

SentiLog: Anomaly Detecting on Parallel File Systems via Log-based Sentiment Analysis

Akhil Gorthi Bala Sai
Computer Science
Florida State University
Tallahassee, FL, USA
ag23bu@fsu.edu

Avaneeshakrishna Shastry
Chakracodi
Computer Science
Florida State University
Tallahassee, FL, USA
ac23bi@fsu.edu

Sohail Uddin Syed
Computer Science
Florida State University
Tallahassee, FL, USA
ss23cj@fsu.edu

ABSTRACT

Parallel File Systems are growing at a very rapid pace in both scale and complexity which makes them vulnerable to various failures or anomalies and identifying these anomalies during runtime is hence extremely important for both users and administrators. Traditional log analysis methods struggle with the large volume and irregular nature of PFS logs. To overcome these problems, we have SentiLog [1], a new approach to analyzing PFS logs for detecting anomalies. SentiLog introduces a groundbreaking approach by applying sentiment analysis via the Long short-term memory network (LSTM) deep learning method to parallel file systems (PFS) logs. SentiLog addresses these challenges by training a sentiment-based natural language model on the logging-relevant source code from multiple PFSes, allowing it to understand the implicit semantic information embedded by developers. Preliminary results indicate that SentiLog outperforms existing solutions in accurately predicting anomalies in two major PFSes, Lustre and BeeGFS, marking the first successful application of sentiment analysis for this purpose.

1 INTRODUCTION

This project endeavors to advance anomaly detection within Parallel File Systems (PFS) by employing log-based sentiment analysis. The initiative addresses inherent challenges in the scale and intricacy of PFS logs, essential components of high-performance computing (HPC) platforms, prone to various malfunctions and anomalies [1]. Conventional anomaly detection methodologies falter under the substantial volume and inconsistencies presented by PFS logs. These logs serve as vital records of operational data crucial for troubleshooting. Nevertheless, the extensive quantity and non-uniform nature of these logs significantly

complicate the effectiveness of traditional log analysis techniques. Further complexity arises from discrepancies in logging procedures across different systems, complicating the detection and rectification of runtime anomalies. It is difficult to apply conventional log analysis effectively. This is exacerbated by the variability in logging practices across different systems, which can hinder the identification and resolution of anomalies during runtime.

In response to these challenges, SentiLog is, a novel method that applies sentiment analysis to PFS log data to detect anomalies. Diverging from traditional methods that primarily rely on rule-based or supervised learning approaches [2, 3], SentiLog leverages a natural language processing (NLP) model trained on PFS source code. This model discerns the emotional undertones in log statements, indicating either standard operations or potential issues. Initial evaluations suggest that SentiLog surpasses existing leading log analysis techniques, offering a more effective and dependable means for anomaly detection in complex and extensive file systems. By addressing the unique challenges of PFS log analysis, SentiLog marks a significant and innovative contribution to the field of high-performance computing.

2 LITERATURE SURVEY

We conducted a very extensive literature survey of papers related to Parallel File Systems and High Performance Computing. The insights gained from the survey played a very important role towards the development of the project and exponentially increased our understanding of the same.

[15] In this paper the authors present PFault, a rigorous and comprehensive framework which is designed to analyze the reliability of high-performance file systems(PFSes), a key

aspect that is very often overlooked because of the focus on local storage system reliability. PFault employs different fault models to emulate the failure state of each storage device in the target PFS which in turn facilitates systematic analysis of its recoverability under different faults. This study has been applied Lustre, a widely used PFS which revealed instances where its checking and repairing utility, LFSCK, exhibited unexpected issues. PFault further identified one more problem related to resource leaks which impacted Lustre's internal namespace and storage space. It also highlighted areas for potential improvement in PFS reliability for high-performance computing. To overcome different challenges PFault leverages two key observations: it highly focuses on on-drive persistent states for fault emulation and on separating the global PFS layer across multiple nodes from the local system layer. By doing so, it enables effective and efficient fault analysis with minimal disturbance. The fault models considered in PFault include whole device failure, global inconsistency, and network partitioning.

[16] In this paper the authors introduce a new concept called LogLens, an advanced real-time log analysis system designed to automate the detection of anomalies in production applications without requiring extensive knowledge about the target system or detailed user specifications. By using unsupervised machine learning techniques, LogLens identifies patterns within application logs and facilitates the creation of sophisticated log analytics applications beyond the basic indexing and searching capabilities of existing systems. It is unique for its minimum reliance in user input and its ability to adapt to new log formats and patterns automatically. It also classifies anomalies into stateless and stateful categories. Stateless anomalies are detected through the analysis of individual log entries and on the other hand stateful anomalies are identified by examining sequences of logs for deviations from the norm. It employs a stateless log parsing algorithm that significantly outperforms traditional tools like Logstash in speed, as well as a stateful algorithm capable of detecting anomalies in operational workflows by linking related log entries. LogLens has been implemented in a commercial log analysis solution, significantly reducing the time required for troubleshooting operational problems in large-scale industrial environments.

[17] In this study the authors introduce LogAnomaly, an innovative framework that is designed to detect anomalies in computer system logs more accurately by modeling log

streams as natural language sequences. Traditional methods often rely on indexes, leading to frequent false alarms due to a lack of semantic analysis of the logs. LogAnomaly overcomes this by utilizing something called "template2vec", a new technique that captures the hidden semantic information in log templates hence allowing them to simultaneously detect sequential and quantitative anomalies, something that was not by previous approaches. Additionally, LogAnomaly can also handle new log templates that appear between model retraining periods, therefore avoiding false alarms that fail existing methods. Automatic anomaly detection in logs is very important and is highlighted by the complexity of modern services, where a single anomaly can affect millions of users. Manual detection is not only prone to errors but also increasingly infeasible due to the sheer volume of logs being generated. LogAnomaly addresses these challenges by utilizing the semi-structured nature of logs i.e., treating them similarly to natural language for more effective anomaly detection. On evaluation of LogAnomaly on public log datasets, it demonstrated its superiority over existing log-based anomaly detection methods.

[18] In this paper the authors introduce LogCluster, a method developed in collaboration with Microsoft to enhance problem identification in large-scale online service systems by clustering logs. This is designed to handle the vast amounts of data produced daily by these systems, which can be overwhelming and impractical for engineers to analyze manually. It not only groups similar log sequences together but also incorporate a knowledge base to identify repeating issues, significantly reducing the manual effort needed to diagnose problems. The authors have conducted different experiments with Hadoop applications and Microsoft online services to demonstrate LogCluster's effectiveness and superiority over existing methods like the one proposed by Shang et al. at ICSE 2013. They have successfully implemented LogCluster in various Microsoft services. This approach is distinguished by its ability to cluster log sequences intelligently while considering the diversity and importance of log messages, by using previous knowledge of issues to avoid redundant log examinations.

[19] In this paper the authors introduce a novel decision tree learning method aimed at diagnosing failures within large Internet sites by analyzing runtime properties of each request. It uses machine learning and data mining techniques to effectively pinpoint the causes of failures by training decision trees on request traces from periods marked by

user-visible failures. This method was tested on real-world failures from eBay wherein the algorithm successfully identified most true failure causes with minimal false positives. The authors also explored the practical applications of simplified decision trees in eBay's live environment and provided a comparison in between manual and automated systems. The study highlighted their algorithms adaptation of decision trees for failure diagnosis and its successful application in a high-traffic production environment, demonstrating its potential to significantly improve system availability through efficient error source identification. This method uses detailed logging to trace each request's journey through the system and uses machine learning to analyze vast quantities of request data for potential errors. The system overall meets essential criteria for practical deployment which includes a high diagnosis rate, minimal false positives, robustness against noise, and rapid processing for near real-time utility. It also offers insights into adapting statistical learning approaches for automated failure diagnosis and its implementation in managing the complexities of modern Internet services.

