

To further evaluate the effectiveness of LogREAD's online mechanisms, we compare LogREAD to its two variants, each excluding one of the online mechanisms: memory expansion or extractor tuning. Table 4.3 presents the results of this ablation study, with values averaged over three runs. Overall, the results demonstrate the significant contribution of both mechanisms to LogREAD's online detection performance. When either mechanism is excluded, there is a noticeable drop in the overall F1-Score across the three datasets. For instance, the F1-Score for the "w/o memory expansion" variant drops to 87.36% on the BGL dataset, compared to 99.97% for LogREAD. Similarly, the "w/o extractor tuning" variant shows a reduced F1-Score of 95.19% on the BGL dataset. Additionally, we observe specific decreases in either Precision or Recall for each variant. For example, the Precision of the "w/o memory expansion" variant drops to 75.12% on the Thunderbird dataset, whereas LogREAD

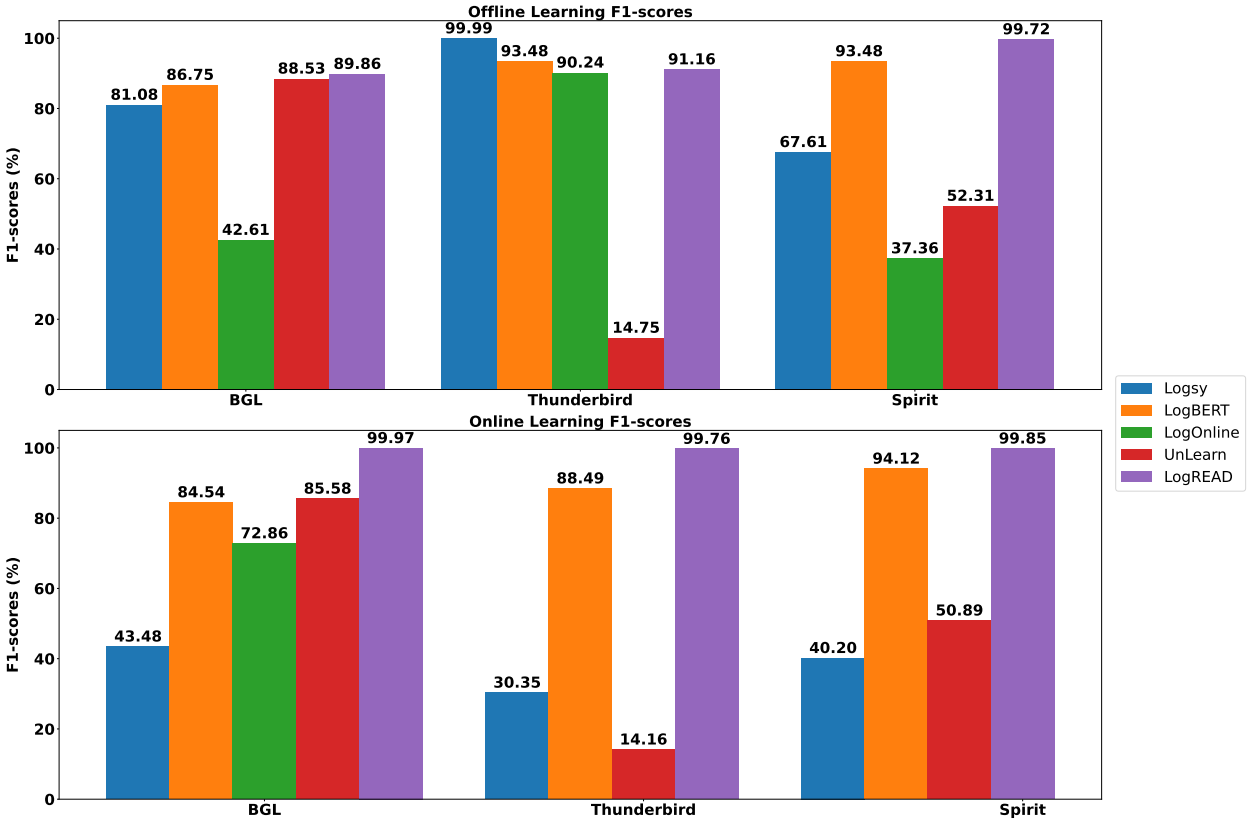

Figure 4.2: Bar plots illustrating the F1-scores of various methods in both offline and online modes across the BGL, Thunderbird, and Spirit datasets.

