SMU Data Science Review

Volume 8
Number 1 *Spring 2024*

Article 10

Intelligent Solutions for Retroactive Anomaly Detection and Resolution with Log File Systems

Derek G. Rogers
Southern Methodist University, dgrogers@mail.smu.edu

Chanvo Nguyen
Southern Methodist University, vnguyen@mail.smu.edu

Abhay Sharma
Southern Methodist University, abhays@mail.smu.edu

Intelligent Solutions for Retroactive Anomaly Detection and Resolution with Log File Systems

Abhay Sharma[1], Derek Rogers[1], Vo Nguyen[1], Paul Huggins[2], Dan Ma[2]
[1] Master of Science in Data Science, Southern Methodist University
(Email: Abhays@mail.smu.edu, Dgrogers@mail.smu.edu, Vnguyen@mail.smu.edu)
[2] Microsoft, NE 36th St,
Redmond, WA 98052 USA
paulhuggins29@gmail.com
danma@microsoft.com

Abstract. This paper explores the intricate challenges log files pose from data science and machine learning perspectives. Drawing inspiration from existing methods, LAnoBERT, PULL, LLMs, and the breadth of recent research, this paper aims to push the boundaries of machine learning for log file systems. Our study comprehensively examines the unique challenges presented in our problem setup, delineates the limitations of existing methods, and introduces innovative solutions. These contributions are organized to offer valuable insights, predictions, and actionable recommendations tailored for Microsoft's engineers working on log data analysis.

1 Introduction

Log files are indispensable system monitoring and management tools, providing essential data for real-time performance assessment and system health checks. They play a crucial role in security and compliance, detecting breaches, auditing activities, and ensuring regulatory adherence. Logs are essential for debugging, troubleshooting, and analyzing network performance in software development and network administration. Furthermore, data scientists and engineers increasingly utilize log files in machine learning for predictive analytics, while customer support teams employ them to enhance product and service quality.

Support engineers, software developers, and other technical services interpret these logs, which lead to actions such as enhancing security measures, troubleshooting, optimizing system operations, understanding user behaviors, ensuring regulatory compliance, and preventing future issues.

While invaluable, these record-keeping systems have limitations due to the inherent complexity and volume of log data. Traditional methods often struggle to identify and explain relevant information effectively, leading to challenges in efficiently extracting meaningful insights. This complexity heightens the challenge of detecting system failures, underscoring the urgent need for more effective automated solutions. Support engineers methodically analyze the varied data in logs, filtering and interpreting extensive information to construct viable solutions for system issues. The broad scope of logs results in a level of complexity that no singular support engineer can generalize to and becomes a complex task for large corporations that provide a plethora of applications to consumers. The role of artificial intelligence for IT operations (AIOps) aims to improve reliability and stability in IT services by detecting failures and fully utilizing logs as a resource for troubleshooting.

Recent advancements in natural language processing (NLP) have unlocked new opportunities for application design. Neural networks and the introduction of transformers, as highlighted by Vaswani et al. in their pivotal 2017 paper "Attention is

All You Need." prove transformers have outperformed traditional neural network models by enhancing text parsing efficiency and context comprehension from input text. Large Language Models (LLMs) build on transformers' capabilities, offering a context-rich analysis and enabling users to interact with models in a human-understandable text, receiving probabilistic responses from training data. This approach efficiently analyzes text-heavy data, such as log messages, as noted by Mahowald et al. in 2023. Developments of robust platforms, such as TensorFlow and HuggingFace, allow any interested party to use models such as OpenAI's GPT 3.5/4, Meta's LLaMA, and Google Gemini innovatively. The community has already started testing and ideating how to best leverage these models for log files and anomaly detection. (DeepLog: Anomaly Detection and Diagnosis from System Logs through Deep Learning, 2017) Despite these advancements, there is still disagreement on their capabilities and the future of generative AI (What's wrong with LLMs and what we should be building instead, 2023).

Significant development has been made in utilizing NLP methods to identify relevant information within log analysis. (Karlsen, 2023) Log files contain a slew of relevant numerical information regarding the state of the program or the machine and many human-readable messages in the form of system status updates. Potential NLP solutions include topics such as summarizing information found within log files, developing trends to visualize data, and the ability to isolate and highlight anomalous log files within large datasets. The generalization of an approach using unsupervised models is a uniquely helpful branch of NLP. A distinct challenge for applying NLP methods to these messages is that they can be treated as both natural and artificial language. They contain readable variable names, insightful sentences, and programmatic rules with regularity and consistency. This unique data hints at the need for NLP models to be explicitly tailored to log data.

The application of Transformers in methods such as *Bidirectional Encoder Representations from Transformers* (**BERT**) has revolutionized NLP-driven tasks and iterated and improved on the NN-driven models. The paper *Log Anomaly detection based on BERT* (**LAnoBERT**) sets a foundational understanding for implementing transformers to isolate and highlight anomalous logs while maintaining the unsupervised nature, allowing for novel applications without previous information/documentation.

Despite transformer advancements, their predictive anomaly detection methods have significant limitations due to the need for enormous amounts of accurately labeled data, which is time-consuming and costly. Traditional classification methods degrade performance when dealing with underrepresented classes in the data set, which is a detriment to detecting anomalous log events. Also, most transformer models are trained on datasets with typical words and formatting. Log files often use a wide variety and heavy use of delimiters and frequently use unstandardized abbreviations, making it difficult for models to interpret their meanings. Other limitations, such as quadratic scaling attention, are quickly being addressed, allowing for multimodal understanding across millions of tokens (Gemini 1.5: Unlocking multimodal understanding across millions of tokens of context, 2024) (Mamba: Linear-Time Sequence Modeling with Selective State Spaces, 2023).

The PULL method, proposed by Wittkopp T in 2023, introduces a reactive anomaly detection strategy that leverages attention-based models with a unique objective function for weakly supervised deep learning, focusing on predicted failure time windows. Similarly, isolation forests present an innovative and effective technique for anomaly detection, especially when integrated with failure time windows.

Many problems remain unsolved despite innovative solutions. Within academic research, a common approach leverages labeled datasets from repositories such as LogHub (Loghub: A Large Collection of System Log Datasets for AI-driven Log Analytics). These datasets typically categorize log events into distinct classes like messages, warnings, and errors based on pre-defined labels. While this classification method provides a structured way to identify potential anomalies, it often overlooks log data's nuanced and contextual nature, which is crucial for a comprehensive analysis. The reliance on such hard-coded labels can lead to a myopic view of anomaly detection, where the actual value lies not just in predicting these labels but in understanding the broader context in which these log events occur.

Anomalies in log data are frequently characterized by unusual patterns, such as a sudden surge in log messages or a clustering of specific types of events, which may not always correspond to error labels. For instance, a series of failed login attempts might be classified simply as messages. Nevertheless, their concurrent occurrence within a short timeframe could indicate a brute force attack, a critical security issue that warrants immediate attention. Similarly, an error indicating a server's inability to process requests becomes significantly more meaningful when observed with preceding logs showing an abnormal server usage spike. This context, which extends beyond the scope of predefined labels, is pivotal in accurately identifying and diagnosing anomalies. It underscores the necessity for a more sophisticated approach to log anomaly detection that can capture and analyze the intricate patterns and relationships within log data, thereby providing deeper insights and more actionable intelligence for support engineers and system administrators.

This paper focuses on creating valuable tools to effectively contextualize the unique problem setup of analysis on unknown log data. By limiting the amount of prior information from each log dataset and focusing on statistical inferences from each dataset, we can provide a generalized approach to each log set, allowing parts of the finalized product to be used with any log dataset.

2 Literature Review

This section focuses on setting a foundational understanding of modern methods used for log analysis and classification.