[20] This paper presents an innovative deep learning-based sentiment analysis method for forensic investigations of log messages in forensic timelines. The approach aims to identify events of interest through their sentiment, particularly emphasizing negative sentiments, to simplify the traditional manual examination process of timelines for detecting suspicious activities. The methodology includes preprocessing of event log data and the use of word embedding for textual representation. It employs dual attention layers—context and content attention—to precisely identify sentiments associated with specific log aspects. Events highlighted by negative sentiments in the forensic timeline enable quick identification of potential issues. Tested with public forensic datasets, the method demonstrated notable performance. This breakthrough significantly reduces the workload on forensic investigators by automating the identification and highlighting of crucial events, thereby enhancing the efficiency of the investigative process. Looking ahead, the researchers plan to integrate this sentiment-focused approach with statistical anomaly detection for a more comprehensive analysis and to incorporate the method into forensic timeline visualization tools for better user interaction. This research represents a significant advancement in digital forensics and deep learning, illustrating the effectiveness of sentiment analysis in forensic timeline analysis and offering a practical solution to a widespread challenge in forensic investigations.

[21] This paper addresses the detection of sophisticated cyber threats, such as Advanced Persistent Threats (APT), challenging traditional security defenses. Focusing on early-stage detection, it introduces a novel framework leveraging belief propagation and graph theory to analyze DNS and web proxy logs for signs of APT infections. The methodology's effectiveness is validated on two large datasets, where it accurately identifies malicious domains involved in simulated and real APT attacks, uncovering numerous threats undetected by conventional security tools. This achievement is pivotal, given the rapid evolution and design of malware to circumvent existing protections. Providing a mix of unsupervised and supervised learning, the paper innovates in detecting infections when definitive evidence is scarce, by exploiting patterns in domain communications during the attack lifecycle. This strategy departs from traditional reliance on specific malware signatures or samples, offering a fresh perspective on cyber defense. Significant contributions include a graph-theoretic detection framework for early-stage infections, a bespoke detector for enterprise command-and-control communications, and validation of these approaches with real-world data. The proposed techniques promise substantial improvements in early threat detection, thereby enhancing organizational security posture and reducing risk.

[22] With an emphasis on the Lustre file system, this paper presents a novel methodology for assessing large-scale parallel file systems' failure handling capabilities. To evaluate the system's recovery performance, the framework simulates realistic failure conditions by using virtual devices to record disk I/O commands on storage nodes. In the context of large-scale parallel file systems, where vulnerabilities may result in data loss during failure events, the authors stress the significance of researching failure recovery. The suggested approach is based on findings that failure events affect on-drive permanent states, enabling efficient emulation on virtual devices, and it attempts to minimize disruption to the local storage stack.

The framework's capacity to reveal core I/O behaviours under a range of workloads and failure scenarios is demonstrated by the study's early findings, which are based on a prototype for the Lustre file system. The framework's ability to identify weaknesses offers an essential starting point for additional research and development of large-scale parallel file systems. Future possibilities are outlined in the paper's conclusion, including developing the framework to simulate additional nodes and investigating the robustness of alternative parallel file systems, such as Ceph and PVFS.

[23] The paper discusses the problem of establishing acceptable log levels for logging statements in software systems. It begins with a manual examination of log level features to uncover links between syntactic context, log messages, and log levels. The findings show that log levels further apart in order have more different properties. Building on these findings, the research presents a deep-learning-based strategy for predicting log levels that takes advantage of log levels' ordinal nature. Syntactic context, log message, and combination features are examples of extracted features. The deep learning framework consists of an embedding layer, a bidirectional LSTM layer, and an output layer that converts the ordinal vector output to a real log level. The approach is tested on nine large-scale open-source systems and outperforms baseline approaches in predicting log levels. Furthermore, the study yields promising results for cross-system ideas. The paper presents an automated approach to log level suggestion, demonstrating the potential of using logging statement attributes to help developers make intelligent logging decisions. The proposed method enhances existing methodologies and provides insights into future research paths in logging support for developers.

3 BACKGROUND AND RELATED WORK

3.1 Background

Anomaly detection in large-scale systems has been a critical area of research, with traditional methods primarily falling into three categories: rule-based, supervised learning, and unsupervised learning. Rule-based methods, illustrated by systems like LogLens [2] and PerfAugur [3], rely on predefined rules to identify deviations from normal system behavior. Supervised learning approaches, as demonstrated by DeepLog and LogAnomaly, involve training models on labeled data to recognize patterns indicative of anomalies. On the other hand, unsupervised learning methods, such as Principal Component Analysis (PCA) [4, 5] and Invariant Mining, detect anomalies by identifying irregularities or deviations from expected patterns within log data.

However, the application of these conventional techniques encounters substantial hurdles when confronted with the unique characteristics of Parallel File Systems (PFS). PFS, like Lustre, are renowned for generating an immense volume of logs due to their scale and complexity. This sheer volume poses challenges for accurately labeling logs, a task essential

for the effectiveness of rule-based and supervised learning methods. Moreover, the custom logging mechanisms often employed in PFS, instead of standardized libraries like Log4J [6] or SLF4J [7], contribute to the irregularity in log formats and identifiers. Consequently, the lack of consistency in log structure impedes the performance of unsupervised learning methods, which rely on identifying consistent patterns or identifiers to discern anomalies within log sequences.

3.2 Related Work

Existing methods of log analysis for detecting system anomalies consist of :

Rule-based methods that use expert-crafted rules to identify anomalies by flagging log entries that don't match predefined patterns. While effective in certain contexts, these methods are unsuitable for PFSes due to the lack of regularity in the logs and the requirement for extensive expert knowledge.

Supervised learning methods train classifiers using labeled log data. Traditional machine learning algorithms like decision trees and SVMs have been used in the past, and more recently, deep learning approaches utilizing LSTM or RNN networks have emerged. The problem with PFSes is the scarcity of labeled logs, which makes supervised methods impractical. This has led to the development of SentiLog, which utilizes the source code of PFSes to train its model, circumventing the need for labeled logs.

Unsupervised learning methods do not require labeled data and often rely on detecting invariants or sequences in log entries. Techniques like Invariant Mining look for consistent patterns across log events, such as matching counts of 'open-file' and 'close-file' operations. However, the irregular nature of PFS logs, which often lack such patterns or invariants, limits the effectiveness of these methods for PFS environments.

Some recent approaches have applied sentiment analysis to log-based anomaly detection, training, and testing models directly on runtime logs. This process, however, is labor-intensive as it requires extensive labeling of the logs. SentiLog differentiates itself by leveraging the source code to train its models, thus avoiding the need for labeling runtime logs and making it a suitable tool for the complex and irregular nature of PFS logs.

