The American Journal of Applied Sciences
ISSN 2689-0992 Volume 08 - 2026

Proceedings of Women in Academia, Research and Management for Work-life Initiatives for Sustainable Health & Empowering Safety (WARM-WISHES 2026)

Automated Log Intelligence System using Machine Learning Techniques

Ananya Srivastava

Department of Computer Science & Engineering

Shikha Singh

Amity School of Engineering & Technology

Vineet Singh

Amity University, Uttar Pradesh, Lucknow, India

Received: 21 Apr 2026 | Received Revised Version: 23 Apr 2026 | Accepted: 08 May 2026 | Published: 16 May 2026

DOI: 10.37547/tajas/warm-09

Abstract

Every application, server, and network device generate logs whenever it performs an action, making these logs a valuable source of information for determining whether a system is in a normal state or has any fault that can cause failure. In large distributed systems, the volume of generated logs is extremely high, often reaching millions of log entries on a daily basis, which makes the manual analysis of these logs almost impossible. Traditional rule-based log management methods require engineers to anticipate all possible faults and their corresponding alerts, making them ineffective for detecting new types of anomalies according to the changing times. This paper presents a machine learning based log intelligence system that processes raw, unstructured log data and performs anomaly detection through a complete pipeline of log parsing, preprocessing, and finally model evaluation. The dataset used is collected from the LogHub repository and is called the HDFS dataset. The HDFS dataset contains about 11 million logs. In this project we have used 3 approaches for log analysis: Isolation Forest (unsupervised algorithm), Random Forest (supervised algorithm), and LSTM (deep learning). The model is trained on each of the methods listed above and then is evaluated using the metrics of precision, recall, accuracy and F1 score. The result after evaluation shows that random forest achieves the highest F1 score of 0.97, then comes LSTM with the score of 0.89 and Isolation Forest a score of 0.79. The results achieved clearly demonstrate that machine learning and deep learning can be used to detect log anomalies in real world data for IT operations.

Keywords: Log analysis, machine learning, HDFS Dataset, Isolation Forest, Random Forest, LSTM

Cite This Article: Ananya Srivastava, Shikha Singh, & Vineet Singh. (2026). Automated Log Intelligence System using Machine Learning Techniques. The American Journal of Applied Sciences, 84–93. https://doi.org/10.37547/tajas/warm-09

1.0 Introduction

Softwares have become an essential part of today's world. Softwares are used in every activity be it posting a simple social media post or performing heavy bank transactions. If any of these servers fail it can cause a minor inconvenience or a significant loss. The data of critical sectors like banking and healthcare needs to be secure, available, accessible, and updated at all times. Making sure these systems are reliable is a crucial job of engineers and operators, requiring constant monitoring of system health and immediate response to any alerts before any user is affected.

Logs are timestamped records of events or actions that occur inside a computer system, application, network, or device. They are automatically generated the instant an action is performed and consist of six core components: Timestamp, Severity Level, Source/Component, Context, Message, and Metadata. Table 1 summarizes the anatomy of a log line. Since the logs capture all the activities performed on the system so can be used to detect faults and failures before happening. He et al. (2016) explained how logs can be used for anomaly detection.

TABLE 1. Anatomy of a Log Line

Component	Meaning
Timestamp	Exact time at which the event occurred.
Severity Level	How serious is the event?
Source/Component	From which part of the system is this log generated.
Context	Additional info about which parts of the system was involved in the event.
Message	The actual human readable part that tells what happened.
Metadata	Extra supporting details about the event.

Since each component of a system generates logs continuously, millions of log entries are produced each day, and this process continues 24×7. This makes the manual inspection of these logs nearly impossible. Even automated systems that collect all logs in a central place still face the challenge that for every million logs generated, only a few represent anomalies while most are normal system logs. Current rule-based approaches are not reliable, as they require engineers to anticipate all possible types of problems before, and they fail for detecting any new or rapidly changing faults and threats. Zhang et al. (2019) proposed more robust log anomaly detection systems capable of adapting to evolving systems.