2.1 BERT Log Text Analysis

Understanding the nuanced integration of log data into BERT models provides valuable context and a universal lens on the fundamental differences and challenges log data provides within the context of models primarily trained and tested on text from web pages, books, and other sources most humans read and write. Logs can contain much numerical information and enough English to explain and provide context. These clips of English, in combination with column data, are usually enough for a support engineer to piece together strings of logs to create a narrative and understand overarching problems when performing a diagnosis. However, since logs can come in the tens of thousands per second, it becomes prohibitive to manually piece through and parse the descriptive English from each log to generate a picture. To effectively parse and understand English en-masse, it is advantageous to utilize deep learning text-interpretation methods, one of which is used heavily within the NLP industry, BERT. BERT or ‘Bi-directional Encoder Representations from Transformers’ is a novel processing tool that introduces the application of the Transformer in conjunction with

using bidirectional 'context' from large strings of sentences. When generating context from sentences passed in, the BERT model can tune faster and more effectively while performing comparable to, if not surpassing, traditional Transformer approaches (Devlin et al., 2019.). The BERT model utilizes the power of *transfer learning*, which allows for the initial pre-training of the model and then specifically on any generalized NLP task after the fact. By having the model extensively trained on sources such as *Wikipedia* or *BooksCorpus*, BERT can maintain an expansive 'context' for large strings of sentences. This pre-trained model can then be trained on more specific tasks; however, since the deepest layers of BERT are already pre-trained out of the box, the tuning time for specific executions is much faster than standard approaches.

BERT's approach is the foundation for deep learning bidirectional Transformers. However, there are more standard methods that are used within the industry. These methods include models such as LSTMs (Du et al., 2017), an application of RNNs, and CNNs (Lu et al., 2018) for parsing text. However, due to the rigid structure of these neural networks, they are exclusively dependent on a parser that needs to sit between the data and the model. While some applications develop their parser unique to their task, the standard parser that is used is called the *Drain* parser, which utilizes decision trees to parse lengths of strings and then force log data into a log template, designed to work with previously mentioned NN architectures (He et al., 2017). The decision to use NN or Transformer models depends on the task due to the sporadic and unstructured nature of logs around the industry.

2.2 Log Anomaly Detection

With the overarching goal of detecting anomalies, several models exist that perform anomaly detection on log files. These methods usually vary in two ways: first, the necessity of a log parser, and second, the supervised/unsupervised nature of the model. With the objective being the generalized approach to log anomaly detection, the less a parser is utilized, the broader the application of the log anomaly detection methods can be. One of the initial methodologies that represents this ideology is the PULL model. PULL (Reactive Log Anomaly Detection Based on Iterative PU Learning) utilizes a text classifier that is trained on Positive and Unlabeled Examples (PU Learning) (Wittkopp et al., 2023). This data framework focuses on applying attention-based models with a non-parser approach due to the unavailability of capturing and labeling structured data for these log files. The PULL model uses a combination of supervised and unsupervised analysis to provide a guided analysis that allows it to perform significantly better than most of its single-approach counterparts. The focus on the lack of a parser and novel application of attention-based text analysis models provides a robust foundation for any anomaly detection.

The second major paper that approaches the task of log anomaly without a parser is the LAnoBERT paper, which utilizes the previously mentioned BERT text encoder model in conjunction with statistical methods in isolating and classifying anomalous data within the dataset (Lee et al., 2023). The approach uses an MLM (Masked Language Model) approach, where a pre-trained BERT model is loaded and trained on non-anomalous data within the log files. The data is methodically masked to provide the BERT model with a unique context representative of the data it is trained on. Once trained, the model passes in both anomalous and non-anomalous data, where the resulting logs are classified with a probability to highlight which log files might be anomalous and which might not. This method provides a robust, unsupervised method that effectively removes any dependence on a log parser, keeping in line with the objective of a generalized log anomaly detection method.

2.3 ChatGPT for Log-Based Anomaly Detection

Log-based anomaly detection is a new and robust technique in monitoring network and system activities to determine suspicious behavior. While traditional methods have been commonly used in analyzing logs, such as PULL and LAnoBERT, the introduction of GenAI and LLMs demonstrates that new, robust techniques are on the horizon. One such framework, LogGPT, proposes the methodology to leverage LLMs for the enhancement of log-based anomaly detection (LogGPT: Exploring ChatGPT for Log-Based Anomaly Detection, 2023)This approach utilizes three frameworks:

- Log preprocessing: This technique transforms raw log messages into a structured format utilizing log parsing methods like Drain.
- Prompt Construction: This uses basic prompt engineering to create specific prompts in log anomaly detection tasks.
- Response Parser: Extracting outputs from ChatGPT to evaluate detected anomalies.

The paper focuses on five research questions regarding LogGPT performance with different prompts, window sizes, human knowledge injection, comparison to baseline methods, and interpretability. The datasets contain the NGL and Spirit, which were used for evaluation. These datasets have logs from a high-performing computing environment with 4.7 million messages. The baseline methods included deep learning methods that were previously established, such as LogAnomaly and LogRobust. There were some limitations in the sensitivity of the prompt variation and window sizes. There were also high false positive rates and hallucination problems provided by ChatGPT.

Significant findings from the paper include:

- Prompt Construction has a significant impact on anomaly detection performance.
- Window Size: Chunk sizes improve performance based on the configuration of how much data is used for contextual information.
- Human knowledge injection of specific domain knowledge helps enhance anomaly detection.
- LogGPT promises performance that often outperforms traditional methods.
- The interpretability of LogGPT provides helpful feedback and specific information about the anomalies and potential preventive suggestions.

The study concludes with LogGPT representing a step forward in using LLM for log-based anomaly detection, promising performance, and increased interpretability for future work. This study is also pivotal in exploring the applications of large language models in log-based anomaly detection. It opens avenues for integrating advanced languages to understand system monitoring and security analysis capability, potentially transforming how log data is analyzed in various industries.

2.4 Unsupervised Clustering Count

One of the manual processes performed in this paper was determining cluster sizes for grouping error log messages. Established methods of determining cluster size, such as the elbow method and average silhouette method, are a good starting point to automate this process. Because of the nuances and complexity of the log messages, more advanced methods, such as supervised graph embeddings, might be able to take

advantage of the additional context of the log messages to provide better clustering results. (Automatic selection of clustering algorithms using supervised graph embedding, 2020)

3 Methods

This study utilizes methods that provide varied and valuable perspectives on log data presented by Microsoft. We focus on identifying patterns and anomalies, extracting relevant information, and classifications within the log events. Methods and insights are stacked on one another, providing an architecture that offers analysis where the whole is more valuable than the sum of the parts.

3.1 Data Description with Existing Tools and Methodologies

The data provided by Microsoft consisted of two CSV files containing snippets of Azure ServiceBus event logs. Both log files were close to 10 minutes long and contained between 600,000 and 1,000,000 event logs, where each event in the file had over 20 columns of details, including log level, clock time, size, IDs, and event names. The *Message Compiler* (MC.exe) compiles instrumentation manifests and logs message content into message resource files to which an application can link. *SvcPerf*, visualized in Figure 1, is an application with a UI designed for filtering and searching through log files using explicit query selections.

The typical *SvcPerf* workflow involved adding the log event data and a manifest file, which was used to populate the message information about event logs. There is also a plot of log-level frequencies; however, so many log events occur over the 10 minutes that it is impossible to see any change in frequency for all but the most critical level of errors for how the plot is designed. Using the MC backend, *SvcPerf* created the message content utilized by many of our subsequent analysis steps. *SvcPerf* also provided a baseline for information such as start and end times, diagnostics, and the number of rows.

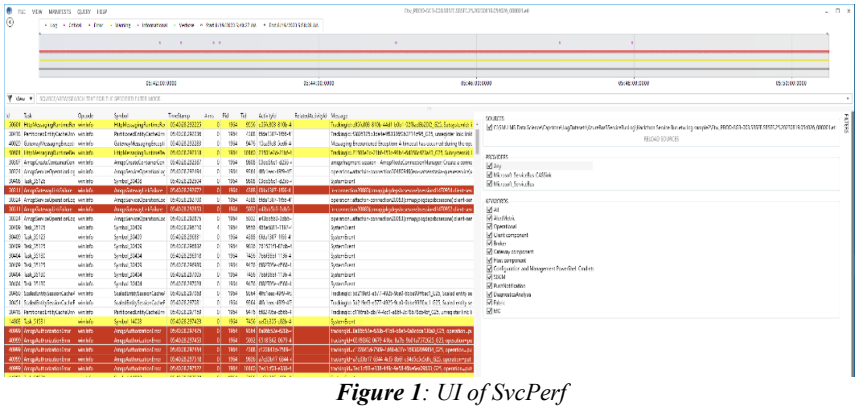


***Figure 1**: UI of SvcPerf*

3.2 Data Collection and Preprocessing

The initial phase of our research focused on data preprocessing of Microsoft Azure Service Bus log files, which are the basis of all our subsequent analyses. To prepare

this data for analysis, we started with the tabular data. We implemented preprocessing techniques to convert the data into a structured *.csv* format and correctly assigned the data type for each column. Utilizing *SvcPerf*, we extracted the message data and parsed it into a list of log event messages. From here, we had structured column data that could be matched with its corresponding message string.