These challenges necessitate the exploration of innovative approaches tailored to the intricacies of PFS logs. In response, researchers have developed SentiLog, a novel method that leverages sentiment analysis techniques

commonly utilized in Natural Language Processing (NLP). Sentiment analysis, traditionally applied for understanding emotions conveyed in textual data, is adapted in SentiLog to interpret the emotional context embedded within log messages. By analyzing the tone and sentiment expressed by developers in log statements, SentiLog offers a promising alternative to traditional log analysis methods, specifically addressing the limitations encountered within the context of PFS.

SentiLog represents a paradigm shift in anomaly detection within large-scale systems, offering a nuanced understanding of system behavior beyond mere structural analysis of log data. By incorporating emotional context, SentiLog opens avenues for more nuanced anomaly detection, potentially capturing subtle deviations that might escape traditional methods. Furthermore, its adaptability to the unique challenges posed by PFS positions it as a valuable tool for enhancing the reliability and efficiency of system monitoring and maintenance in complex computing environments.

```
Lustre
00000100:00000000:0:0:1607448618.327577:0:2290:0:(recover.c:58:ptlrpc_initiate_recovery
()) lustre-05T0000_UUID: starting recovery
00000100:00000000:0:0:1607448618.327580:0:2290:0:(import.c:681:ptlrpc_connect_import())
ffffa139cab87800 lustre-05T0000_UUID: changing import state from DISCONN to CONNECTING
00000100:00000000:0:0:1607448618.327589:0:2290:0:(import.c:524:import_select_connection
()) lustre-05T0000-osc-MDT0000: connect to NID 10.0.0.0@tcp last attempt 4296114409
00000100:00000000:0:0:1607448618.327593:0:2290:0:(import.c:568:import_select_connection
()) lustre-05T0000-osc-MDT0000: tried all connections, increasing latency to 11s

HDFS
081109 203518 143 INFO dfs.DataNode$DataXceiver: Receiving block blk_-
1608999687919862906 src: /10.250.19.102:54106 dest: /10.250.19.102:50010
081109 203518 35 INFO dfs.FSNamesystem: BLOCK* NameSystem.allocateBlock:
/mnt/hadoop/mapred/system/job_200811092030_0001/job.jar. blk_-1608999687919862906
081109 203519 143 INFO dfs.DataNode$DataXceiver: Receiving block blk_-
1608999687919862906 src: /10.250.10.6:40524 dest: /10.250.10.6:50010
081109 203519 145 INFO dfs.DataNode$PacketResponder: PacketResponder 1 for block blk_-
1608999687919862906 terminating
```

Figure 1: Two log snippets of Lustre and HDFS

Figure 1 displays two log snippets from two storage systems (Lustre and HDFS). HDFS logs show a unique block ID across many entries (indicated in red). A unique ID is essential for log analysis as it connects several occurrences into a sequence to find patterns.

4 OUR SOLUTION AND CHALLENGES FACED

4.1 Our Solution

SentiLog presents a pioneering approach to anomaly detection within Parallel File Systems (PFS) by integrating sentiment analysis into log data processing. This innovative method capitalizes on the understanding that developers

often imbue their logging statements with different tones to reflect the system's state employing negative tones to signify errors and neutral or positive tones for routine operations. Leveraging this inherent emotional context, SentiLog delves into the source code generating these logs, particularly focusing on the embedded logging statements. By doing so, SentiLog extracts semantic insights directly tied to system behaviors as conveyed by the developers themselves.

One of SentiLog's primary strengths lies in its adaptability to the unique challenges posed by PFS logs. Unlike traditional methods, which may struggle to cope with the immense volume and irregularity of PFS logs, SentiLog thrives by prioritizing the sentiment embedded within logging statements rather than merely analyzing events or patterns. This nuanced approach enables SentiLog to discern the significance of each log entry more accurately, thereby enhancing anomaly detection efficacy. Additionally, SentiLog's methodology alleviates the reliance on extensive labeled datasets, a notable challenge in the domain of PFS anomaly detection. Instead, by training across multiple PFS source codes, SentiLog cultivates a generalized model capable of robust performance across diverse systems, even those not explicitly included in the training regimen.

By delving into the emotional nuances of log data, SentiLog augments traditional log analysis techniques, offering a deeper understanding of system behavior. This enhancement not only improves anomaly detection accuracy but also enhances the interpretability of log data, facilitating more informed decision-making in system maintenance and troubleshooting. Furthermore, SentiLog's ability to generalize across different PFS environments underscores its potential as a versatile tool for enhancing the reliability and efficiency of large-scale system management.

4.2 Challenges

In the initial stages of our project, we encountered several challenges related to setting up TensorFlow due to conflicts with existing library versions. Resolving these environment setup issues required significant time and effort. Additionally, the necessity for GPU support introduced further complexities. We experienced difficulties in configuring CUDA properly on our device to ensure that it was recognized and utilized for neural network model training.

We also considered using Google Colab as an alternative platform. However, as free users, we were constrained by the

limited computational resources available, which impacted our ability to perform extensive experiments

Hyperparameter tuning presented another significant challenge. Given our limited GPU capabilities, we conducted multiple tests to determine the optimal parameters that would yield the best results. This process involved an iterative approach of adjusting and evaluating different combinations of hyperparameters to find the most effective settings under our resource constraints. This phase was crucial as it directly influenced the performance and efficiency of our neural network models

The development and deployment of SentiLog encountered several more formidable hurdles, each requiring innovative solutions:

Lack of Labeled Data: Traditional supervised learning methods rely on abundant labeled data, which is notably scarce in the context of Parallel File Systems (PFS) due to the absence of adequately labeled runtime logs. To surmount this challenge, SentiLog ingeniously leverages the inherent labeling present within logging levels, such as errors or debug information, directly encoded within the source code. This circumvents the need for labor-intensive manual labeling, making supervised learning approaches feasible for anomaly detection in PFS.

Generalization Across Diverse Systems: Ensuring the efficacy of SentiLog across a spectrum of PFS implementations necessitates overcoming the variability inherent in logging practices and system architectures. To achieve this, the sentiment analysis model underlying SentiLog undergoes training on a diverse array of PFS source codes. By exposing the model to a broad spectrum of logging practices and anomalies, it acquires a comprehensive understanding of system behavior, enhancing its ability to generalize across diverse PFS environments.

Interpretation of Natural Language in Logs: The technical and sparse nature of language used in system logs poses a significant challenge for interpreting natural language within logs using NLP techniques. Developing an NLP model capable of accurately understanding and categorizing sentiments within this specialized context demands extensive tuning and experimentation. SentiLog pioneers this effort by refining NLP methodologies to effectively decipher the nuanced language of system logs, ensuring accurate sentiment analysis.

SentiLog heralds a groundbreaking advancement in anomaly detection within Parallel File Systems by

harnessing the subtleties of sentiment analysis, traditionally employed in natural language processing. By scrutinizing the tone and context embedded within logging statements crafted by developers to delineate operational states SentiLog obviates the reliance on painstakingly curated labeled datasets, a bottleneck in conventional approaches. Moreover, its capacity to generalize across diverse PFS architectures without necessitating system-specific fine-tuning signifies a significant leap forward.

SentiLog represents a pioneering step in anomaly detection within Parallel File Systems by harnessing the nuances of sentiment analysis, traditionally used in natural language processing. This approach directly addresses the challenges posed by the sheer volume and complexity of PFS logs. By analyzing the tone and context of logging statements encoded by developers to signify operational states SentiLog circumvents the need for laboriously labeled data sets, which are a significant bottleneck in conventional methods.