Machine learning provides a better solution to this problem of anomaly detection. Instead of manually inspecting millions of logs, a machine learning or deep learning model can be trained on historical log data to learn what normal behaviour of the system looks like and to detect deviations from that normal pattern that can represent an anomaly. This type of system will be able to successfully identify anomalies in logs. It can even detect the new ones without any manual effort. But to build such a machine learning model comes with its own challenges. Firstly, the logs data from the real world is highly unstructured. Before any preprocessing or training is applied to it, it needs to be converted in a structured format. This is achieved by a process called as log parsing. Zhu et al. (2019) demonstrated that poor log parsing can reduce the accuracy of a model by significantly, clearly highlighting the importance of log parsing in log analysis systems.

The next challenge that we encounter is that the anomalies in real world logs are very rare, which already makes the dataset biased towards normal logs. In a stable computing system, anomalous events may account for only 2 to 3 percent of the total data, creating a strong class imbalance problem. A model that predicts every log

as normal could achieve around 97 percent accuracy while completely failing to detect any real issues. The third challenge is that no single machine learning approach works best in all situations. Unsupervised methods are useful because they do not require labelled data, while supervised methods achieve higher accuracy when labelled data is available, and deep learning approaches like LSTM can capture the changes in data across sequences of events.

This paper presents a complete end-to-end machine learning based log intelligence system, starting from raw unstructured HDFS logs and ending with an interactive dashboard, implementing and comparing three approaches: Isolation Forest, Random Forest, and LSTM.

2.0 Literature Survey

Research in automated log analysis has expanded over the past two decades as distributed computing systems have become more complex and harder to monitor manually. This work falls into three main areas: log parsing, log-based anomaly detection, and the use of machine learning in this field.

2.1 Early Work on Log Analysis

One of the earliest important works was by Xu et al. (2009), who demonstrated that large-scale system issues could be detected using statistical methods applied to Hadoop logs through principal component analysis. Before applying machine learning, logs must first be converted into structured data. Lou et al. (2010) proposed mining invariants from logs, observing that under normal conditions system components follow consistent relationships and that failures cause these relationships to break. This introduced the idea that learning normal patterns and detecting deviations could identify anomalies without manual rules.

2.2 Log Parsing Methods

Before applying machine learning, logs must first be converted into structured data. Vaarandi (2003) introduced an early clustering-based approach to group similar log messages and extract patterns. Makanju et al. (2009) proposed IPLOM, which groups logs based on length and token position to form templates. He et al. (2017) introduced the Drain algorithm, which uses a fixed-depth tree to group logs efficiently in an online manner, processing logs one by one with the speed and scalability required for real systems. In a research paper by Zhu et al. in 2019 the authors compared 13 parsing methods and found that Drain algorithm performs well emphasizing on importance of log parsing step. A similar study was conducted by Le and Zhang (2022), where they found that the quality of parsing plays an important role in the context of log analysis.

2.3 Anomaly Detection Methods

Lin et al. (2016) introduced LogCluster, a method that groups log sequences over a fixed time interval and treats the sequences occurring least frequently as anomalous. Du et al. (2017) proposed DeepLog, one of the most well-known approaches in log analysis, which uses LSTM networks to learn normal log sequences and classifies them as anomalous when they deviate from the expected pattern. LSTM is used here because of this idea. Meng et al. (2019) introduced LogRobust, which used semantic vectors to handle different log formats present in the dataset. Guo et al. (2021) proposed LogBERT, which used the transformer architecture for log analysis. Recent studies have continued exploring transformer and attention-based approaches for log anomaly detection because of their ability to capture long-range dependencies in log sequences (Liu et al., 2024).