BGL				
Methods	Precision (%)	Recall (%)	F1 (%)	Time (ms)
Logsy* [47]	27.78 ± 0.01	**99.99 ± 0.00**	43.48 ± 0.01	1.52 ± 0.07
LogBERT [21]	88.63 ± 0.62	80.81 ± 0.01	84.54 ± 0.28	3.27 ± 0.01
LogOnline [64]	63.10 ± 10.60	85.88 ± 9.65	72.86 ± 2.67	1.96 ± 0.11
UnLearn* [13]	77.53 ± 0.26	95.50 ± 0.25	85.58 ± 0.06	0.39 ± 0.00
LogREAD	**99.97 ± 0.02**	**99.99 ± 0.00**	**99.97 ± 0.01**	2.06 ± 0.03
Thunderbird				
Methods	Precision (%)	Recall (%)	F1 (%)	Time (ms)
Logsy* [47]	20.28 ± 21.86	85.13 ± 25.74	30.35 ± 17.97	1.22 ± 0.08
LogBERT [21]	79.51 ± 6.33	**99.99 ± 0.00**	88.49 ± 3.96	3.89 ± 0.22
UnLearn* [13]	8.47 ± 0.57	43.22 ± 1.90	14.16 ± 0.90	0.53 ± 0.01
LogREAD	**99.54 ± 0.21**	**99.99 ± 0.00**	**99.76 ± 0.10**	1.91 ± 0.01
Spirit				
Methods	Precision (%)	Recall (%)	F1 (%)	Time (ms)
Logsy* [47]	25.25 ± 4.08	99.96 ± 0.05	40.20 ± 5.30	1.28 ± 0.09
LogBERT [21]	88.90 ± 0.31	**99.99 ± 0.00**	94.12 ± 0.17	3.32 ± 0.02
UnLearn* [13]	43.20 ± 0.80	62.04 ± 4.21	50.89 ± 0.86	0.52 ± 0.00
LogREAD	**99.71 ± 0.13**	**99.99 ± 0.00**	**99.85 ± 0.06**	1.90 ± 0.02

Table 4.2: Online results on BGL, Thunderbird and Spirit datasets with a 0.3 training split ratio. The best metric values are **bolded** and the runner-up is highlighted with undeline.

achieves 99.54%. The Recall for the "w/o extractor tuning" variant on the Spirit dataset is 91.34%, compared to 99.99% for LogREAD. This outcome aligns with our design goals: memory expansion aims to reduce the number of normal log data predicted as abnormal, thereby lowering the false positive rate. Similarly, the feature extractor tuning is designed to lower the false negative rate by improving the model's ability to identify anomalous logs correctly.

By combining these two mechanisms, LogREAD achieves a superior F1-Score, balancing both Precision and Recall effectively.

Variant	BGL			Thunderbird			Spirit		
	P (%)	R (%)	F1 (%)	P(%)	R (%)	F1 (%)	P (%)	R (%)	F1 (%)
LogREAD	99.97	99.99	99.97	99.54	99.99	99.76	99.71	99.99	99.85
w/o memory expansion	77.94	99.99	87.36	75.12	98.45	85.26	80.56	97.88	88.50
w/o extractor tuning	99.98	90.84	95.19	99.45	89.56	94.22	98.72	91.34	94.89

Table 4.3: Ablation study results across three datasets in online scenarios. "w/o memory expansion" denotes the variant without memory expansion for false positives, and "w/o extractor tuning" corresponds to the variant without extractor tuning for false negatives. "P" and "R" stand for Precision and Recall metrics. All the values are averaged over three runs.