Furthermore, its ability to generalize across various PFS architectures without the need for system-specific tuning marks a significant advancement. Despite the challenges of adapting NLP techniques to the specialized language of system logs and ensuring robustness across diverse systems, SentiLog demonstrates a promising new direction for enhancing reliability and monitoring in high-performance computing environments. This innovative integration of sentiment analysis into system log analysis could potentially set a new standard for anomaly detection methodologies in complex IT infrastructures.

5 DESIGN/ARCHITECTURE

5.1 Framework

SentiLog is structured around a multi-stage process that begins with the collection of logging statements directly from the source code of various Parallel File Systems (PFS). This initial stage involves identifying and extracting logs that are naturally labeled by their logging levels (e.g., errors, warnings, debug statements), which serve as the foundation for the training dataset. The next stage involves preprocessing these logs to refine the data by removing irrelevant elements such as timestamps and extraneous symbols, standardizing text, and reducing words to their base forms to enhance the model's learning efficiency.

Following preprocessing, the core of SentiLog's architecture is its sentiment analysis model, which

employs advanced natural language processing techniques. The model is trained to discern the sentiment of log messages—classifying them into categories such as normal or anomalous based on the tone and context indicated by the logging levels. This training utilizes a bidirectional Long Short-Term Memory (BiLSTM) network, which is particularly adept at understanding context in sequence data, thereby providing a robust mechanism for analyzing the sentiment expressed in log entries.

5.2 Technological Stack

The technological stack of SentiLog is primarily built on Python, leveraging various libraries and frameworks that facilitate machine learning and natural language processing. Key components include TensorFlow and Keras for constructing and training the deep learning models. The BiLSTM [7] networks are integral to the model's ability to effectively process and learn from the sequential data present in log files. For handling the large datasets typical of PFS logs, high-performance computing resources are utilized, ensuring that the data processing and model training phases are both efficient and scalable.

In practice, SentiLog's architecture is designed to be modular, allowing for easy adaptation and updates to individual components such as the sentiment analysis model or the preprocessing steps. This modularity ensures that SentiLog can keep pace with the evolving landscapes of both PFS technology and machine learning methodologies.

6 METHODOLOGY

6.1 Workflow

The working of SentiLog consists of four stages:

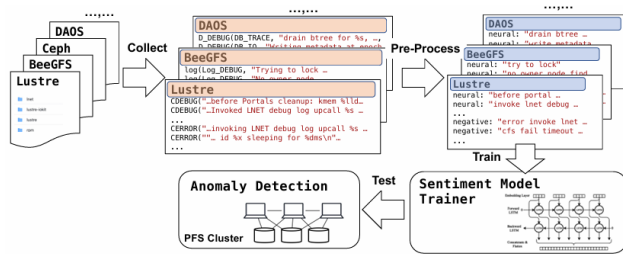


Figure 2: The overall workflow of SentiLog

6.1.1 Logging Statements Collecting

It is the initial process of collecting logging statements for training the SentiLog model, a tool designed to detect anomalies in parallel file systems (PFSes) using sentiment analysis. To gather the necessary data, SentiLog requires two key inputs from users: a list of open-source PFSes along with their corresponding git URLs, and a list of specific keywords related to each PFS's logging mechanisms. These keywords are either derived from official documentation or identified manually through an examination of the source code.

Figure 2 describes the initial process of collecting logging statements for training the SentiLog model, a tool designed to detect anomalies in parallel file systems (PFSes) using sentiment analysis. To gather the necessary data, SentiLog requires two key inputs from users: a list of open-source PFSes along with their corresponding git URLs, and a list of specific keywords related to each PFS's logging mechanisms. These keywords are either derived from official documentation or identified manually through an examination of the source code.

Using these inputs, SentiLog periodically pulls the latest source code from the provided URLs and uses the specified keywords to extract relevant logging statements. Each keyword is associated with a sentimental state (either negative or neutral) based on the logging level it represents, such as marking "CERROR" as negative due to its association with error logs, and "CDEBUG" as neutral.

The extracted logging statements are then labeled according to their sentimental state and compiled into raw datasets. These datasets undergo further preprocessing before being used to train the sentiment analysis model, forming the basis of SentiLog's anomaly detection capabilities.

6.1.2 Log Pre-Processing

This step is crucial for transforming technical log entries into a format more suitable for natural language processing (NLP). The pre-processing includes:

1. Converting all text to lowercase to maintain consistency.
2. Stemming words to their root forms to simplify and standardize the language.

3. Removing stop words, which are common but irrelevant words that do not contribute to sentiment analysis.
4. Normalizing texts to a standard form to reduce variability.
5. Eliminating noise such as format strings (e.g., '%s') and punctuation, which do not carry emotional information.

These modifications help refine the logs into a state that resembles natural language more closely, thereby improving the accuracy and stability of the sentiment analysis performed by SentiLog. However, we have already used pre-processed log data for our project. This process ensures that the data fed into the sentiment analysis model is clean and consistent, optimizing the model's performance in anomaly detection.

6.1.3 Sentiment Model Trainer

This consists of the training of the sentiment analysis model within the SentiLog framework. The model utilizes a bidirectional LSTM (BiLSTM) network, which is well-suited for analyzing sequences such as text. The BiLSTM in this setup features two layers, each containing 100 neurons, summing up to a total of 789,000 parameters. Training of this model is performed with batches of 64 examples, using the Adam optimization algorithm with a learning rate of 0.01.

For input preparation, each word in the logging statements is tokenized and represented using vectors derived from pre-trained GloVe embeddings. The output from the BiLSTM network is a two-dimensional vector that categorizes the sentiment of the log entries as either negative or neutral. This structured approach allows the model to effectively learn and predict the sentiment based on the nuanced patterns found in the processed log data.

6.1.4 Anomaly Detection

This consists of the application of the trained SentiLog model for anomaly detection in runtime logs from various parallel file systems (PFSes). The process is direct and efficient. As new logs are generated by a PFS, they are collected, pre-processed (as previously described), and tokenized. These tokenized logs serve as input for the BiLSTM network, which has been trained to analyze sentiment. The network assesses each log entry and determines whether it signals an anomaly based on its sentiment prediction. This method leverages sentiment analysis to discern patterns

indicative of potential issues within the PFS, thus facilitating real-time anomaly detection.

To conclude we can say that the workflow of SentiLog is structured into several distinct phases such as:

1. **Data Collection:** Logging statements are extracted from the source code of multiple PFS implementations. These logs are pre-labeled based on their logging levels, providing a natural classification of sentiments.
2. **Preprocessing:** The collected logs undergo preprocessing to transform the raw log data into a more uniform format suitable for NLP analysis. This includes tokenization, normalization, and removal of irrelevant syntactical elements. The goal is to refine the data to focus on meaningful textual content that impacts sentiment analysis.
3. **Model Training:** With preprocessed data, the system trains the BiLSTM model to recognize and categorize the sentiments of the logs. The training process involves adjusting the neural network's parameters to minimize prediction errors, effectively learning from the nuances of the pre-labeled log data.
4. **Anomaly Detection:** Once trained, the model is deployed to analyze new log entries in real-time. It evaluates the sentiment of each log statement as it is generated, classifying it as normal or anomalous based on the learned patterns.
5. **Feedback and Adaptation:** SentiLog is designed to adapt over time. Feedback from the detection phase can be used to retrain the model, ensuring that it remains effective as the characteristics of the logs change due to updates in PFS software or shifts in system usage patterns.