2.4 Machine Learning Algorithms

Breiman (2001) introduced the Random Forest algorithm, which is used in this project as a supervised machine learning model. In 2008, Liu et al. proposed Isolation Forest, an unsupervised method that detects anomalies by isolating rare data points quickly and repeatedly dividing the data until each point is separated from the others. One of the biggest advantages of this approach is that it works without requiring labelled data. In 1997, Hochreiter and Schmidhuber developed LSTM (Long Short-Term Memory) networks, which were capable of learning changes in data over time.

2.5 Summary

Research over time has clearly shown a shift from statistical methods to the use of advanced machine learning and deep learning approaches in the context of log analysis. It has been proven that log parsing is a crucial step, presence of labelled data improves performance, and deep learning models are effective but computationally very expensive. This project follows these insights by combining strong log parsing with three models: unsupervised (Isolation Forest), supervised (Random Forest), and deep learning (LSTM), all evaluated on the HDFS dataset.

3.0 Methodology

3.1 Dataset

The dataset used in this project is the Hadoop Distributed File System (HDFS) log dataset collected from the LogHub repository on GitHub (LogHub Dataset Repository, 2023). No synthetic data is used because it can cause false positives or cause the model to learn false patterns. Hadoop is a framework built by Apache for storing and processing data that is too large to be stored on one computer. HDFS is the storage component of Hadoop that splits files into multiple blocks stored across different servers worldwide. One server called the name node acts as an index keeping track of all the other blocks, while data nodes store the actual data. The interaction between the name node and data nodes produces the rich set of logs known as HDFS logs. The official HDFS documentation clearly explains how large files are stored across multiple systems (Apache Hadoop Project, 2023).

For this dataset, 203 servers working together for 38 hours generated approximately 11 million log lines. All this data was then manually inspected for anomalies, and each log was labelled as either normal or anomaly. The anomalies in this dataset are rare: for every 100 log lines, only approximately 3 are labelled as anomaly, making the dataset already biased towards normal logs.

HDFS is chosen for this project because it is widely used in real industry by various multinational companies, which ensures that the data being used is relevant to real-world engineering. Many published research papers have also used the HDFS dataset, providing an opportunity to compare results with already published and accepted work. The approximately 11 million log entries provide a rich dataset that can be trained and tested properly.

3.2 Algorithms Used

Three machine learning approaches are used and compared in this project.

Isolation Forest, proposed by Liu et al. (2008), is an unsupervised machine learning algorithm based on the idea that anomalies are different from normal data and are therefore easier to isolate. It builds multiple random trees called isolation trees, where a random data point is taken and the entire data is divided into two groups through a random split. This process is repeated until each data point is isolated. The number of splits required to isolate a point is called the isolation depth. Points that are isolated in fewer splits are considered anomalous, while those requiring many splits are considered normal. By default, the algorithm builds 100 isolation trees. The key advantage of this algorithm is that it works without any labelled data.

The next algorithm used in this project is Random Forest, proposed by Breiman (2001). It is a supervised machine learning algorithm that requires labelled data. It builds multiple decision trees and combines their results through majority voting. To introduce randomness, each tree is trained on a different subset of data points. This is achieved through a method called bootstrap sampling. This makes the algorithm more reliable because different trees work on different data points and features, making them unlikely to make the same mistakes. It can handle high dimensional data, meaning all 34 features inside a block can be processed without requiring excessive configuration.

LSTM (Long Short-Term Memory), developed by Hochreiter and Schmidhuber (1997), is a type of recurrent neural network that solved the vanishing gradient problem of standard RNNs by introducing a specially designed cell state capable of carrying important information throughout a sequence. Normal log sequences always occur in a specific pattern. Since LSTM learns this pattern, it can easily spot anomalies as they will differ from the normal log sequence. The DeepLog paper by Du et al. (2017) demonstrated this approach by training LSTM only on normal log patterns and flagging any deviation as an anomaly.