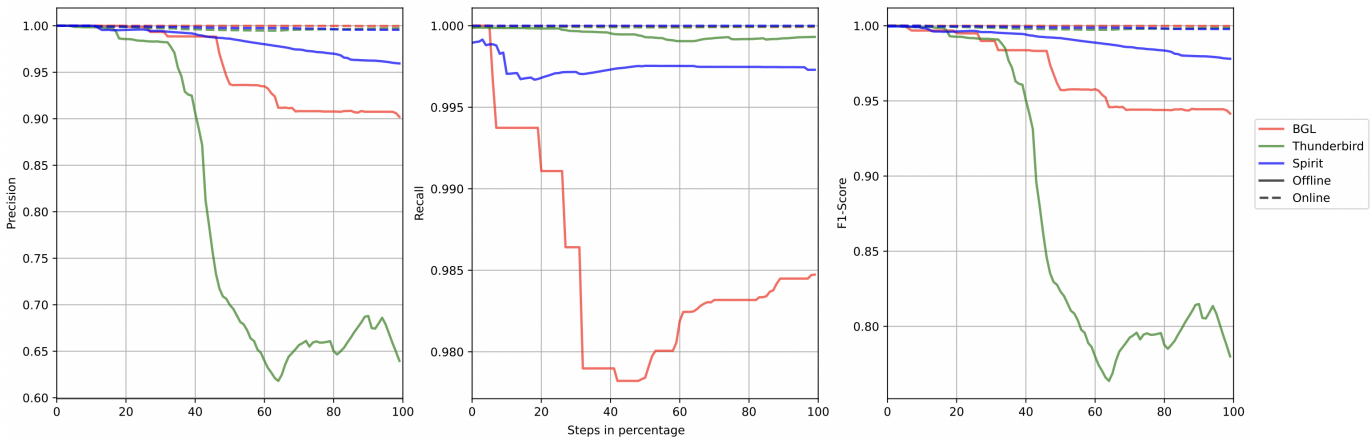

Figure 4.3: Comparison of cumulative test scores of LogREAD in offline and online modes across three datasets with training ratio set to 0.3.

4.3.4 Visualization

Figure 4.3 provides a visual comparison of cumulative test scores of LogREAD in offline and online modes across the three datasets. The x-axis represents the percentage steps during testing, starting from 1000 samples. The y-axis represents the performance metrics as percentages. The curves represent the Precision, Recall, and F1-scores for both offline and online learning scenarios. From the figure, we observe a rapid decline in test scores when no online learning is applied. LogREAD's online learning mechanism effectively maintains superior performance, with no significant drop over time. The online learning curves for Precision, Recall, and F1-Score remain relatively stable and high compared to their offline counterparts, demonstrating the robustness and adaptability of LogREAD in an evolving log data environment.

4.4 Conclusion

In this chapter, we introduced LogREAD, a robust and effective approach for log anomaly detection that is particularly suited for online continual learning scenarios. Our method addresses the limitations of existing online models, which often struggle with non-stationary data and rely on extensive preprocessing. LogREAD is a parsing-free approach that employs an adaptively reduced memory bank and a continuously evolved feature extractor, making it highly adaptable to various log formats and more resilient to data distribution drifts. Extensive experiments demonstrate LogREAD's stable performance across all the datasets in both online and offline environments.

Chapter 5

Conclusion and Future Work

5.1 Conclusion

In this thesis, we addressed the challenges of log anomaly detection in large-scale and dynamic computer systems by developing two approaches: FastLogAD for offline log sequence anomaly detection and LogREAD for online log instance anomaly detection.

FastLogAD was designed to tackle the need for fast and accurate log anomaly detection in offline settings. By incorporating Mask-Guided Anomaly Generation (MGAG) and Discriminative Abnormality Separation (DAS), FastLogAD generates tailored pseudo-anomalies and trains a robust discriminator to efficiently separate normal and anomalous log sequences. Our experiments on the HDFS, BGL, and Thunderbird datasets demonstrated that FastLogAD not only achieves the highest F1-scores among existing methods but also significantly increases detection speed, making it a practical solution for real-time log analysis.

LogREAD addresses the challenges of online anomaly detection in dynamic log environments, with a focus on log instances. By leveraging contrastive learning for offline training and maintaining an adaptive memory bank along with a continuously evolved feature extractor, LogREAD can dynamically adjust to new log patterns with minimal preprocessing as a parsing-free approach. Our evaluations on the BGL, Thunderbird, and Spirit datasets showed that LogREAD outperforms all existing online methods and performs on par with or even better than existing offline methods in instance-based anomaly detection, demonstrating its robustness and adaptability in real-world scenarios.