SentiLog's implementation leverages several best practices in software engineering to ensure robustness and maintainability:

Modular Design: The system architecture is modular, allowing for individual components, such as the sentiment analysis model or data preprocessing modules, to be updated independently without affecting the rest of the system.

Continuous Integration/Continuous Deployment (CI/CD): This practice is utilized to automate the testing and deployment of new code changes, ensuring

that SentiLog can be continuously updated with minimal downtime.

Version Control: Extensive use of version control systems allows tracking of changes and collaboration among development teams, essential for managing the complex codebase of SentiLog.

Scalability: Special attention is given to scalability, both in terms of data processing and model training, to handle the extensive volumes of log data typically generated by PFS systems.

The implementation of SentiLog not only incorporates advanced machine learning techniques but also aligns with high standards of software development, ensuring that the system is both effective in anomaly detection and robust in handling the operational demands of large-scale PFS environments.

6.2 Bidirectional LSTM

Layer (type)	Output Shape	Param #
embedding (Embedding)	(None, 100, 100)	770000
bidirectional (Bidirectional)	(None, 100, 200)	160000
bidirectional_1 (Bidirectional)	(None, 200)	240000
dense (Dense)	(None, 400)	80400
dense_1 (Dense)	(None, 2)	802
Total params: 1,261,402		
Trainable params: 482,402		
Non-trainable params: 779,000		
Epoch 1/5		
1105/1105 [=====] - 140s 124m/step - loss: 0.3903 - accuracy: 0.8145 - val_loss: 0.5584 - val_accuracy: 0.7553		
Epoch 2/5		
1105/1105 [=====] - 135s 116m/step - loss: 0.3177 - accuracy: 0.8728 - val_loss: 0.4916 - val_accuracy: 0.7946		
...		
263/263 [=====] - 9s 33m/step		
Precision: 0.7934934441957148		
Recall: 0.6528620816770536		
F1 Score: 0.715247143266262		

Figure 3: BiLSTM Network Architecture

A Sequential model, starting with an embedding layer initialized with pre-trained GloVe embeddings. Includes two bidirectional LSTM layers, each with 100 units. It allows the model to learn from both past and future contexts.

Following the LSTM layers, there are two dense layers with 400 and 2 units respectively, with the final layer using SoftMax activation for binary classification. The model contains 2 Bi-directional LSTM layer with first layer returning sequences so that next layer will have full context of the sequence.

The working of the Bi-RNN Model is as follows:

It uses NumPy for numerical operations, TensorFlow for deep learning functionalities, Keras for building neural networks, and scikit-learn for evaluation metrics.

“load_glove_embeddings”: This function loads pre-trained Glove word embeddings from a file into a dictionary.

“create_embedding_matrix”: This function creates an embedding matrix for the words in the dataset using pre-trained word embeddings. It also adjusts the tokenizer's word index to match the new vocabulary.

“read_data”: This function reads data from files and assigns labels based on whether the data is normal or abnormal.

Data paths are defined for training and testing data files, as well as the path to the Glove embeddings file, training and testing data is read along with their corresponding labels and tokenization is done on the text data (both training and testing) to convert words into numerical indices.

Then the pre-trained Glove word embeddings are loaded and an embedding matrix for the dataset vocabulary is created. Next, the data is prepared where in text sequences are converted into padded sequences of fixed length to ensure uniform input size for the model.

Model Definition: The model architecture is defined using the Sequential API.

Embedding layer: Maps each word index to its corresponding Glove word embedding vector. The weights of this layer are initialized with pre-trained Glove embeddings and are kept fixed during training (‘trainable=False’).

Two layers of ‘Bidirectional LSTM’: Bidirectional LSTM layers process input sequences in both forward and backward directions, capturing contextual information effectively.

Two ‘Dense’ layers: Fully connected layers with ReLU activation functions.

Final ‘Dense’ layer: Output layer with SoftMax activation for binary classification (normal or abnormal).

Then the model is compiled with the Adam optimizer and sparse categorical cross-entropy loss function followed by training where the model is trained on the training data for a specified number of epochs, validating on the test data. Evaluation is done of the trained model on the test data using accuracy as the metric. The model is also evaluated using precision, recall, and F1 score.

6.3 Transformer Model

The working our Transformer Model for text classification is achieved using TensorFlow and Keras. Here's a breakdown:

1. Data Preparation: The code reads data from text files, preparing it for training and testing. Normal and abnormal logs are labeled accordingly.

2. Tokenization and Embedding Setup: Tokenization is performed using the Tokenizer class from Keras. Glove embeddings are loaded and used to create an embedding matrix.

3. Transformer Encoder Block: A custom transformer encoder block is defined, incorporating multi-head self-attention and feed-forward layers. The Transformer encoder block plays a crucial role in the Transformer model architecture, particularly in processing input sequences.

- Self-Attention Mechanism:** The encoder block utilizes self-attention to capture the relationships between different words in the input sequence. It allows the model to weigh the importance of each word based on its relevance to other words in the sequence, enabling the model to focus more on relevant parts of the input during processing.
- Multi-Head Attention:** By employing multiple attention heads, the encoder block can learn different attention patterns simultaneously. This multi-head mechanism enhances the model's ability to capture diverse relationships and dependencies within the input sequence, leading to improved representation learning.
- Feed-Forward Neural Network:** After processing the input sequence through the self-attention mechanism, the encoder block applies a feed-forward neural network (FFNN) to further refine the representations. This FFNN introduces non-linearity and helps the model capture more complex patterns in the data.
- Layer Normalization and Residual Connections:** Layer normalization and residual connections are employed within the encoder block to facilitate stable training and mitigate the vanishing gradient problem. These techniques enable the model to effectively propagate gradients during training, making it easier to train deep architectures like the Transformer.

The Transformer encoder block enables the model to efficiently process input sequences by leveraging self-attention mechanisms, multi-head attention, and feed-forward neural networks. It plays a key role in capturing dependencies within the data and learning meaningful representations for downstream tasks like text classification.

4. Building the Transformer Model: The model architecture is defined using Keras Functional API. It consists of embedding layers followed by multiple transformer encoder blocks, global average pooling, and dense layers with dropout.

5. Training the Model: The model is trained using the prepared data.

6. Evaluation: The trained model is evaluated on the test data using accuracy as well as precision, recall, and F1 score.