3.3 Data Analysis

With the formatted data, we started collecting basic information about the log files to verify that our results matched SvcPerf's and established a foundational understanding of our data. Analyzing the log files' metadata provided vital insights into the scope of the data and some of the limitations we would have to consider, as represented in Table 1.

***Table 1**: Collected Meta Data*

File Size	350 MB
Start and End Time	8/19 7:01 – 8/19 7:11
Duration of Files	10 minutes and 8 seconds
Total Number of Log Events	968,737
Average Messages Per Second	1592 messages/second

This provided relevant information about the data size, which would restrict many of our data analysis techniques and motivate the anomaly detection techniques to occur after the data was distilled to a more focused subset, as running anomaly detection on close to a million log messages was too computationally complex.

We also used this phase to gauge and understand the efficacy of our columns. Many columns were filled with technical information irrelevant to our analysis and were omitted from modeling techniques. These included naming schemes, activity IDs, etc., where the data had patterns we could not interpret. Outside of the messages, one of the most relevant columns we utilized was the *Levels* column, with a set of pseudo-labels indicating some severity level within the data. Following the labeling schema of 4 through 1, where a log level 4 was considered normal, and 1 was the most abnormal, the dataset contained information on the logs following this schema.

Though the log file used for analysis contained approximately a million rows of information, our method of extracting data from *SvcPerf* required further preprocessing. Most notably, the contents of our *Message* column were desynced from the rest of the dataset due to the lack of a consistent pattern to indicate where one message ends and the following message begins. This was a quirk of using *SvcPerf*, which we overcame by implementing custom parsing logic and manual changes to the message data file.

Our second significant analysis focused on a more statistically driven time series analysis to create a focused subset of data. To visualize log frequency distributions within the dataset, we created bins of log events at consistent time intervals and found the count for each bin. Plotting this message frequency helped identify patterns in log events, which can efficiently highlight the location of relevant anomalies.

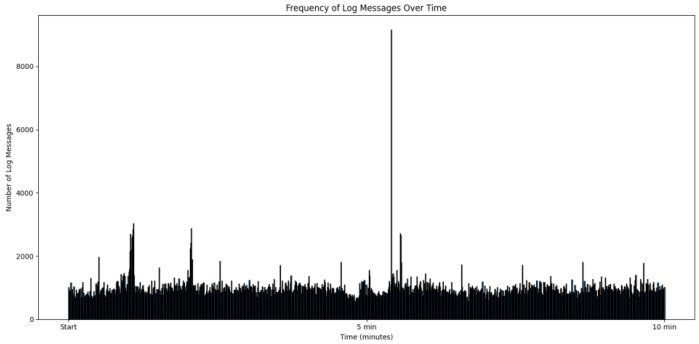


***Figure 2**: Plot of Message Frequency*

When plotting the message frequency data, visualized in Figure 2, we saw that there were distinct locations within the distribution of log data that would be of interest for our analysis. We used these spikes as general locations of possible anomalies to distill our data.

3.4 LanoBERT

With a partially distilled dataset, we utilized LanoBERT to perform a more robust anomaly detection sweep, reducing the number of log files even further. The execution of LanoBERT was split into two major components: first, training the requisite BERT Masked Language Model, then building an inference algorithm that allowed us to score the log files utilizing the trained BERT MLM model. We utilized the Level column to separate the nominal and abnormal files when organizing the data. We collapsed a training subset of log levels 3 and 4 into a dataset defined as nominal data and used it to train the MLM.

With a trained MLM, we recreated the inference algorithm pseudo-code, represented in Python to perform a scoring loop over any new dataset. For the inference step, we took a test subset of the files, ensuring that the dataset had at least some count of anomalous files (represented as log levels 1 and 2) and passed it through the scoring function of LanoBERT. Having generated an 'Anomalous Score' for each log file, we could define an arbitrary scoring threshold to pick out any files that scored above the threshold as anomalous to build our final distilled dataset.

3.5 Embedding Generation

Our next steps began with embedding generation using an embedding model. This technique translated our text data into high-dimensional vector space. These embeddings are dense vector representations, where each vector captures the semantic essence of the text using floating-point numbers. The proximity between any two vectors within this space is a semantic similarity; closer vectors represent closely related content.

To generate the embeddings, we utilized OpenAI's third-generation embeddings models, specifically designed to capture their robustness and ability to capture textual relationships of unstructured text. The embedding model we initially chose was text-*embedding-ada-002*, but with the newest release, *text-embedding-3-small* was considered.

3.6 Cluster Visualization & Analysis

Next, we applied clustering algorithms to the high-dimensional embedding representation. Our principal aim was to simplify analysis by providing information about groups of similar messages instead of analyzing messages individually. We experimented with multiple clustering techniques to achieve this, ultimately opting for the k-means clustering algorithm. This algorithm is celebrated for its efficiency and effectiveness in aggregating data based on feature similarity.

The k-means algorithm segments the dataset into a predefined number of clusters. Manual validation of the output results informed the selection of this number. The algorithm iteratively assigns each data point to the nearest cluster center, subsequently updating the center based on the aggregate of current members in each cluster. This iterative refinement persists until the cluster assignments stabilize, thus uncovering the inherent groupings within our dataset.

Given the high-dimensional nature of our vectors, dimensionality reduction techniques were essential for visualization. We chose Uniform Manifold Approximation and Projection (UMAP) over other methods, such as PCA and t-SNE, due to its superior performance in preserving the data's local and global structure. UMAP provided a robust and insightful two-dimensional representation of our clusters. Applying UMAP to our embedding data allowed us to project the clusters into a two-dimensional space, significantly enhancing our capacity to visually discern relationships and distinctions between the datasets.

Visualization plays a pivotal role in preliminarily identifying each cluster's characteristics. For instance, a distinctly separate cluster could indicate unique error log messages. This observation necessitates further examination of the data points within this unique cluster, applying prompt engineering techniques to uncover commonalities. Such analytical depth is crucial in understanding the nuances of our dataset, guiding our investigation toward meaningful insights, and potentially revealing novel patterns within the log messages.

The next step in our investigation was to label our identified clusters. This endeavor aims to understand the distinct groupings revealed comprehensively and allows a deeper exploration of cluster analysis.

3.7 Fine-Tuning LLM for Naming the Clusters

Upon successfully labeling the clusters, our next step was using prompt engineering to allow a large language model to summarize a sample of each cluster. This summarization allowed us to get any main ideas or similar patterns in determining the underlying patterns or characteristics of the grouping. To achieve this, we used GPT-4-Turbo, which utilized the following prompt to generate meaningful summaries:

3.7.1 Prompt Engineering Strategy

"Analyze Microsoft Azure Service Bus error log messages. This cloud service is designed to facilitate communication between applications and services. In analyzing Azure Service Bus error logs, common issues include permissions, connectivity, sender, receiver, processor, and transaction-related problems. These encompass handling Service Bus exceptions, managing access rights, addressing network connection challenges, troubleshooting message sending and receiving difficulties, resolving processor operation concerns, and navigating transaction complexities. Your

primary role is identifying log error patterns clustered around a suspected anomaly and summarizing what the clusters of log messages represent."

3.7.2 Selection of Text Samples

Each identified cluster had a random sample of up to 10 messages. This selection aims to capture a broad spectrum of the clusters' characteristics while remaining manageable in cost and time. After selecting text samples, we will apply the prompt to summarize and name them.

3.8 Random Forest for Anomaly Detection

The Isolation Forest algorithm represents an alternative strategy for anomaly detection. It is distinguished by its focus on isolating anomalies rather than modeling non-anomalous data points. It identifies significant deviations within the dataset, assuming the outlier is highly distinct.