Together, these contributions advance the field of log anomaly detection by providing effective solutions for both log sequence and log instance anomaly detection, ensuring timely and accurate identification of anomalies while adapting to evolving log data.

5.2 Future Work

While this thesis has made certain contributions to log anomaly detection, several challenges remain open in this field:

- **Solving the Out-Of-Vocabulary (OOV) issue:** Despite FastLogAD's high overall F1-Score, it exhibits a comparatively lower precision score due to false positive predictions, particularly with sequences that are out of the vocabulary (OOV) trained on the training data. These sequences are encoded into [unk] tokens, increasing the likelihood of being recognized as anomalies. To address this, future research should explore the incorporation of additional features, such as semantic features, to enhance robustness against Out-Of-Vocabulary (OOV) issues.

- **Extending FastLogAD to pure unsupervised anomaly detection:** FastLogAD could be adapted to support pure unsupervised anomaly detection. This extension would eliminate the need for fully normal training data, thereby reducing the cost associated with manual labelling and broadening FastLogAD's applicability.

- **Optimizing LogREAD:** Future work could focus on optimizing LogREAD's design to improve latency and eliminate the need for manual feedback while maintaining performance. Enhancing its adaptability to log sequence anomaly detection and a broad range of systems would extend its applicability in real-world scenarios.

- **Scability to diverse systems:** It is crucial to ensure the scalability and effectiveness of FastLogAD and LogREAD across various log formats. Future research could involve creating more complex log datasets relevant to contemporary computer systems and developing generalized models adapting to diverse logging environments. Cross-domain adaptability and universal log representation frameworks are promising areas for this endeavor.

Bibliography

[1] Charu C. Aggarwal. *An Introduction to Outlier Analysis*, pages 1–34. Springer International Publishing, Cham, 2017.

[2] Mohiuddin Ahmed, Abdun Naser Mahmood, and Md Rafiqul Islam. A survey of anomaly detection techniques in financial domain. *Future Generation Computer Systems*, 55:278–288, 2016.

[3] Montdher Alabadi and Yuksel Celik. Anomaly detection for cyber-security based on convolution neural network : A survey. In *2020 International Congress on Human-Computer Interaction, Optimization and Robotic Applications (HORA)*, pages 1–14, 2020.

[4] Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. Enriching word vectors with subword information. *Transactions of the association for computational linguistics*, 5:135–146, 2017.

[5] Zhipeng Cai, Ozan Sener, and Vladlen Koltun. Online continual learning with natural distribution shifts: An empirical study with visual data. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 8281–8290, 2021.

[6] Raghavendra Chalapathy and Sanjay Chawla. Deep learning for anomaly detection: A survey. *arXiv preprint arXiv:1901.03407*, 2019.

[7] Varun Chandola, Arindam Banerjee, and Vipin Kumar. Anomaly detection: A survey. *ACM Comput. Surv.*, 41(3), jul 2009.

[8] Kyunghyun Cho, Bart van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using RNN encoder–decoder for statistical machine translation. In Alessandro Moschitti, Bo Pang, and Walter Daelemans, editors, *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1724–1734, Doha, Qatar, October 2014. Association for Computational Linguistics.

[9] Kevin Clark, Minh-Thang Luong, Quoc V Le, and Christopher D Manning. Electra: Pre-training text encoders as discriminators rather than generators. *arXiv preprint arXiv:2003.10555*, 2020.

[10] T. Cover and P. Hart. Nearest neighbor pattern classification. *IEEE Transactions on Information Theory*, 13(1):21–27, 1967.

[11] Anwesha Das, Frank Mueller, Charles Siegel, and Abhinav Vishnu. Desh: deep learning for system health prediction of lead times to failure in hpc. In *Proceedings of the 27th international symposium on high-performance parallel and distributed computing*, pages 40–51, 2018.

[12] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. In *North American Chapter of the Association for Computational Linguistics*, 2019.

[13] Min Du, Fei Tony Li, Longhui Zheng, and Vivek Srikumar. Lifelong anomaly detection through unlearning. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery Data Mining*, pages 2787–2795, 2019.