3.3 System Architecture

The entire system consists of five stages. It is shown clearly by the figure 1. The first stage is ‘log’ ingestion, where raw unstructured HDFS logs are given as input into the system. The second stage is ‘log’ parsing, where unstructured logs are converted into structured templates using Drain's algorithm. The third stage is, feature engineering, where structured text log templates are converted into numerical feature vectors for use by machine learning algorithms. The fourth stage is model training using three different approaches: Isolation Forest, Random Forest, and LSTM. The fifth stage involves performance comparison of the three approaches on a held-out test dataset, followed by a visual dashboard showing the number of logs, performance comparison, and detected anomalies.

Each stage in this system is self-contained and independent. Any stage can be changed without affecting the other stages. For example, any algorithm other than Drain can be substituted for log parsing without requiring any changes to the feature engineering or model training stages. It is important to note that the visual dashboard operates on the already stored dataset and does not involve live streaming of log data. This is for research and evaluation purposes only. To be deployed in a real-time system the model would need to work continuously on a stream of data logs.

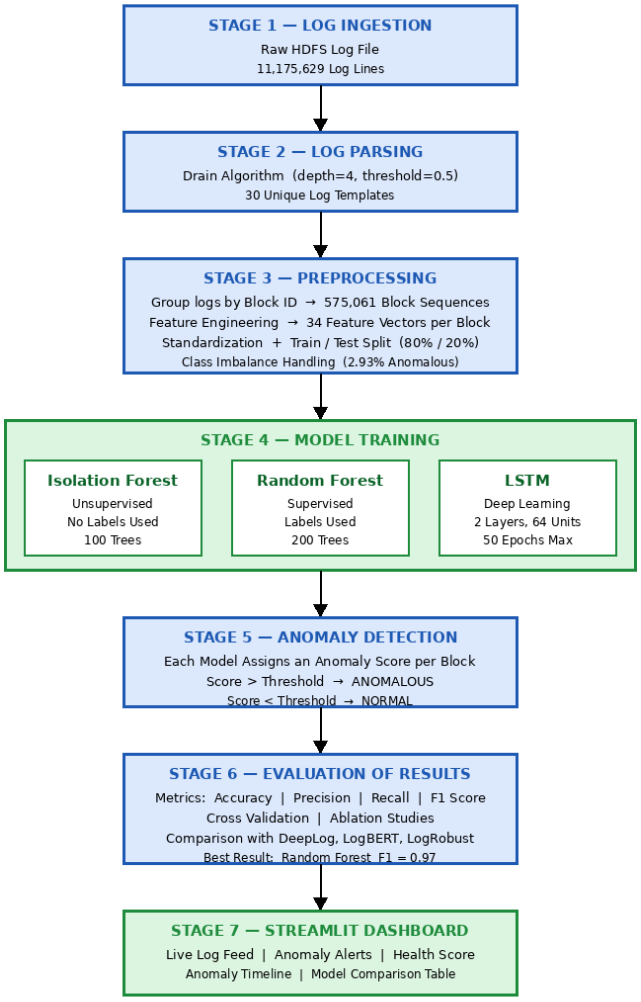

Figure 1: System Architecture — ML Based Log Intelligence System

Figure 1 representing the architecture of the system.

4.0 Implementation

The complete project is implemented in Python, which is rich in machine learning tools and highly efficient for handling massive amounts of data. Libraries used include Pandas for data loading, NumPy for all high-level calculations, scikit-learn for implementing Random Forest and Isolation Forest, TensorFlow/Keras for implementing LSTM, and Matplotlib for demonstration of graphs and charts. The code editor used is Visual Studio Code.

4.1 Step 1: Loading the Raw Log Data

The first step is to load the raw, unstructured log data file downloaded from the LogHub repository. The file is a plain text file containing one log per row, with each row having five parts: date, time, process ID, level of log, and the human-readable message. The meaning of all these parts of the logs is explained earlier. These parts are separated by spaces. After loading this unstructured text file into a Pandas data frame, the date and time columns are combined into a single timestamp column, and any

log line missing either the time or date column is removed to prevent false patterns from being learned by the model.