Due to the difficulty of anomaly classification, an atypical data point was introduced into the dataset when testing the random forest by adding a quote from "The Lord of the Rings." This outlier aimed to test the detection mechanism, and the random forest correctly identified the quote as the anomaly.

3.9 Analytical Application of LAA (Log Analysis Assistant)

Retrieval Augmented Generation (RAG) LLMs retrieve relevant information from a curated knowledge base and use this as context when responding to user prompts. Using OpenAI's custom GPT, we create the Log Analysis Assistant, which serves as our RAG LLM with which the support engineer could interact. Information such as troubleshooting documentation, metadata, signal data, cluster summaries, and the anomaly from the isolation forest are all stored in the LAA's knowledge base.

3.10 Web User Interface Application Proof of Concept

Once the entire pipeline was developed, a Microsoft User Interface Streamlit application was conceptually created for an interactive and user-friendly interface. This application leveraged Streamlit, an open-source app framework, to create a lightweight web application that integrates with Microsoft Error Log uploads. The application was designed to allow users to upload their error logs and view the analysis results from the pipeline, including classification, anomaly detection, and machine learning models. The application's backend incorporated Python scripts that perform the necessary pipelines' executions, effectively recreating most of our methods through programmatic means.

4 Results

4.1 Log Analysis Results

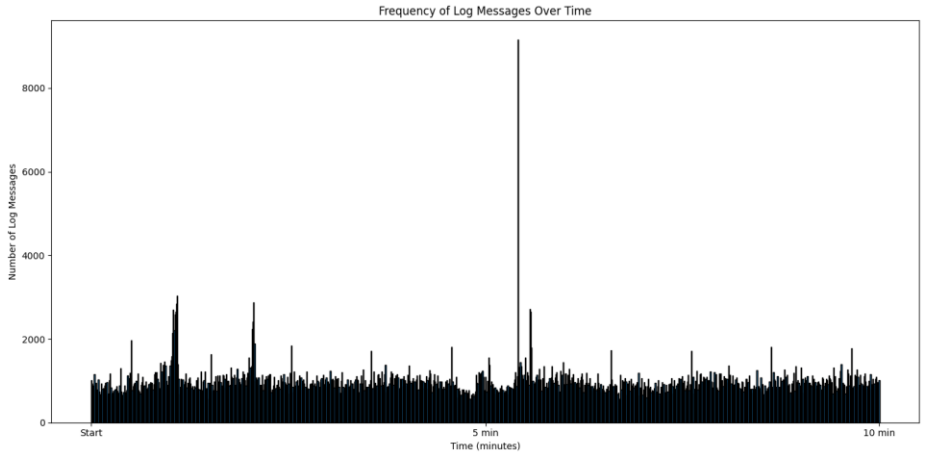


***Figure 3**: Plot of the frequency of log messages over 10 minutes.*

When plotting the message frequency data, visualized in Figure 3, we see a significant spike soon after the 5-minute mark. We decided to focus our following levels of analysis on this subset of the data. There are other intriguing patterns. For example, the minute before the significant spike, there is a wandering pattern in contrast to the noise of the rest of the plot. Smaller spikes occur about every minute. There are two medium-sized spikes near minutes 1 and 2. With the proper domain knowledge and analysis, this plot can tell a story to the support engineer and direct their analysis to focused subsets of relevant log events.

4.2 LAnoBERT Results

In executing the LanoBERT process, we first obtained a fine-tuned BERT-masked language model trained on log data selected at levels 3 and 4 to represent the nominal dataset. This fine-tuned model was conceptually apt in representing the context of nominal log files, which was verifiable by the decreasing loss function through the model training process.

With the model fine-tuned, we executed the inference algorithm by passing in a separate, isolated dataset that combined normal and abnormal messages. Using our scoring metric as a combination of loss and abnormal probability, we generated a score for each log within the provided dataset. This method allowed us to then focus on the 'Anomaly Score' and create arbitrary thresholds that could be used to cut through and pick out messages with the highest scoring. Of the 1,000 messages selected from the bin containing the most significant spike in message frequency, LnoBERT helped select the 77 most anomalous log events comprised of levels 1 and 2 and errors.

4.3 Embedded Results

The embedding function created from Section 3.1 yielded valuable outcomes that are instrumental in understanding the semantics in a vector space. Utilizing OpenAI's text embedding-3-small model, we translated the table of messages from error logs into a structured, high-dimensional embedding vector space, as seen in Table 2.

***Table 2**: The embedding generation results using the text-embedding-3-small model.*

Index	Message	Embedding
0	"Faulting messaging object due to an error. Na...	[-0.01512…, ...
1	"Aborting messaging object. Name = srikandi:qu...	[-0.02315…, ...
2	"Aborting messaging object. Name = srikandi:Qu...	[-0.02454…, ...
3	"Faulting messaging object due to an error. Na...	[-0.01683…, ...

These embedding vectors detail the error log messages and represent the information within the message content abstracted to some vector space. This allows the messages to be utilized in techniques that only function over highly structured data.

4.4 Cluster Analysis: Visualization in 2D

Using k-means clustering and applying UMAP for dimensionality reduction helped identify log message groupings. The data table in Figure 3 was also viewed in ascending order to determine correct grouping matching, which produced significant and compelling results.

MESSAGE	CLUSTER
Entity size become negative. Entity name = trac …	**0**
Entity size become negative. Entity name = trac …	**0**
Aborting messaging object. Name = mcapesea …	**1**
Aborting messaging object. Name = mcapesea …	**1**
Aborting messaging object. Name = mcapesea …	**1**
…	…
Protocol Client of type of ‘AmqpMessageConsuder …	**2**
Protocol Client of type of ‘AmqpMessageConsuder …	**2**
…	…
"Amqp publish failed. TrackingId:	**3**
"An exception was handled. TrackingId:	**3**
…	…
"The AmqpProtocolClient is being closed …	**8**
"The AmqpProtocolClient is being closed …	**8**
"The AmqpProtocolClient is being closed …	**8**

***Figure 3**: KMeans Clustering of Error Log Messages (K = 8) Table.*

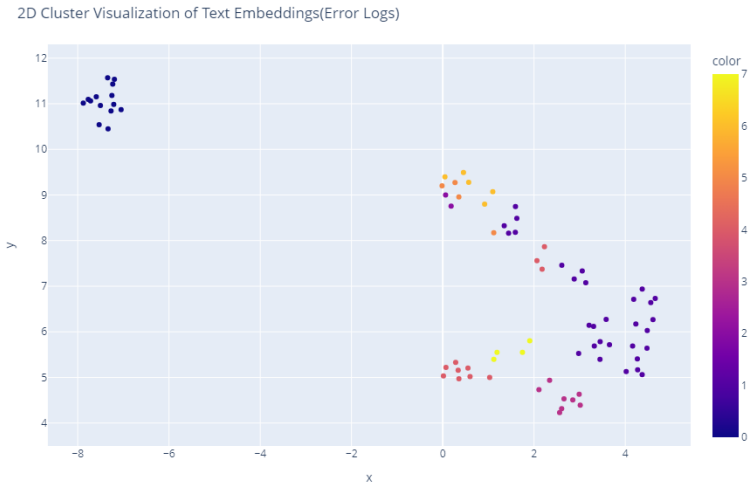


***Figure 4**: KMeans Clustering of Error Log Messages (K = 8) 2D Visualization.*

When plotting the results of *KMeans* in a reduced two-dimensional space, visualized in Figure 4, there was a grouping of clusters near $x = 0$ to $x = 5$, whereas near $x = -6$, there was a distinct grouping separated from all the rest. This preliminary observation guided further analysis into the content of this unique cluster and employed prompt engineering to extract and understand the specific nature of these errors.

***Figure 5**: A bar graph of the distribution of clusters within each cluster label.*

In addition to our clustering, we plotted a distribution of the log files within each cluster, visualized in Figure 5, to gain information on the types of log files clustered together. Through visual inspection, cluster label 1 has the highest count, followed by cluster labels 0 and 4. Although we can inspect the messages and confirm they present similar information contextually, further domain knowledge would be required to interpret why these log files are being grouped. Another noteworthy result is that the isolation forest result was grouped in cluster 2, which had the lowest count. This hints at an agreement between the *KMeans* and random forest results.