[14] Min Du and Feifei Li. Spell: Streaming parsing of system event logs. In *2016 IEEE 16th International Conference on Data Mining (ICDM)*, pages 859–864, 2016.

[15] Min Du, Feifei Li, Guineng Zheng, and Vivek Srikumar. Deeplog: Anomaly detection and diagnosis from system logs through deep learning. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, CCS '17, page 1285–1298, New York, NY, USA, 2017. Association for Computing Machinery.

[16] Tharindu Fernando, Harshala Gammulle, Simon Denman, Sridha Sridharan, and Clinton Fookes. Deep learning for medical anomaly detection–a survey. *ACM Computing Surveys (CSUR)*, 54(7):1–37, 2021.

[17] Tianyu Gao, Xingcheng Yao, and Danqi Chen. SimCSE: Simple contrastive learning of sentence embeddings. In Marie-Francine Moens, Xuanjing Huang, Lucia Specia, and Scott Wen-tau Yih, editors, *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 6894–6910, Online and Punta Cana, Dominican Republic, November 2021. Association for Computational Linguistics.

[18] Markus Goldstein and Seiichi Uchida. A comparative evaluation of unsupervised anomaly detection algorithms for multivariate data. *PloS one*, 11(4):e0152173, 2016.

[19] Ian J Goodfellow, Mehdi Mirza, Da Xiao, Aaron Courville, and Yoshua Bengio. An empirical investigation of catastrophic forgetting in gradient-based neural networks. *arXiv preprint arXiv:1312.6211*, 2013.

[20] Alex Graves and Alex Graves. Long short-term memory. *Supervised sequence labelling with recurrent neural networks*, pages 37–45, 2012.

[21] Haixuan Guo, Shuhan Yuan, and Xintao Wu. Logbert: Log anomaly detection via bert. In *2021 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8, 2021.

[22] R. Hadsell, S. Chopra, and Y. LeCun. Dimensionality reduction by learning an invariant mapping. In *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06)*, volume 2, pages 1735–1742, 2006.

[23] Hossein Hamooni, Biplob Debnath, Jianwu Xu, Hui Zhang, Guofei Jiang, and Abdullah Mueen. Logmine: Fast pattern recognition for log analytics. In *Proceedings of the 25th ACM International on Conference on Information and Knowledge Management*, CIKM '16, page 1573–1582, New York, NY, USA, 2016. Association for Computing Machinery.

[24] Shangbin Han, Qianhong Wu, Han Zhang, Bo Qin, Jiankun Hu, Xingang Shi, Linfeng Liu, and Xia Yin. Log-based anomaly detection with robust feature extraction and online learning. *IEEE Transactions on Information Forensics and Security*, 16:2300–2311, 2021.

[25] Pinjia He, Jieming Zhu, Zibin Zheng, and Michael R. Lyu. Drain: An online log parsing approach with fixed depth tree. In *2017 IEEE International Conference on Web Services (ICWS)*, pages 33–40, 2017.

[26] 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.

[27] John J Hopfield. Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the national academy of sciences*, 79(8):2554–2558, 1982.

[28] Tong Jia, Yifan Wu, Chuanjia Hou, and Ying Li. Logflash: Real-time streaming anomaly detection and diagnosis from system logs for large-scale software systems. In *2021 IEEE 32nd International Symposium on Software Reliability Engineering (ISSRE)*, pages 80–90, 2021.

[29] Tatsuaki Kimura, Keisuke Ishibashi, Tatsuya Mori, Hiroshi Sawada, Tsuyoshi Toyono, Ken Nishimatsu, Akio Watanabe, Akihiro Shimoda, and Kohei Shiomoto. Spatio-temporal factorization of log data for understanding network events. In *IEEE INFOCOM 2014 - IEEE Conference on Computer Communications*, pages 610–618, 2014.

[30] James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, et al. Overcoming catastrophic forgetting in neural networks. *Proceedings of the national academy of sciences*, 114(13):3521–3526, 2017.

[31] Harold W Kuhn and Albert W Tucker. Nonlinear programming. In *Traces and emergence of nonlinear programming*, pages 247–258. Springer, 2013.