4.2 Step 2: Log Parsing using Drain Algorithm

Each log message is written by a developer during system configuration, with the format remaining the same while values like process ID and file names change. The Drain algorithm reads all log lines and converts them into clean templates by grouping lines first by the same number of words, then by the same first word, and finally comparing the entire sentence word by word. Identical words are kept the same; while differing words are replaced with wildcard characters, in this case the asterisk symbol. For example, the lines 'Received block blk_111 from datanode_1 size 67108864' and 'Received block blk_222 from datanode_5 size 67108864' both become one template: 'Received block * from * size *'. The Drain algorithm is configured with a depth of 4 and a similarity threshold of 0.5. Nagappan and Vouk (2010) proposed the idea of converting raw logs into structured events. For log parsing, Drain3 can also be used as it is a modern implementation of Drain's algorithm that automatically parses logs (Drain3 Library, 2023). After processing all 11 million log lines, 30 unique log templates are obtained, meaning all log lines are reduced to exactly 30 types of events.

4.3 Step 3: Grouping Logs by Block ID

After log parsing, the log lines are grouped by block ID. In HDFS, every part of the same file is stored with the same block ID. All block IDs are extracted using regular expressions, and all logs belonging to the same ID are grouped together. This provides the sequence of all events that took place on a particular file from its creation through its lifecycle. After this stage, the dataset reduces from 11 million log lines to 575,061 blocks. The anomaly labels are then loaded from a separate file provided with the HDFS dataset and matched to each block sequence. Blocks appearing in the anomaly list receive a label of 1 and all other blocks receive a label of 0. Out of 575,061 total blocks, 16,838 are labelled as anomalous, which is approximately 2.93 percent of all blocks.

4.4 Step 4: Feature Engineering

Since each block contains a different number of templates, a fixed-size numerical representation is required for all blocks. This is achieved in two parts. First, an event count vector is created by counting how many times each of the 30 templates appeared in a block's sequence, providing a 30-number vector for each block. Second, this event count vector is expanded by adding four additional numbers: the total count of all log events for that block; the duration in seconds from the block's first log event to its last log event; a binary flag of 1 if any ERROR or FATAL level log appeared and 0 otherwise; and the count of how many unique template types appeared for that block. Combining the 30 event count numbers and the 4 additional numbers gives a final feature vector of 34 numbers for every block. The entire dataset is now a table of 575,061 rows and 34 columns, which can be used as input to machine learning models. But before training the model all the features are standardized so that all features are treated equally, and the model does not give more importance to features with bigger values.

4.5 Step 5: Handling Class Imbalance

In HDFS dataset only 2.93 percent of blocks are anomalous, so we need to implement different strategies for each approach, to ensure the model does not get biased towards the normal logs. For Random Forest, the class weight parameter is set to 'balanced', which automatically instructs the model to pay more attention to the minority anomalous class during training. For Isolation Forest, the contamination parameter is set to 0.0293, telling the model that approximately 2.93 percent of the data is expected to be anomalous. For LSTM, the same class weight balancing approach used in Random Forest is applied during training.

4.6 Step 6: Training the Three Models

The dataset is split in a ratio of 80:20, where 80 percent of blocks go into the training set and 20 percent go into the test set. Stratified splitting is used to ensure both sets have approximately the same proportion of anomalous blocks at about 2.93 percent each. Isolation Forest is trained on the training set without using any labels and builds 100 isolation trees where each tree randomly partitions the feature space until every block is isolated. Random Forest is trained using both feature vectors and labels, building 200 decision trees where each tree is trained on a randomly selected subset of training data. LSTM receives block log sequences as input. Since different blocks may have different lengths, padding up to 100 is applied to ensure every sequence has exactly one hundred elements, with zeros added at the beginning of shorter sequences. The LSTM model consists of 2 LSTM layers with 64 neurons each and is trained for 50 epochs to prevent overfitting.