4.5 Fine-Tune GPT-4-Turbo using Prompt Engineering

After labeling the clusters from the visualization of the distinct cluster groups, prompt engineering with a large language model was employed to summarize a sample of up to 10 log messages for each cluster group. This process extracts the central ideas or patterns to determine the error logs' underlying characters. For this task, GPT-4-Turbo was selected as our deployment model, which helped generate meaningful summaries at an acceptable cost and speed.

The successful clustering of log files highlights the integration of OpenAI's Text Embedding ChatGPT 4 Turbo with sophisticated prompt engineering techniques to dissect and interpret log messages across various clusters. The output uncovered common patterns and aided us in identifying anomalies within the data. Despite the initial analysis yielding distinct outputs, the figure suggests a potential need to refine our prompt strategies or explore alternative feature extraction methodologies to enhance effective clustering. Figure 4 underscored the critical role of prompt engineering and feature extraction in leveraging AI for deep insights into system log analysis, setting the stage for further explorations in optimization techniques.

4.6 Anomaly Detection Results

The experiment to evaluate *KMeans'* effectiveness for anomaly detection yielded significant insights. We introduced a deliberate outlier, a line from "The Lord of the Rings," into the dataset, which served as an artificial anomaly.

***Table 4**: Results of the Cluster Anomaly Detection*

Message	Embedding	Cluster
Entity size become negative. En...	[-0.00871…, -0.01648…, …]	6
Entity size become negative. En...	[-0.00952…, -0.01545…, …]	6
Let this be the hour when we d ...	[-0.01556…, -0.03345…, …]	6

As seen in Table 4, the clustering did not isolate the movie line into its distinctive group; instead, it was grouped with other log event messages. This outcome could suggest a potential grouping of anomalies, indicating that clustering has created one distinct group of outlier categories. This finding may lead to further analysis or prompt engineering of cluster group 6 to identify it as the anomaly grouping. This highlights the limitations of *KMeans* for identifying anomalies and motivates our use of the isolation forest method for anomaly detection.

4.7 Isolation Forest Algorithm Findings

The Isolation Forest algorithm's application demonstrated its effectiveness as a viable strategy for detecting anomalies. This method successfully identified the introduced outlier – a quote from “The Lord of the Rings” as detailed in Table 5. This underscores the algorithm's capability to distinguish irregularities within the dataset. Moreover, Table 6 presents the Isolation Forest’s findings in the absence of the “Lord of the Rings” quote, where it selects a message from cluster 2, which recorded the lowest count, thereby corroborating the concordance between our *KMeans* clustering and Isolation Forest analyses in pinpointing the most anomalous messages.

***Table 5**: Results of the Isolation Forest Algorithm with Lord of the Rings Quote*

Message	Embedding	Cluster
Let this be the hour when we d ...	[-0.01556,-0.03345, ...]	6

***Table 6**: Results of the Isolation Forest Algorithm without Lord of the Rings Quote*

Message	Embedding	Cluster
Protocol client of type 'SoapMessageGroup ...	[-0.0210134294, ...]	2

Both methods demonstrated their respective strengths in anomaly and outlier detection. Cluster analysis with text embeddings revealed semantic analysis's potential in categorizing and summarizing atypical data points. The isolation forest algorithm, on the other hand, was notable for efficiently isolating and identifying anomalies based on their uniqueness. These observations offer valuable insights into anomaly detection methodologies in textual data, suggesting that combining both strategies could improve detection and analysis capabilities.

4.8 Streamlit User Interface Application

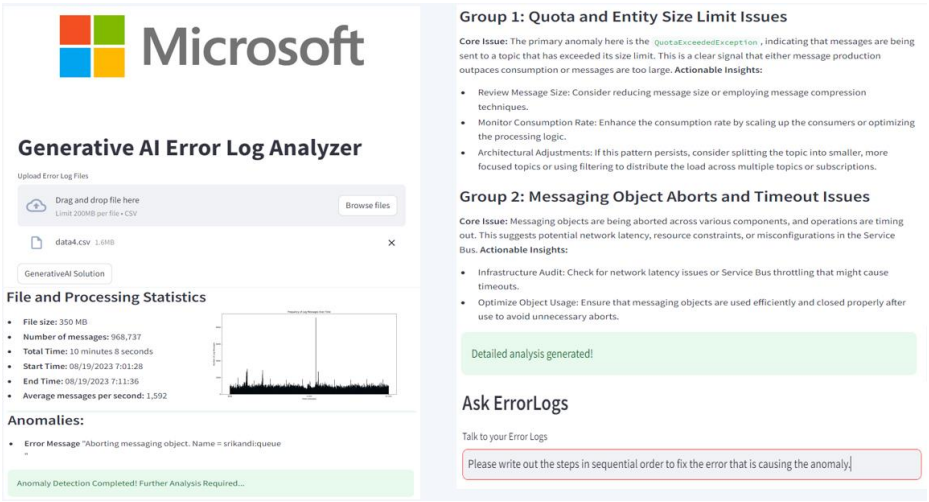


***Figure 6**: Sample Microsoft UI for Log Analysis*

As our final step and the culmination of our work, we developed a prototype UI for Microsoft using Streamlit. The UI, shown in Figure 6, allows support engineers to upload their log files. Inexpensive metadata analysis and plots are generated. This will allow the support engineers to create a more focused dataset where they believe relevant information can be found. After that, the error clustering analysis will appear, including a summarization and a question-and-answer system for engineers to chat with the data. The streamlined workflow and insights gained will allow the engineers to maintain the system more quickly and develop better error resolution strategies.

5 Discussion

5.1 Creating a Manageable Subset of Log Data

The methods and architecture proposed overcome the challenges of log analysis. Signal analysis of event log frequency or data size frequency provides unique visualizations and interpretations using simple column data. After narrowing down the dataset with signal analysis visualizations, LAnoBERT shows a way to remove the dependency on log parsers, feature embedding for long-term dependencies, a model for valuable anomaly detection, and a foundation for creating a model for understanding log messages that contain elements of natural and artificial languages. PULL further elevates the reactive anomaly detection problem setup and sets the gold standard for approaching reactive anomaly detection. Combined with the tools and knowledge already provided by existing methods, these ideas create a first level of log analysis that helps engineers focus their search on a manageable subset of essential information from an overwhelmingly large dataset.

5.2 Extracting Valuable Information from Focused Dataset

After narrowing the focus to the most relevant information, we can now call upon methods with more computational complexity. Utilizing embeddings, we can extract the most anomalous message using an isolation forest and cluster categories of messages. From here, methods can summarize clusters and generate insightful labels. All the information collected for this specific dataset merged with general knowledge about the log system provides a knowledge base for a retrieval augmented generation large language model. This RAG LLM can directly interact with engineers to help look up information, interpret results, and provide suggestions. Combining the ability to talk to the dataset with a log analysis application that provides other valuable functionality and information will revolutionize system diagnostics, leading to more efficient and accurate analysis. Enhancing the depth of analysis and improving the cost-efficiency of log examination will empower engineers to elevate their focus to higher-order tasks and strategic initiatives.

5.3 Noteworthy Discoveries

During our investigation, remarkable discoveries emerged, particularly in the context of anomaly detection within log files. Notably, the frameworks presented by LAnoBERT and PULL distinguish themselves by addressing the nuanced complexities inherent in log file analysis. Traditional methodologies often simplify anomaly detection to a binary classification task, relying on datasets with an explicit error-level column for categorization. While these approaches retain their merit, they frequently fall short in scenarios devoid of explicit indicators, a common occurrence in real-world datasets.

The innovative approaches employed by LAnoBERT and PULL offer a refreshing perspective on this challenge. LAnoBERT, with its foundation in the BERT architecture, leverages deep learning to comprehend the intricate semantics of log files, moving beyond mere surface-level analysis. This method demonstrates a significant stride in identifying anomalies within logs by understanding their contextual nuance rather than relying solely on predefined error levels.

Similarly, PULL introduces a novel paradigm by adopting a reactive anomaly detection strategy that is particularly adept at managing imbalanced datasets.

Traditional methods often struggle with high false positive rates in such contexts, leading to inefficient anomaly detection processes. PULL's approach mitigates this issue by focusing on the underlying patterns within the data, thereby enhancing the precision of anomaly detection.