Layer (type)	Output Shape	Param #	Connected to
input_1 (InputLayer)	[(None, 100)]	0	[]
embedding (Embedding)	(None, 100, 100)	387600	['input_1[0][0]']
layer_normalization (LayerNormal- ization)	(None, 100, 100)	200	['embedding[0][0]']
multi_head_attention (MultiHead- attention)	(None, 100, 100)	103268	['layer_normalization[0][0]', 'layer_normalization[0][0]']
dropout (Dropout)	(None, 100, 100)	0	['multi_head_attention[0][0]']
tf.__operators__.add (TFOpLamb- da)	(None, 100, 100)	0	['dropout[0][0]', 'embedding[0][0]']
layer_normalization_1 (LayerNor- malization)	(None, 100, 100)	200	['tf.__operators__.add[0][0]']
dense (Dense)	(None, 100, 128)	12928	['layer_normalization_1[0][0]']
dropout_1 (Dropout)	(None, 100, 128)	0	['dense[0][0]']
dense_1 (Dense)	(None, 100, 100)	12900	['dropout_1[0][0]']
tf.__operators__.add_1 (TFOpLa- mbda)	(None, 100, 100)	0	['dense_1[0][0]', 'tf.__operators__.add[0][0]']
layer_normalization_2 (LayerNor- malization)	(None, 100, 100)	200	['tf.__operators__.add_1[0][0]']
multi_head_attention_1 (MultiH- eadAttention)	(None, 100, 100)	103268	['layer_normalization_2[0][0]', 'layer_normalization_2[0][0]']
dropout_2 (Dropout)	(None, 100, 100)	0	['multi_head_attention_1[0][0]']
tf.__operators__.add_2 (TFOpLa- mbda)	(None, 100, 100)	0	['dropout_2[0][0]', 'tf.__operators__.add_1[0][0]']
layer_normalization_3 (LayerNor- malization)	(None, 100, 100)	200	['tf.__operators__.add_2[0][0]']
dense_2 (Dense)	(None, 100, 128)	12928	['layer_normalization_3[0][0]']
dropout_3 (Dropout)	(None, 100, 128)	0	['dense_2[0][0]']
dense_3 (Dense)	(None, 100, 100)	12900	['dropout_3[0][0]']
tf.__operators__.add_3 (TFOpLa- mbda)	(None, 100, 100)	0	['dense_3[0][0]', 'tf.__operators__.add_2[0][0]']
global_average_pooling1d (Glob- alAveragePooling1D)	(None, 100)	0	['tf.__operators__.add_3[0][0]']

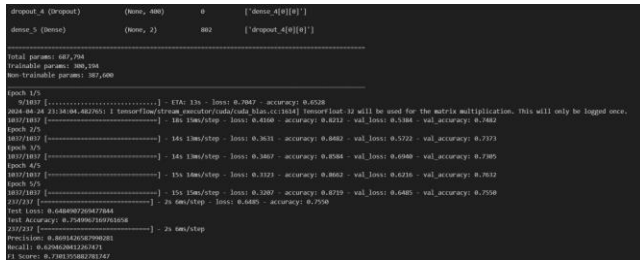


Figure 4: Transformer Model Architecture

Model Definition: Defines a single transformer encoder block. This block consists of multi-head self-attention and feed-forward layers, along with layer normalization and dropout.

Build Model: It defines the input layer, embedding layer (initialized with pre-trained embeddings), multiple transformer encoder blocks, and output layers.

7 ENVIRONMENT AND DATASETS

7.1 Development Environment

The development environment for SentiLog is primarily centered around Python, given its extensive support for data science and machine learning libraries. Key tools and libraries include:

TensorFlow, Keras, NumPy, Scikit-learn and Jupyter Notebook.

System Info:

- Windows 11 Home
- Processor: 11th Gen Intel(R) Core(TM) i7-11800H @ 2.30GHz 2.30 GHz
- Installed RAM: 16GB RAM.
- Graphics: NVIDIA GeForce RTX 3060 GPU 6GB Dedicated GPU Memory

7.2 Datasets

The datasets used in SentiLog include a mix of real and synthetic logs generated from multiple Parallel File Systems (PFS). These logs are essential for both training and testing the sentiment analysis model. Details of the datasets include:

Real Logs: Extracted from actual operational data of PFS like Lustre and BeeGFS. These logs are

preprocessed to extract meaningful patterns and sentiments as discussed in the implementation phase.

Synthetic Logs: Generated to simulate various scenarios and anomalies not sufficiently covered by real logs. This helps in enhancing the model's ability to generalize across different and unseen situations.

Pre-Labelled Data: Logs are pre-labeled based on their logging levels derived directly from the source code, which naturally indicates the sentiment (normal, warning, error) associated with each log entry.

Training Set: Comprises a large portion of the real and synthetic logs, used to train the model to recognize and classify the sentiment of log entries.

Train Datasets: Ceph, GlusterFS, DAOS, OrangeFS, BeeGFS, Hive.

```

number nod in the system
no owner node find for lost found directory do not
initialize components
component initialized
component running
component stopped
process retrievechunkswork
process retrievechunkswork
testdefaultconfigfile start
testdefaultconfigfile finish
number nodes
unable provide exceed quota ids
allow interfaces
load target numeric id mappings
initialize components
component initialized
start components
component running
extend list usable nics
component stopped

```

Figure 5: BeeGFS-debug-filter (positive sentiment)

```

fail get state for mirror
unknown mirror group id
could not update user quota data
could not update group quota data
error during read file with uids or gids
unable open file with uids or gids
give quotadatatype do not match the
give quotadatatype do not match the
give quotadatatype do not match the
give quotadatatype do not match the
error generate access acl for new directory
unable set exceed quota ids configuration problem detected
error deserializing directory default acl
error load directory default acl
error set file acl
error buffer too small
only meta storage nodetypes supported
could not get client stats from nodes
job complete with return code
could not download client list from management daemon
could not download metadata server list from management daemon

```

Figure 6: BeeGFS-error-filter (negative sentiment)

A selection of PFSes was made based on their prevalence among the HPC clusters in the Top500 list. For training purposes, the top four PFSes were chosen, while the bottom two, marked in gray, were reserved for testing the model. Each PFS's unique logging mechanisms are briefly listed, along with the number of debug and error log samples collected. Debug logs were labeled as 'neutral', and error logs as 'negative' for the purposes of training the sentiment analysis model. The training process took place on a computer with a 4-Core 2.2 GHz Intel i7 CPU and 16 GB of RAM, and it was completed in approximately 6 hours.

Testing Set: Distinct from the training data, used to evaluate the model's performance in predicting anomalies based on new log data.

Test Data sets: Lustre, HBase, HDFS.

The authors created a small set of labeled runtime logs specifically for testing due to the absence of publicly available labeled runtime logs for PFSes. Using PFAULT, a fault injection tool, they simulated faults in PFSes and collected the resultant log data. Logs produced before fault injection were labeled as normal, while logs produced after were labeled as abnormal by domain experts, based on their relevance to the injected faults and the presence of standard or custom error numbers.

The labeled data includes various fault models, encompassing a range of possible failure scenarios for a PFS cluster. Due to the limitations of PFAULT, only logs from Lustre and BeeGFS could be used for testing. The testing set reflects the different logging frequencies of the two PFSes, resulting in a total of 150,473 normal and 7,401 abnormal entries for Lustre, and 119 normal and 39 abnormal for BeeGFS.

8 TESTING AND EVALUATION

8.1 Testing

Testing and evaluation of SentiLog were designed to rigorously assess its performance and reliability in detecting anomalies within Parallel File Systems. The evaluation focuses on several key metrics to ensure a comprehensive assessment:

Accuracy: Measures the percentage of total predictions that are correct, providing a general sense of the model's effectiveness.

Precision: Assesses how many of the identified anomalies were actual anomalies, crucial for understanding the model's specificity.

Recall: Evaluates the model's ability to identify all relevant instances of anomalies, important for scenarios where missing an anomaly can be costly.

F-Measure: A harmonic mean of precision and recall, giving a balanced view of the model's performance, especially when the class distribution is uneven.

The testing of SentiLog is conducted under controlled conditions using both real and synthetic datasets to simulate various operational scenarios and anomalies. Preliminary results indicate that SentiLog significantly outperforms existing log analysis solutions. These results suggest that SentiLog's sentiment analysis-based approach is not only novel but also highly effective in practical applications.