[32] Max Landauer, Sebastian Onder, Florian Skopik, and Markus Wurzenberger. Deep learning for anomaly detection in log data: A survey. *Machine Learning with Applications*, 12:100470, 2023.

[33] Van-Hoang Le and Hongyu Zhang. Log-based anomaly detection without log parsing. In *2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 492–504. IEEE, 2021.

[34] Van-Hoang Le and Hongyu Zhang. Log-based anomaly detection with deep learning: How far are we? In *Proceedings of the 44th international conference on software engineering*, pages 1356–1367, 2022.

[35] Y. Li, Y. Liu, H. Wang, Z. Chen, W. Cheng, Y. Chen, W. Yu, H. Chen, and C. Liu. Glad: Content-aware dynamic graphs for log anomaly detection. In *2023 IEEE International Conference on Knowledge Graph (ICKG)*, pages 9–18, Los Alamitos, CA, USA, dec 2023. IEEE Computer Society.

[36] Qingwei Lin, Hongyu Zhang, Jian-Guang Lou, Yu Zhang, and Xuewei Chen. Log clustering based problem identification for online service systems. In *Proceedings of the 38th International Conference on Software Engineering Companion*, ICSE '16, page 102–111, New York, NY, USA, 2016. Association for Computing Machinery.

[37] Huihui Liu, Yiding Yang, and Xinchao Wang. Overcoming catastrophic forgetting in graph neural networks. In *Proceedings of the AAAI conference on artificial intelligence*, volume 35, pages 8653–8661, 2021.

[38] Jiaqi Liu, Kai Wu, Qiang Nie, Ying Chen, Bin-Bin Gao, Yong Liu, Jinbao Wang, Chengjie Wang, and Feng Zheng. Unsupervised continual anomaly detection with

contrastively-learned prompt. *Proceedings of the AAAI Conference on Artificial Intelligence*, 38(4):3639–3647, Mar. 2024.

[39] Jiaqi Liu, Guoyang Xie, Jinbao Wang, Shangnian Li, Chengjie Wang, Feng Zheng, and Yaochu Jin. Deep industrial image anomaly detection: A survey. *Machine Intelligence Research*, 21(1):104–135, 2024.

[40] David Lopez-Paz and Marc'Aurelio Ranzato. Gradient episodic memory for continual learning. *Advances in neural information processing systems*, 30, 2017.

[41] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017.

[42] Jian-Guang Lou, Qiang Fu, Shengqi Yang, Ye Xu, and Jiang Li. Mining invariants from console logs for system problem detection. In *Proceedings of the 2010 USENIX Conference on USENIX Annual Technical Conference*, USENIXATC'10, page 24, USA, 2010. USENIX Association.

[43] Andrei Manolache, Florin Brad, and Elena Burceanu. DATE: Detecting anomalies in text via self-supervision of transformers. In Kristina Toutanova, Anna Rumshisky, Luke Zettlemoyer, Dilek Hakkani-Tur, Iz Beltagy, Steven Bethard, Ryan Cotterell, Tanmoy Chakraborty, and Yichao Zhou, editors, *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 267–277, Online, June 2021. Association for Computational Linguistics.

[44] Michael McCloskey and Neal J Cohen. Catastrophic interference in connectionist networks: The sequential learning problem. In *Psychology of learning and motivation*, volume 24, pages 109–165. Elsevier, 1989.

[45] 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. In *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI-19*, pages 4739–4745. International Joint Conferences on Artificial Intelligence Organization, 7 2019.

[46] Haibo Mi, Huaimin Wang, Yangfan Zhou, Michael Rung-Tsong Lyu, and Hua Cai. Toward fine-grained, unsupervised, scalable performance diagnosis for production

cloud computing systems. *IEEE Transactions on Parallel and Distributed Systems*, 24(6):1245–1255, 2013.

[47] Sasho Nedelkoski, Jasmin Bogatinovski, Alexander Acker, Jorge Cardoso, and Odej Kao. Self-attentive classification-based anomaly detection in unstructured logs. In *2020 IEEE International Conference on Data Mining (ICDM)*, pages 1196–1201, 2020.