4.7 Step 7: Evaluation Metrics

After training all models, their performance is evaluated on the held-out test set using four metrics. Accuracy measures how many predictions were correct in total including both normal and anomalous ones, though it is noted that this metric is somewhat misleading for imbalanced datasets. Precision measures out of all blocks flagged as anomaly, how many were anomalous, with high precision indicating fewer false alarms. The metric recall tells us out of total number of anomalies present, how many the model was able to identify correctly. The next metric we have is the F1-Score which is the harmonic mean of precision and recall. A high recall values tells us the trained model only missed a few anomalies, and F1 score provides a balanced view of overall model performance.

4.8 Step 8: Visualization Dashboard

In the final step we build an interactive dashboard using Streamlit, which is a Python library that allows creating web-based interfaces using only Python code. The dashboard displays a colour-coded live log feed where INFO logs appear in green, WARNING in yellow, and ERROR in red. It also displays an anomaly alert panel listing all detected anomalous blocks along with their scores from each model, and finally a model comparison table displaying precision, recall, and F1-score for all three approaches. This dashboard gives an idea of what is happening in the system even to people who do not understand technical jargon.

5.0 Results and Discussion

5.1 Log Parsing Results

The Drain algorithm converted 11 million log lines into 30 templates. One thing to note here is that the 60 percent of the entire logs are classified into just the top 5 templates. This clearly demonstrates the imbalance present in the dataset and emphasizes on importance of not only finding out the presence of rare events but also learning any unusual patterns that occur. For example, a system that shows many 'request' events but very few or no corresponding 'success' events indicates a potential failure in the system.

5.2 Isolation Forest Results

During evaluation the Isolation Forest achieves an accuracy of 96.4%, precision of 0.71, recall of 0.89, F1 score of 0.79. The high recall indicates that the model successfully identifies most actual anomalous cases. But the presence of low precision shows that it also identifies many normal logs as anomalous. This approach is closer to real world because it does not require the labelled data which is the case in real world when logs are generated. But a high number of false positives increase the workload of engineers because it will require them to investigate logs that are not anomalous.

5.3 Random Forest Results

The Random Forest achieves an accuracy of 98.4%, with a precision of 0.96, recall of 0.98, F1 score of 0.97. It turns out to be the best performing of all three. This strong performance can be due to presence of labelled data, which allows the model to clearly identify the difference between normal and anomaly. This approach is accurate and reliable at the same time.

5.4 LSTM Results

The LSTM model achieves an accuracy of 98.2%, precision of 0.87, recall of 0.91, F1 score of 0.89. The performance lies in middle of Isolation Forest and Random Forest. The main advantage of using LSTM is that it learns the changes in data over time, so if a normal process follows a fixed pattern so an anomaly will break this pattern. This is captured with the help of LSTM. The major challenge of using LSTM is that it is computationally very expensive, requires a lot more time in training than other approaches. It requires heavy computational resources.

5.5 Comparative Analysis

TABLE 2. Performance Comparison of All Three Approaches

Model	Accuracy	Precision	Recall	F1 Score
Isolation Forest	96.4%	0.71	0.89	0.79
Random Forest	98.4%	0.96	0.98	0.97
LSTM	98.2%	0.87	0.91	0.89

The model that we choose depends on what the requirements and constraints are. It does not only depend on the performance numbers. A comparison of all the methods used in given in Table 2. If the labelled data is not available, we will have to choose the algorithm of Isolation Forest because it works even without labels. If we only require high accuracy we will choose Random Forest, in that case we must make sure that the labelled data is available for it to work. It is also faster to train and easier to deploy compared to deep learning models. The requirement for labelled data remains a challenge because labelling millions of logs requires significant time and effort. If anomalies depend on the order of events, then LSTM becomes useful because it captures sequence patterns that other models cannot, though this comes at the cost of higher training time and resource usage.