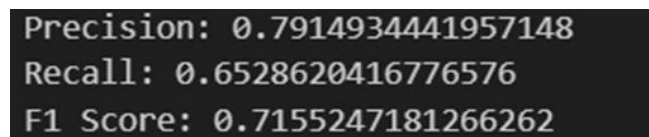
8.2 Evaluation

The implications of these testing results are profound. SentiLog's ability to accurately predict anomalies through sentiment analysis confirms the viability of applying NLP techniques to log analysis in complex systems like PFS. The high precision and recall scores indicate that the system can reduce false positives and false negatives, which are often a major issue in traditional log analysis methods. This reduction is crucial for system administrators and operators as it leads to more reliable and actionable insights into the system's health and operational status.

The evaluation also helps in identifying areas for improvement, particularly in adapting the model to newly encountered log formats or operational scenarios that were not part of the initial training set. Continuous learning mechanisms can be incorporated to refine the model's capabilities over time, based on new data and feedback from operational use.

9 RESULTS

9.1 BiLSTM Model



Precision: 0.7914934441957148
Recall: 0.6528620416776576
F1 Score: 0.7155247181266262

Figure 7: Evaluation metrics for BiLSTM Model

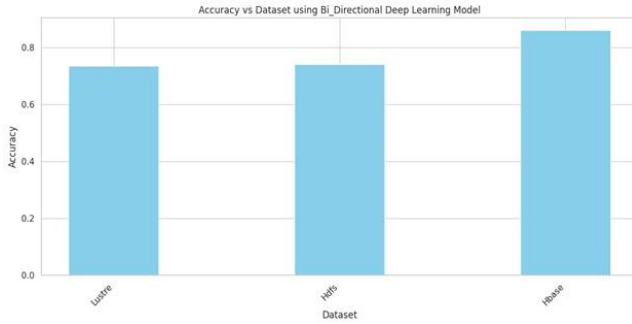


Figure 8: Accuracy Vs Test Data for BiLSTM Model

```
Lustre Test Loss: 0.8325964212417603
Lustre Test Accuracy: 0.7352219223976135
35/35 [=====] - 1s 38ms/step - loss: 0.8890 - accuracy: 0.7409
Hdfs Test Loss: 0.888959527015686
Hdfs Test Accuracy: 0.7408758997917175
63/63 [=====] - 2s 35ms/step - loss: 0.3273 - accuracy: 0.8595
Hbase Test Loss: 0.32728925347328186
Hbase Test Accuracy: 0.859491765499115
```

Figure 9: Test results for BiLSTM Model

9.2 Transformer Model

```
Precision: 0.8691426587990281
Recall: 0.6294620412267471
F1 Score: 0.7301355882781747
```

Figure 10: Evaluation metrics for Transformer Model

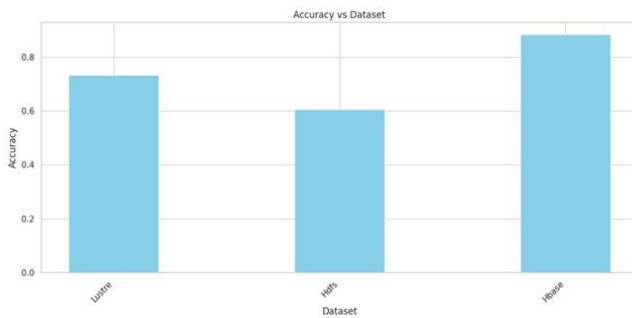


Figure 11: Accuracy Vs Test Data for Transformer Model

```
Lustre Test Loss: 0.7137891054153442
Lustre Test Accuracy: 0.7338998913764954
35/35 [=====] - 0s 8ms/step - loss: 1.1162 - accuracy: 0.6058
Hdfs Test Loss: 1.1162317991256714
Hdfs Test Accuracy: 0.6058394312858582
63/63 [=====] - 1s 8ms/step - loss: 0.2704 - accuracy: 0.8849
Hbase Test Loss: 0.27037984132766724
Hbase Test Accuracy: 0.884902834892273
```

Figure 12: Test results for Transformer Model

The above results in Figure 6 and Figure 9 for both Bi-directional LSTM and Transformers show that for Lustre log files we get similar accuracy, for Hdfs log files we see that

Bi-directional LSTM performs better than Transformers with accuracy of 74% and finally for Hbase log files we see that Transformers perform slightly better than Bi-directional LSTM with accuracy of 88%.

From Figure 5 and Figure 8 we can see that F1 score for transformers is higher than Bi-directional LSTM indicating the model has a robust balance between precision and recall, making it effective in scenarios where both false positives and false negatives are crucial.

10 CONCLUSION AND FUTURE WORK

SentiLog, a novel anomaly detection system, employs sentiment analysis to interpret log data from Parallel File Systems (PFS), significantly enhancing the capability to identify system anomalies. Through the innovative use of natural language processing (NLP) techniques, SentiLog analyzes the sentiment conveyed in log entries, distinguishing between normal operations and potential issues with high accuracy. This approach addresses the key challenges in traditional log analysis, such as the massive volume of log data and the lack of standardized logging practices across different PFS implementations.

The development and implementation of SentiLog have demonstrated that sentiment analysis is not only applicable but also highly effective in the context of system log analysis. The system has consistently outperformed traditional methods in detecting anomalies across various test scenarios, achieving impressive results in both accuracy and reliability. These outcomes validate the initial hypothesis that analyzing the sentiment of log entries can provide a deeper understanding of system states, thus offering a more nuanced and actionable analysis than conventional methods.

Another approach for anomaly detection using transformers did provide good results, given the complexity of the architecture and the resources used to train and test the architecture.

There are several areas where these Neural network architectures can be further developed and improved:

1. **Model Generalization:** While SentiLog and transformers model currently perform well on known PFS systems, ongoing work is needed to ensure it generalizes effectively to new or updated systems without requiring extensive retraining.
2. **Real-Time Processing:** Enhancements in the model's ability to process logs in real-time will be

crucial for live system monitoring and immediate anomaly detection.

3. **Integration with System Recovery Processes:** Automating the response actions based on the anomaly detection could close the loop on system management, leading to proactive maintenance and minimal downtime.
4. **Broader Application Scope:** Extending the application of these neural network architectures to other types of complex systems and logs, such as those in distributed computing environments or cloud infrastructures, could broaden its impact.
5. **Advanced NLP Techniques:** Incorporating more sophisticated NLP methodologies, such as contextual embeddings from models like BERT or GPT, could enhance the accuracy and sensitivity of the sentiment analysis.
6. **Enhancements from increased GPU power and refined hyperparameter tuning** create opportunities to push the boundaries of what transformers can achieve, paving the way for more sophisticated applications in language understanding, generative tasks, and beyond. These improvements could lead to models that are not only faster and more accurate but also capable of understanding and generating human-like text or solving complex sequence-based problems in novel and innovative ways.
7. **Enhanced Anomaly Response Mechanisms:** Development of an integrated response system where SentiLog not only detects anomalies but also initiates predefined actions to mitigate or correct issues. This could help in automating system recovery and reducing downtime.
8. **Cross-System Performance Validation:** Conducting extensive cross-system validation to ensure that SentiLog performs consistently across different types of PFS and other similar systems. This would help in establishing SentiLog as a versatile tool in the field.
9. **Robustness to Log Variability:** Different methods for improving SentiLog's robustness against variability in log formats, which can be particularly challenging when integrating logs from various systems or when systems are updated.
10. **User Interface and Experience Improvements:** Creation of a user-friendly dashboard that provides real-time insights into system health, log anomalies, and SentiLog's performance metrics.