[48] Kim Anh Nguyen, Sabine Schulte im Walde, and Ngoc Thang Vu. Integrating distributional lexical contrast into word embeddings for antonym-synonym distinction. In Katrin Erk and Noah A. Smith, editors, *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 454–459, Berlin, Germany, August 2016. Association for Computational Linguistics.

[49] Adam Oliner and Jon Stearley. What supercomputers say: A study of five system logs. In *37th annual IEEE/IFIP international conference on dependable systems and networks (DSN'07)*, pages 575–584. IEEE, 2007.

[50] German I. Parisi, Ronald Kemker, Jose L. Part, Christopher Kanan, and Stefan Wermter. Continual lifelong learning with neural networks: A review. *Neural Networks*, 113:54–71, 2019.

[51] Jiaxing Qi, Shaohan Huang, Zhongzhi Luan, Shu Yang, Carol J. Fung, Hailong Yang, Depei Qian, Jing Shang, Zhiwen Xiao, and Zhihui Wu. Loggpt: Exploring chatgpt for log-based anomaly detection. In *IEEE International Conference on High Performance Computing & Communications, Data Science & Systems, Smart City & Dependability in Sensor, Cloud & Big Data Systems & Application, HPCC/DSS/SmartCity/DependSys 2023, Melbourne, Australia, December 17-21, 2023*, pages 273–280. IEEE, 2023.

[52] Jiaxing Qi, Zhongzhi Luan, Shaohan Huang, Carol Fung, Hailong Yang, and Depei Qian. Spikelog: Log-based anomaly detection via potential-assisted spiking neuron network. *IEEE Transactions on Knowledge and Data Engineering*, pages 1–15, 2023.

[53] Karsten Roth, Latha Pemula, Joaquin Zepeda, Bernhard Schölkopf, Thomas Brox, and Peter Gehler. Towards total recall in industrial anomaly detection. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 14318–14328, 2022.

[54] Lukas Ruff, Robert Vandermeulen, Nico Goernitz, Lucas Deecke, Shoaib Ahmed Siddiqui, Alexander Binder, Emmanuel Müller, and Marius Kloft. Deep one-class classification. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International*

Conference on Machine Learning, volume 80 of *Proceedings of Machine Learning Research*, pages 4393–4402. PMLR, 10–15 Jul 2018.

[55] Lukas Ruff, Robert Vandermeulen, Nico Goernitz, Lucas Deecke, Shoaib Ahmed Siddiqui, Alexander Binder, Emmanuel Müller, and Marius Kloft. Deep one-class classification. In *International conference on machine learning*, pages 4393–4402. PMLR, 2018.

[56] Gerard Salton and Christopher Buckley. Term-weighting approaches in automatic text retrieval. *Information processing & management*, 24(5):513–523, 1988.

[57] Xiuyao Song, Mingxi Wu, Christopher Jermaine, and Sanjay Ranka. Conditional anomaly detection. *IEEE Transactions on Knowledge and Data Engineering*, 19(5):631–645, 2007.

[58] Albin Soutif-Cormerais, Antonio Carta, Andrea Cossu, Julio Hurtado, Vincenzo Lomonaco, Joost van de Weijer, and Hamed Hemati. A comprehensive empirical evaluation on online continual learning. *2023 IEEE/CVF International Conference on Computer Vision Workshops (ICCVW)*, pages 3510–3520, 2023.

[59] Liang Tang, Tao Li, and Chang-Shing Perng. Logsig: generating system events from raw textual logs. In *Proceedings of the 20th ACM International Conference on Information and Knowledge Management*, CIKM '11, page 785–794, New York, NY, USA, 2011. Association for Computing Machinery.

[60] R. Vaarandi. A data clustering algorithm for mining patterns from event logs. In *Proceedings of the 3rd IEEE Workshop on IP Operations Management (IPOM 2003) (IEEE Cat. No.03EX764)*, pages 119–126, 2003.

[61] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[62] Yi Wan, Yilin Liu, Dong Wang, and Yujin Wen. Glad-paw: Graph-based log anomaly detection by position aware weighted graph attention network. In *Advances in Knowledge Discovery and Data Mining: 25th Pacific-Asia Conference, PAKDD 2021, Virtual Event, May 11–14, 2021, Proceedings, Part I*, page 66–77, Berlin, Heidelberg, 2021. Springer-Verlag.