These methodologies significantly depart from conventional binary classification models, offering a more holistic and effective strategy for anomaly detection in log files. Their ability to adapt to the unique challenges of imbalanced datasets and the absence of explicit error-level indicators underscores their value and ingenuity in advancing the log file anomaly analysis field.

Our research unveiled compelling insights that significantly influenced our analytical approach and methodology. Foremost among these was signal analysis's remarkable simplicity and efficacy. This technique emerged as our preferred method for distilling the dataset, facilitating a more focused and refined subsequent level of analysis. Its ability to efficiently parse and prioritize data for further scrutiny underscored the power of integrating straightforward analytical methods into more complex investigative frameworks.

Additionally, our methodologies' adeptness in deciphering log messages, characterized by a blend of natural and artificial language components, was noteworthy. This capability highlighted our models' advanced understanding and interpretation skills, which could navigate the intricate interplay between human-readable text and system-generated codes. This nuanced comprehension is pivotal for extracting actionable insights from log data, bridging the gap between human analysts and machine-generated logs.

Moreover, our exploration revealed substantial prospects for developing systems automating routine analytical tasks. Such systems can transform the landscape of log analysis, offering avenues to streamline processes, enhance efficiency, and allow engineers and analysts to allocate their expertise to more complex, high-value challenges. These findings underscore our methodologies' transformative impact on log file analysis, heralding a new era of efficiency and effectiveness in data interpretation and anomaly detection.

5.4 Challenges

While comprehensive in its scope and ambition, our investigation encountered certain constraints that must be acknowledged. Firstly, realizing the proposed architecture delineated within this paper remains theoretical. The transition from concept to implementation is anticipated to be a substantial endeavor, necessitating extensive development time. During this phase, unforeseen challenges inherent in the practical application of theoretical models will likely surface, adding complexity to the project.

Moreover, our project focuses on developing an analytical tool capable of examining and interacting with log datasets. However, it stops short of integrating automated resolution steps. This delineation means that while our tool can identify and highlight anomalies or patterns within the data, the user remains responsible for executing resolution strategies.

Furthermore, akin to the premise underpinning the PULL model, our approach is predicated on the assumption that the dataset under analysis contains the answers to the questions posed. This presupposes the presence of relevant data within the dataset and the dataset's comprehensiveness in encompassing the necessary information for analysis. Consequently, the efficacy of our methodology is inherently tied to the availability and relevance of the provided dataset and documentation for the log system.

Without a dataset that aligns closely with the analytical objectives, the tool's utility could be compromised, limiting its applicability to scenarios where comprehensive and pertinent data is readily available.

5.5 Future Research

Our work opens avenues worth exploring in future research, each with the potential to refine log analysis methodologies significantly. A key area involves developing methods that incorporate contextual understanding from the start of the automated analysis. Future studies could investigate how starting with a clear context can lead to more targeted analysis, focusing efforts on the most relevant log events. This approach promises to make data analysis more efficient and relevant, setting a precedent for context-driven investigations in various domains.

The *Kmeans* clustering algorithm required us to set the number of clusters manually. For a working implementation of the application proposed in this paper, an automated solution for setting the number of clusters is essential for streamlining the workflow.

Our holistic strategy—understanding complex problems deeply and integrating various methods—also offers valuable insights for various data science challenges. Future research could explore how this adaptable framework can be applied in different settings, potentially revealing universal strategies that enhance data science practices.

Another promising area is exploring streamlined models and innovative architecture. Future work could aim to develop models that maintain high accuracy while being more computationally efficient. This balance is crucial for making advanced analysis techniques more accessible and practical for broader applications.

Furthermore, methods designed for analyzing large datasets need to be refined to ensure efficiency and effectiveness optimization. This could lead to adaptations for real-time log analysis systems capable of swiftly identifying and addressing issues as they arise, enhancing system diagnostics and operational intelligence.

These future research directions build on our findings, leading to innovative advancements in log analysis and broader data science fields.

5.6 Ethics

Ethical considerations are paramount in deploying machine learning models, especially concerning data privacy and security. Ensuring ethical compliance involves several critical measures: maintaining transparency and securing explicit consent from users and system owners regarding data collection and analysis; adhering to data minimization principles by processing only the essential information; establishing robust accountability and auditability frameworks to monitor data access and safeguard against unauthorized use; ensuring data integrity to prevent detrimental alterations or manipulations; implementing anonymization or pseudonymization techniques to protect sensitive information; enforcing strict access controls to limit data access to authorized personnel only; ensuring full compliance with relevant legal and regulatory standards; and fostering interdisciplinary collaboration with experts in ethics, law, and social sciences to assess the ethical implications of log analysis practices holistically. These steps collectively ensure that the application of machine learning in log file analysis upholds the highest ethical standards, safeguarding the interests and privacy of all stakeholders involved.

6 Conclusion

Log files, often called the "system diaries," have been integral to diagnostics and monitoring for decades. Their unusual complexity, diversity, lack of context, and the sheer volume of data necessitate the evolution of methods and tools for efficient analysis. Our research delved deep into the core challenges of log files, especially regarding interpretability, adaptability, and the methodologies used for preprocessing, feature embedding, anomaly detection, diverse analysis, and insights into root causes and potential solutions.

By addressing these challenges, we aim to revolutionize how we interpret and analyze log files, paving the way for more efficient and accurate anomaly detection in systems and enhancing the overall reliability and performance of the systems they monitor at lower costs. We encourage the academic and professional community to build upon the research findings and continue to innovate by developing an accurate problem setup and combining methods into holistic architectures.

Acknowledgements. We thank the Southern Methodist University faculty and Paul Huggins for their guidance and support throughout this research.

References:

Cohen-Shapira, N., & Rokach, L. (2020). *Automatic selection of clustering algorithms using supervised graph embedding*. https://github.com/noycohen100/MARCO-GE

Devlin, J., Chang, M.-W., Lee, K., Google, K. T., & Language, A. I. (2018). *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*. https://github.com/tensorflow/tensor2tensor

Du, M., Li, F., Zheng, G., & Srikumar, V. (2017). DeepLog: Anomaly Detection and Diagnosis from System Logs through Deep Learning. *CCS '17 Proceedings of the 2017 ACM SIGSAC Conference*, 1285–1298. https://doi.org/https://doi.org/10.1145/3133956.3134015

Gemini Team, G. (2024). *Gemini 1.5: Unlocking multimodal understanding across millions of tokens of context*.

Gu, A., & Dao, T. (2023). *Mamba: Linear-Time Sequence Modeling with Selective State Spaces*.

He, P., Zhu, J., Zheng, Z., & Lyu, M. R. (2017). Drain: An Online Log Parsing Approach with Fixed Depth Tree. *Proceedings - 2017 IEEE 24th International Conference on Web Services, ICWS 2017*, 33–40. https://doi.org/10.1109/ICWS.2017.13

Jieming Zhu, Shilin He, Pinjia He, Jinyang Liu, & Michael R. Lyu. (2023). *Loghub: A Large Collection of System Log Datasets for AI-driven Log Analytics*.

Karlsen, E. (2023). *EXPLORATION OF NLP-BASED FEATURE EXTRACTION TECHNIQUES FOR SECURITY ANALYSIS AND ANOMALY DETECTION OF SERVICE LOGS*.

Le, V.-H., & Zhang, H. (2021). *Log-based Anomaly Detection Without Log Parsing*. https://doi.org/10.1109/ASE51524.2021.00051

Lee, Y., Kim, J., & Kang, P. (2023). LAnoBERT: System log anomaly detection based on BERT masked language model. *Applied Soft Computing*, *146*. https://doi.org/10.1016/j.asoc.2023.110689

Lu, S., Wei, X., Li, Y., & Wang, L. (2018). Detecting Anomaly in Big Data System Logs Using Convolutional Neural Network. *IEEE*. https://doi.org/10.1109/DASC/PiCom/DataCom/CyberSciTec.2018.00037

Mahowald, K., Ivanova, A. A., Blank, I. A., Kanwisher, N., Tenenbaum, J. B., & Fedorenko, E. (2023). *Dissociating language and thought in large language models: a cognitive perspective*. http://arxiv.org/abs/2301.06627

Meng, W., Liu, Y., Zhu, Y., Zhang, S., Pei, D., Liu, Y., Chen, Y., Zhang, R., Tao, S., Sun, P., & Zhou, R. (2019). *LogAnomaly: Unsupervised Detection of Sequential and Quantitative Anomalies in Unstructured Logs*. https://doi.org/https://doi.org/10.24963/ijcai.2019/658

Qi, J., Huang, S., Luan, Z., Fung, C., Yang, H., & Qian, D. (2023). *LogGPT: Exploring ChatGPT for Log-Based Anomaly Detection*.