6.0 Conclusion and Future Work

This project successfully builds a machine learning based system to detect unusual patterns in logs of distributed computing systems. The model uses a complete pipeline from taking raw HDFS log data to preprocessing it, training the model and finally evaluating the results. The system demonstrates an end-to-end log analysis pipeline with a clear comparison between supervised, unsupervised, and deep learning approaches for log intelligence. It is clearly demonstrated that using machine learning for the process of log analysis outperforms the rule-based alert systems. The use of the HDFS dataset, which is a benchmark in the field of log analysis used by some of the best research papers in the industry, ensures that the model developed is closer to real-world scenarios.

The most important finding that this study highlights is the critical importance of log parsing. No advanced log analysis system can be built directly on raw log data because it differs in format and structure across systems and even different parts of the same system. Converting it into a fixed and accepted format is the foremost requirement. The Drain algorithm is used for this step in this study, though any other algorithm can be substituted.

Among the three approaches used Random Forest achieves the best overall performance with an F1 score of 0.97, making it most suitable to be deployed. LSTM achieves an F1 score of 0.89 and provides the additional capability of detecting sequence-based anomalies that event count models might miss. Isolation Forest achieves an F1 score of 0.79 but has the important advantage of not requiring any labelled training data, making it the right choice when labels are unavailable which is the case in real world where logs are generated continuously. Overall, using machine learning for log analysis is not only feasible but can achieve great performance and significantly reduce the manual labour of analysing logs thus helping to reduce failure rates and make systems more reliable.

While this study was successful in several aspects, clear areas exist for extending the research and improving performance along with usability.

The most important area for improvement is building a system capable of real-time streaming and analysis of logs. In this study, log data is fed into the model once for learning and analysis. Continuous feeding of newly generated logs into the system would require integration with advanced data streaming platforms and would help the model update its learning as new data arrives.

The next area where improvement can be made in future works will be explainability. It means finding out why a particular log has been classified as anomalous by the model. If this is achieved the real cause of system issues or what is happening in the system can be determined. It

will make the system transparent, hence more trustworthy.

The next improvement will be to use multiple datasets for evaluation of model performance. This will help in determining how well a model is able to generalize. If a model is performing well for multiple datasets, it has more potential to be deployed in industry. One thing to note here is that different datasets will have different formats, so bringing them in same standardized formats will be a crucial task. Also, to determine the most suitable approach for log analysis, more advanced models such as transformer-based models should be explored. Recent studies have also explored the use of large language models for log anomaly detection, showing promising results in understanding complex log semantics and unstable log patterns (Guan et al., 2024). We can also compare the performance of model against an already deployed log analysis system to determine the readiness for real industry.

Declaration

The authors hereby declare that the manuscript submitted for consideration is an original work and has not been published or submitted elsewhere for publication. The authors take full responsibility for the integrity, accuracy, and ethical compliance of the work presented in the manuscript, including all revisions made in response to reviewer comments.

AI Usage Statement

Authors declare that AI tools, if used, were solely employed to improve the clarity, grammar, and language of the manuscript. No data, results, or scientific content were generated or altered using AI.

Conflict of Interest and Ethical Compliance

i. Any potential conflicts of interest, whether financial or non-financial, have been fully disclosed. – Not Applicable
ii. All sources of funding and financial support received for the conduct of the study have been appropriately acknowledged, including any updates made during revision. – Not Applicable
iii. Necessary ethical approvals have been obtained from the relevant institutional or regulatory bodies for studies involving human participants, animals, or sensitive data, wherever applicable, and are clearly stated in the manuscript. – Not Applicable