[63] Dong Wang, Ning Ding, Piji Li, and Haitao Zheng. CLINE: Contrastive learning with semantic negative examples for natural language understanding. In Chengqing Zong, Fei

Xia, Wenjie Li, and Roberto Navigli, editors, *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 2332–2342, Online, August 2021. Association for Computational Linguistics.

[64] Xuheng Wang, Jiaxing Song, Xu Zhang, Junshu Tang, Weihe Gao, and Qingwei Lin. Logonline: A semi-supervised log-based anomaly detector aided with online learning mechanism. In *2023 38th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 141–152, 2023.

[65] Jason Wei and Kai Zou. EDA: Easy data augmentation techniques for boosting performance on text classification tasks. In Kentaro Inui, Jing Jiang, Vincent Ng, and Xiaojun Wan, editors, *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 6382–6388, Hong Kong, China, November 2019. Association for Computational Linguistics.

[66] Thorsten Wittkopp, Alexander Acker, Sasho Nedelkoski, Jasmin Bogatinovski, Dominik Scheinert, Wu Fan, and Odej Kao. A2log: Attentive augmented log anomaly detection. In *55th Hawaii International Conference on System Sciences, HICSS 2022, Virtual Event / Maui, Hawaii, USA, January 4-7, 2022*, pages 1–10. ScholarSpace, 2022.

[67] Yongzheng Xie, Hongyu Zhang, and Muhammad Ali Babar. Loggd: Detecting anomalies from system logs with graph neural networks. In *2022 IEEE 22nd International conference on software quality, reliability and security (QRS)*, pages 299–310. IEEE, 2022.

[68] 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*, pages 117–132, 2009.

[69] Rakesh Bahadur Yadav, P Santosh Kumar, and Sunita Vikrant Dhavale. A survey on log anomaly detection using deep learning. In *2020 8th International Conference on Reliability, Infocom Technologies and Optimization (Trends and Future Directions) (ICRITO)*, pages 1215–1220, 2020.

[70] Lin Yang, Junjie Chen, Zan Wang, Weijing Wang, Jiajun Jiang, Xuyuan Dong, and Wenbin Zhang. Semi-supervised log-based anomaly detection via probabilistic label estimation. In *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*, pages 1448–1460, 2021.

[71] Ding Yuan, Haohui Mai, Weiwei Xiong, Lin Tan, Yuanyuan Zhou, and Shankar Pasupathy. Sherlog: error diagnosis by connecting clues from run-time logs. In *Proceedings of the Fifteenth International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS XV, page 143–154, New York, NY, USA, 2010. Association for Computing Machinery.

[72] Seongjun Yun, Minbyul Jeong, Raehyun Kim, Jaewoo Kang, and Hyunwoo J Kim. Graph transformer networks. *Advances in neural information processing systems*, 32, 2019.

[73] Shengming Zhang, Yanchi Liu, Xuchao Zhang, Wei Cheng, Haifeng Chen, and Hui Xiong. Cat: Beyond efficient transformer for content-aware anomaly detection in event sequences. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, KDD '22, page 4541–4550, New York, NY, USA, 2022. Association for Computing Machinery.

[74] Tianzhu Zhang, Han Qiu, Gabriele Castellano, Myriana Rifai, Chung Shue Chen, and Fabio Pianese. System log parsing: A survey. *IEEE Transactions on Knowledge and Data Engineering*, 35(8):8596–8614, 2023.

[75] Xu Zhang, Yong Xu, Qingwei Lin, Bo Qiao, Hongyu Zhang, Yingnong Dang, Chunyu Xie, Xinsheng Yang, Qian Cheng, Ze Li, et al. 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*, pages 807–817, 2019.

[76] Jieming Zhu, Shilin He, Pinjia He, Jinyang Liu, and Michael R Lyu. Loghub: A large collection of system log datasets for ai-driven log analytics. In *2023 IEEE 34th International Symposium on Software Reliability Engineering (ISSRE)*, pages 355–366. IEEE, 2023.