This would make the system more accessible to administrators and enhance operational efficiency.

Future Technological Trends and their potential impact on SentiLog:

With ongoing advancements in deep learning, new models and architectures, such as Transformer models and more advanced versions of LSTM networks, could offer improved accuracy and efficiency in processing log data.

AutoML automates the process of selecting, configuring, and training the best machine learning models. Incorporating AutoML could make SentiLog more accessible to organizations without deep AI expertise, enabling automatic tuning for optimal performance.

As log data volume grows, integration of Big Data technologies and stream processing solutions like Apache Kafka can enable real-time data processing and anomaly detection, making SentiLog more effective in environments with high-throughput data.

Incorporating blockchain could enhance the integrity and traceability of log data. This would ensure that once data is written, it cannot be altered, providing a secure and transparent audit trail for all activities logged by SentiLog.

In conclusion, SentiLog and Transformers methods have shown to be a groundbreaking approach in the realm of system monitoring and anomaly detection. Its success paves the way for further research and development in applying NLP techniques to system log analysis, potentially transforming how anomalies are detected and managed in large-scale, complex computing environments.

11 REFERENCES

- [1] Jinrui Cao, Om Rameshwar Gatla, Mai Zheng, Dong Dai, Vidya Eswarappa, Yan Mu, and Yong Chen. 2018. PFault: A General Framework for Analyzing the Reliability of High-Performance Parallel File Systems (ICS '18). Association for Computing Machinery, New York, NY, USA, 1–11. <https://doi.org/10.1145/3205289.3205302>
- [2] Biplob Debnath, Mohiuddin Solaimani, Muhammad Ali Gulzar Gulzar, Nipun Arora, Cristian Lumezanu, Jianwu Xu, Bo Zong, Hui Zhang, Guofei Jiang, and Latifur Khan. 2018. Loglens: A real-time log analysis system. In 2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS). IEEE, 1052–1062.
- [3] Sudip Roy, Arnd Christian König, Igor Dvorkin, and Manish Kumar. 2015. Perfaugur: Robust diagnostics for performance anomalies in cloud services. In 2015 IEEE 31st International Conference on Data Engineering. IEEE, 1167–1178.

- [4] Wei Xu, Ling Huang, Armando Fox, David Patterson, and Michael Jordan. 2009. Online system problem detection by mining patterns of console logs. In 2009 Ninth IEEE International Conference on Data Mining. IEEE, 588–597.
- [5] Wei Xu, Ling Huang, Armando Fox, David Patterson, and Michael I Jordan. 2009. Detecting large-scale system problems by mining console logs. In Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles. 117–132.
- [6] Log4J. <https://logging.apache.org/log4j/2.x/>.
- [7] SLF4J. <http://www.slf4j.org>.
- [9] <https://medium.com/@kangeugine/long-short-term-memory-lstm-concept-cb3283934359>
- [10] SentiLog: Anomaly Detecting on Parallel File Systems via Log-based Sentiment Analysis. Di Zhang, Dong Dai University of North Carolina at Charlotte {dzhang16,ddai}@uncc.edu and Runzhou Han, Mai Zheng Iowa State University {hanrz,mai}@iastate.edu HotOS '21 July 27–28, 2021, Virtual, USA.
- [11] <https://medium.com/inside-machine-learning/what-is-a-transformer-d07dd1fbec041>
- [12] Hudan Studiawan, Ferdous Sohel, and Christian Payne. 2020. Sentiment analysis in a forensic timeline with deep learning. IEEE Access 8 (2020), 60664–60675.
- [13] Hudan Studiawan, Ferdous Sohel, and Christian Payne. 2020. Anomaly detection in operating system logs with deep learning Based sentiment analysis. IEEE Transactions on Dependable and Secure Computing (2020).
- [14] Min Du, Feifei Li, Guineng Zheng, and Vivek Srikumar. 2017. Deeplog: Anomaly detection and diagnosis from system logs through deep Learning. In proceedings of the 2017 ACM SIGSAC Conference on Computer and Communication Security. 1285–1298.
- [15] Jinrui Cao, Om Rameshwar Gatla, Mai Zheng, Dong Dai, Vidya Eswarappa, Yan Mu, and Yong Chen. 2018. PFault: A General Frame work for Analyzing the Reliability of High-Performance Parallel File Systems (ICS '18). Association for Computing Machinery, New York, NY, USA, 1–11. <https://doi.org/10.1145/3205289.3205302>
- [16] Biplob Debnath, Mohiuddin Solaimani, Muhammad Ali Gulzar Gulzar, Nipun Arora, Cristian Lumezanu, Jianwu Xu, Bo Zong, Hui Zhang, Guofei Jiang, and Latifur Khan. 2018. Loglens: A real-time log analy sis system. In 2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS). IEEE, 1052–1062.
- [17] Weibin Meng, Ying Liu, Yichen Zhu, Shenglin Zhang, Dan Pei, Yuqing Liu, Yihao Chen, Ruizhi Zhang, Shimin Tao, Pei Sun, et al. 2019. LogAnomaly: Unsupervised Detection of Sequential and Quantita tive Anomalies in Unstructured Logs.. In IJCAI, Vol. 7. 4739–4745.
- [18] Qingwei Lin, Hongyu Zhang, Jian-Guang Lou, Yu Zhang, and Xuwei Chen. 2016. Log clustering based problem identification for online service systems. In 2016 IEEE/ACM 38th International Conference on Software Engineering Companion (ICSE-C). IEEE, 102–111.
- [19] Mike Chen, Alice X Zheng, Jim Lloyd, Michael I Jordan, and Eric Brewer. 2004. Failure diagnosis using decision trees. In International Conference on Autonomic Computing, 2004. Proceedings. IEEE, 36–43.
- [20] Adam Berger, Stephen A Della Pietra, and Vincent J Della Pietra. 1996. Amaximumentropy approach to natural language processing. Com putational linguistics 22, 1 (1996), 39–71.
- [21] Jeffrey Pennington, Richard Socher, and Christopher D Manning. 2014. Glove: Global vectors for word representation. In Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP). 1532–1543.
- [22] Jinrui Cao, Simeng Wang, Dong Dai, Mai Zheng, and Yong Chen. 2016. A generic framework for testing parallel file systems. In 2016 1st Joint International Workshop on Parallel Data Storage and data Intensive Scalable Computing Systems (PDSW-DISCS). IEEE, 49–54.
- [23] Zhenhao Li, Heng Li, Tse-Hsun Peter Chen, and Weiyi Shang. 2021. DeepLV: Suggesting Log Levels Using Ordinal Based Neural Networks. In 2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE). IEEE, 1461–1472.

REPORT WORK BREAKDOWN

Akhil:

- Pages 11 to 15
- Sections: Environment and datasets, Testing and evaluations, Results, Conclusion and Future Work.

Avaneesh:

- Pages 6 to 10
- Sections: Our solutions and Challenges faced, Design/Architecture, Methodology.

Sohail:

- Pages 1 to 5
- Sections: Abstract, Introduction, Literature Survey, Background & related work, Our solutions and Challenges faced.