Staudemeyer, R. C., & Morris, E. R. (2019). *Understanding LSTM-a tutorial into Long Short-Term Memory Recurrent Neural Networks*.

Thomas G. Dietterich. (2023, July 10). *What's wrong with LLMs and what we should be building instead*. https://www.youtube.com/watch?v=cEyHsMzbZBs&t=760s

Vaswani, A., Brain, G., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., & Polosukhin, I. (2017). *Attention Is All You Need*.

Wittkopp, T., Scheinert, D., Wiesner, P., Acker, A., & Kao, O. (2023). *PULL: Reactive Log Anomaly Detection Based On Iterative PU Learning*.

X. Zhang, Y. X. Q. L. B. Q. H. Z. Y. D. C. X. X. Y. Q. C. Z. L. et al. (2019). Robust log-based anomaly detection on unstable log data. *ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, *27*, 807–817.

Zhu, J., He, S., He, P., Liu, J., & Lyu, M. R. (n.d.). *Loghub: A Large Collection of System Log Datasets for AI-driven Log Analytics*. Retrieved December 3, 2023, from https://github.com/logpai/loghub.

Appendix 1:

Output of cluster summary:

```
Cluster 1 Anomalies: The error log messages provided indicate several issues occurring with the Azure
Service Bus, specifically with the queue named "order_pages_fetch_queue" within the namespace
"mcapseamaasdev." The following patterns and anomalies have been identified:

1. **Idle Timeout Expired**:
   - The `AmqpControlProtocolClient` has experienced an idle timeout after a period of 10 minutes
(`00:10:00`). This could suggest that the client was expecting messages or acknowledgments that did not
occur within the expected timeframe.

2. **Aborted Messaging Objects**:
   - Multiple instances of messaging objects being aborted are present in the logs. The reasons for the
aborts are primarily due to the `AmqpControlProtocolClient` being aborted. This could be a consequence
of the idle timeout issue or other underlying connectivity problems.

3. **Different Object Types Affected**:
   - The aborts are affecting various types of messaging objects, including:
     - `FlowControlMessageSender`
     - `FlowControlMessageReceiver`
     - `BrokerMessageBrowser`
     - `PeekLockMessageReceiver`
     - `ReceiveAndDeleteMessageReceiver`
   - This indicates that the issue is not isolated to a single type of operation or object within the
Service Bus but is affecting multiple aspects of message handling.

4. **Communication Object in Faulted State**:
   - An instance of `AmqpControlProtocolClient` is in a faulted state, which means it cannot be used
for communication
Faulting messaging object due to an error. Name = mcapseamaasdev:Queue
```

Appendix 2:

Psuedo Code for LAnoBERT Inference Algorithm:

```
// Initialization
KEY ← ∅, DICT ← ∅, j = 0
        Input  : l_n
Output : abnormal_loss , abnormal_prob
for n = 1 to N do
    // log key matching
    if l_n not in DICT then
        score_loss ← ∅ , score_prob ← ∅
        for i = 1 to s_len do
            l_{n,i} = [MASK]
            loss_j, prob_j ← LAnoBERT(l_n)
            score_loss ← score_loss ∪ {loss_i}
            score_prob ← score_prob ∪ {prob_i}
        end
        abnormal_loss ← TOPK(score_loss)
        abnormal_prob ← TOPK(score_prob)
        key_j = l_n
        KEY ← KEY ∪ {key_j}
        DICT[key_j] = (abnormal_loss , abnormal_prob)
        j ← j + 1
        return abnormal_loss , abnormal_prob
    else
        abnormal_loss , abnormal_prob ← DICT[l_n]
        return abnormal_loss , abnormal_prob
    end
end
```

(Lee et al., 2023)

Appendix 3:

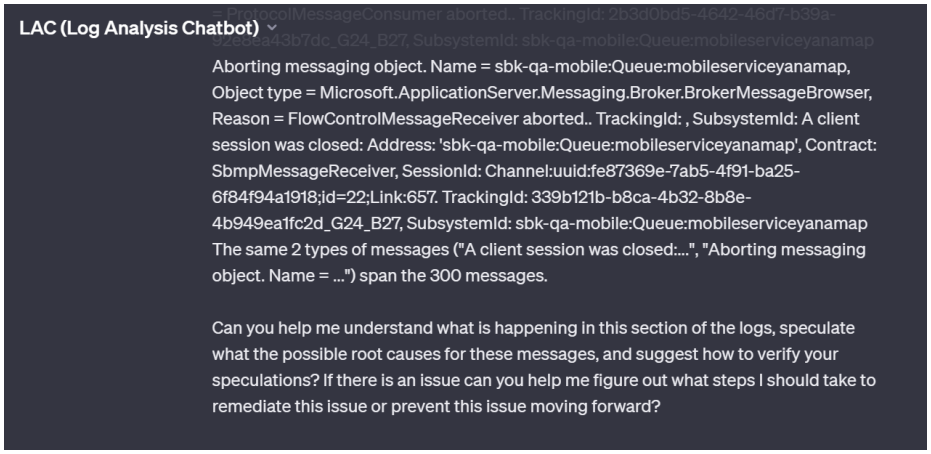
Cluster Labels
Cluster groupings that are produced from Log Analysis Assistant

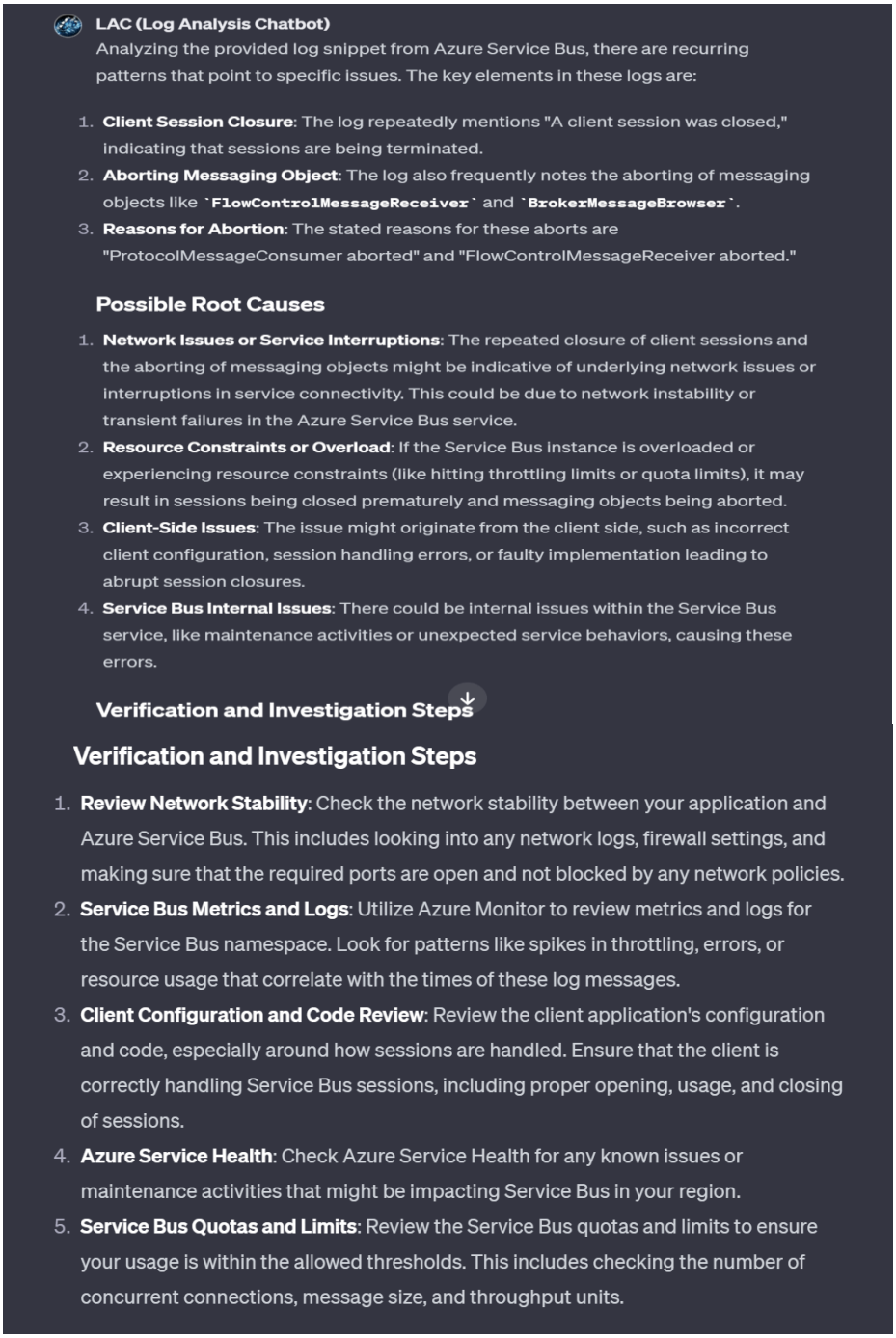
Cluster 0: Negative Entity Size and Consistent Entity Name Issues
Cluster 1: Idle Timeout and Aborted Messaging Objects
Cluster 2: Entity Availability and Timeout Issues
Cluster 3: QuotaExceededException and Entity Size Exceeded
Cluster 4: Messaging Objects Aborted
Cluster 5: Timeout Issues and Messaging Entity Not Found
Cluster 6: Idle Timeout Expired and Consistent TrackingId
Cluster 7: IdleTimerExpired Errors
Cluster 8: AMQP Links Force Detached due to Idle Timeout