References

1. Apache Hadoop Project. (2023). HDFS architecture guide. https://hadoop.apache.org/docs/stable/hadoop-project-dist/hadoop-hdfs/HdfsDesign.html
2. Breiman, L. (2001). Random forests. Machine Learning, 45(1), 5–32.
3. Drain3 Library. (2023). Online log template extraction using Drain. https://github.com/logpai/Drain3
4. Du, M., Li, F., Zheng, G., & Srikumar, V. (2017). DeepLog: Anomaly detection and diagnosis from system logs through deep learning. In Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security (pp. 1285–1298). ACM.
5. Guan, W., Cao, J., Qian, S., Gao, J., & Ouyang, C. (2024). LogLLM: Log-based anomaly detection using large language models. arXiv. https://arxiv.org/abs/2411.08561
6. Guo, H., Yuan, S., & Wu, X. (2021). LogBERT: Log anomaly detection via BERT. In Proceedings of the International Joint Conference on Neural Networks (IJCNN) (pp. 1–8). IEEE.
7. He, P., Zhu, J., Zheng, Z., & Lyu, M. R. (2017). Drain: An online log parsing approach with fixed-depth tree. In Proceedings of the IEEE International Conference on Web Services (ICWS) (pp. 33–40). IEEE.
8. He, S., Zhu, J., He, P., & Lyu, M. R. (2016). Experience report: System log analysis for anomaly detection. In Proceedings of the IEEE International Symposium on Software Reliability Engineering (ISSRE) (pp. 207–218). IEEE.
9. Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. Neural Computation, 9(8), 1735–1780.
10. Le, V. H., & Zhang, H. (2022). Log-based anomaly detection without log parsing. In Proceedings of the IEEE/ACM International Conference on Automated Software Engineering (ASE) (pp. 492–504). IEEE/ACM.
11. Lin, Q., Zhang, H., Lou, J., Zhang, Y., & Chen, X. (2016). Log clustering based problem identification for online service systems. In Proceedings of the International Conference on Software Engineering Companion (ICSE-C) (pp. 102–111). IEEE/ACM.
12. Liu, F. T., Ting, K. M., & Zhou, Z. H. (2008). Isolation forest. In Proceedings of the IEEE

International Conference on Data Mining (ICDM) (pp. 413–422). IEEE.

13. Liu, Y., Ren, S., Wang, X., & Zhou, M. (2024). Temporal logical attention network for log-based anomaly detection in distributed systems. Sensors, 24(24), 7949.
14. LogHub Dataset Repository. (2023). A large collection of system log datasets for AI-driven log analytics. https://github.com/logpai/loghub
15. Lou, J., Fu, Q., Yang, S., Xu, Y., & Li, J. (2010). Mining invariants from console logs for system problem detection. In Proceedings of the USENIX Annual Technical Conference (pp. 1–14). USENIX.
16. Makanju, A., Zincir-Heywood, A. N., & Milios, E. E. (2009). Clustering event logs using iterative partitioning. In Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (pp. 1255–1264). ACM.
17. Meng, W., Liu, Y., Zhu, Y., Zhang, S., Pei, D., Liu, Y., Chen, Y., Zhang, R., Tao, S., Sun, P., & Zhou, Y. (2019). LogRobust: Anomaly detection for unstable log data using prior knowledge. In Proceedings of the 2019 EuroSys Conference (pp. 1–13). ACM.
18. Nagappan, M., & Vouk, M. A. (2010). Abstracting log lines to log event types for mining software system logs. In Proceedings of the IEEE Working Conference on Mining Software Repositories (MSR) (pp. 114–117). IEEE.
19. Vaarandi, R. (2003). A data clustering algorithm for mining patterns from event logs. In Proceedings of the IEEE Workshop on IP Operations and Management (IPOM) (pp. 119–126). IEEE.
20. Xu, W., Huang, L., Fox, A., Patterson, D., & Jordan, M. I. (2009). Detecting large-scale system problems by mining console logs. In Proceedings of the ACM SIGOPS Symposium on Operating Systems Principles (SOSP) (pp. 117–132). ACM.
21. Zhang, X., et al. (2019). Robust log-based anomaly detection on unstable log data. In Proceedings of the ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE) (pp. 807–817). ACM.
22. Zhu, J., He, S., Liu, J., He, P., Xie, Q., Zheng, Z., & Lyu, M. R. (2019). Tools and benchmarks for automated log parsing. In Proceedings of the International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP) (pp. 121–130). IEEE/ACM.