Appendix 3:

Summarize: Analytical Application of LAA (Log Analysis Assistant)

This example demonstrates an initial test of the Log Analysis Assistant (LAA), formerly known as LAC. We input log data we deemed relevant into a custom GPT setup, which lacked a knowledge base about Azure Service Bus and its troubleshooting, as well as log file analysis specifics. These results were encouraging as they provided a general but helpful breakdown of the logs.





LAC (Log Analysis Chatbot)

Analyzing the provided log snippet from Azure Service Bus, there are recurring patterns that point to specific issues. The key elements in these logs are:

1. **Client Session Closure**: The log repeatedly mentions "A client session was closed," indicating that sessions are being terminated.
2. **Aborting Messaging Object**: The log also frequently notes the aborting of messaging objects like `FlowControlMessageReceiver` and `BrokerMessageBrowser`.
3. **Reasons for Abortion**: The stated reasons for these aborts are "ProtocolMessageConsumer aborted" and "FlowControlMessageReceiver aborted."

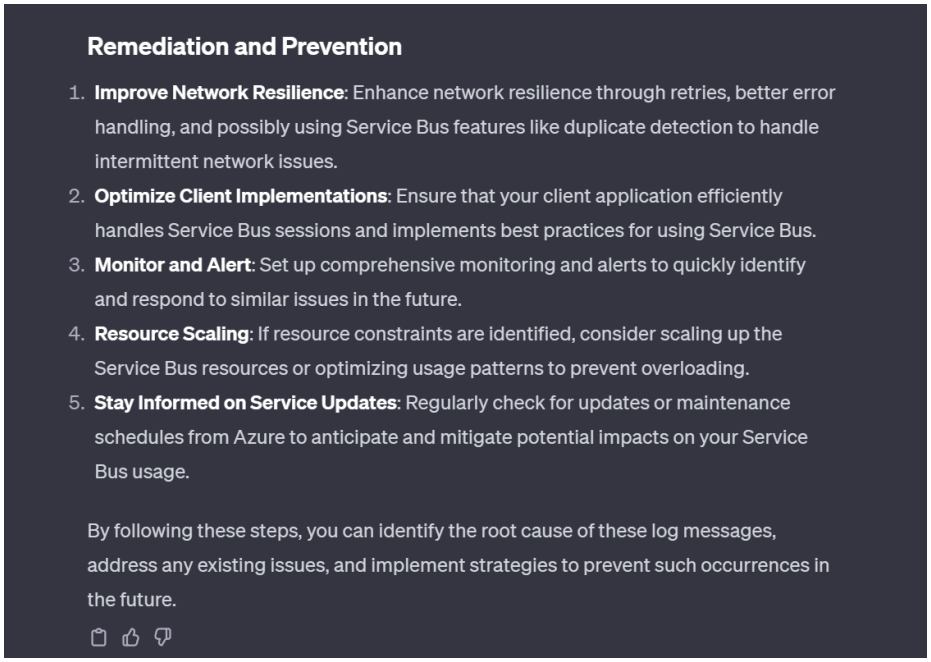
Possible Root Causes

1. **Network Issues or Service Interruptions**: The repeated closure of client sessions and the aborting of messaging objects might be indicative of underlying network issues or interruptions in service connectivity. This could be due to network instability or transient failures in the Azure Service Bus service.
2. **Resource Constraints or Overload**: If the Service Bus instance is overloaded or experiencing resource constraints (like hitting throttling limits or quota limits), it may result in sessions being closed prematurely and messaging objects being aborted.
3. **Client-Side Issues**: The issue might originate from the client side, such as incorrect client configuration, session handling errors, or faulty implementation leading to abrupt session closures.
4. **Service Bus Internal Issues**: There could be internal issues within the Service Bus service, like maintenance activities or unexpected service behaviors, causing these errors.

Verification and Investigation Steps

Verification and Investigation Steps

1. **Review Network Stability**: Check the network stability between your application and Azure Service Bus. This includes looking into any network logs, firewall settings, and making sure that the required ports are open and not blocked by any network policies.
2. **Service Bus Metrics and Logs**: Utilize Azure Monitor to review metrics and logs for the Service Bus namespace. Look for patterns like spikes in throttling, errors, or resource usage that correlate with the times of these log messages.
3. **Client Configuration and Code Review**: Review the client application's configuration and code, especially around how sessions are handled. Ensure that the client is correctly handling Service Bus sessions, including proper opening, usage, and closing of sessions.
4. **Azure Service Health**: Check Azure Service Health for any known issues or maintenance activities that might be impacting Service Bus in your region.
5. **Service Bus Quotas and Limits**: Review the Service Bus quotas and limits to ensure your usage is within the allowed thresholds. This includes checking the number of concurrent connections, message size, and throughput units.



Remediation and Prevention

1. **Improve Network Resilience**: Enhance network resilience through retries, better error handling, and possibly using Service Bus features like duplicate detection to handle intermittent network issues.
2. **Optimize Client Implementations**: Ensure that your client application efficiently handles Service Bus sessions and implements best practices for using Service Bus.
3. **Monitor and Alert**: Set up comprehensive monitoring and alerts to quickly identify and respond to similar issues in the future.
4. **Resource Scaling**: If resource constraints are identified, consider scaling up the Service Bus resources or optimizing usage patterns to prevent overloading.
5. **Stay Informed on Service Updates**: Regularly check for updates or maintenance schedules from Azure to anticipate and mitigate potential impacts on your Service Bus usage.

By following these steps, you can identify the root cause of these log messages, address any existing issues, and implement strategies to prevent such occurrences in the future.

Appendix 4:

Adding Knowledge to LAA (Log Analysis Assistant)

Enhancements to the knowledge base resulted in more precise and higher quality outcomes. The Log Analysis Assistant (LAA) can now offer links to pertinent documents and reference specific Azure Service Bus tools and terminology. However, the effectiveness of the LAA depends on the provided data. Without detailed examples of troubleshooting steps, it still falls short of delivering clear, step-by-step remediation guides.

- **Azure Service Bus Monitoring**: Utilize Azure Monitor and Service Bus metrics to gain insights into performance and operational health.
- **Azure Service Bus Best Practices**: Review and implement Azure Service Bus best practices to optimize performance and reliability.
- **Incident Response Plan**: Develop a comprehensive incident response plan for identifying, addressing, and mitigating issues in real-time.

By addressing these issues with targeted strategies, the stability and performance of the messaging infrastructure can be significantly improved.