Heterogeneous Syslog Analysis: There Is Hope

Andres Quan
aquan@lanl.gov
Los Alamos National Laboratory
Los Alamos, New Mexico, USA
The University of New Mexico
Albuquerque, New Mexico, USA

Leah Howell
leahh@lanl.gov
Los Alamos National Laboratory
Los Alamos, New Mexico, USA
The University of North Carolina at Chapel Hill
Chapel Hill, North Carolina, USA

Hugh Greenberg
hng@lanl.gov
Los Alamos National Laboratory
Los Alamos, New Mexico, USA

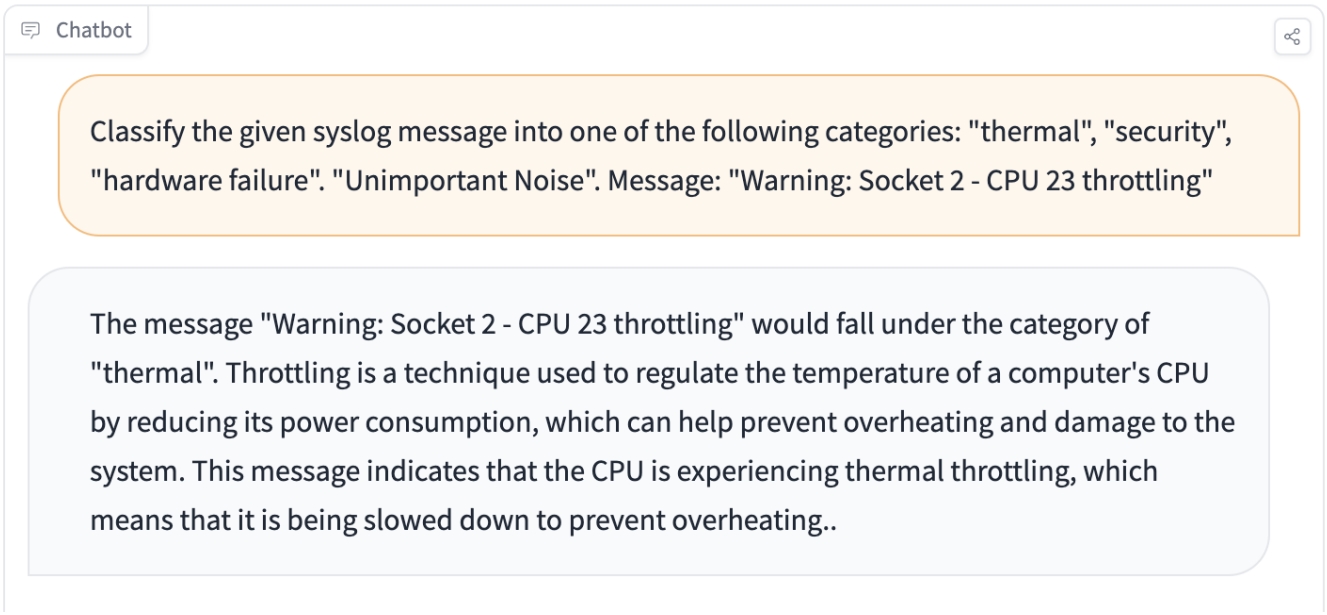

Figure 1: A simplified example syslog message analyzed by llama2-70b-chat-hf

ABSTRACT

Heterogeneous test-bed clusters present a unique challenge in identifying system hardware failures and anomalies as a result of the variation in the ways that errors and warnings are reported through the system log. We present a novel approach for the real-time classification of syslog messages, generated from a heterogeneous test-bed cluster, to proactively identify potential hardware issues and security events. By integrating machine learning models with high-performance computing systems, our system facilitates continuous system health monitoring. The paper introduces a taxonomy for classifying system issues into actionable categories of problems, while filtering out groups of messages that the system administrators would consider unimportant "noise". Finally we experiment with using newly available large language models as a form of message classifier, and share our results and experience with doing so. Results demonstrate promising performance, and more explainable results compared to currently available techniques, but the computational costs may offset the benefits.

SC 2023, SYSPROS-23, November 12–17, 2023, Denver, CO

ACM ISBN 978-1-4503-XXXX-X/18/06…$15.00
https://doi.org/XXXXXXX.XXXXXXX

CCS CONCEPTS

• **Computer systems organization** → **Maintainability and maintenance**; • **Security and privacy** → Intrusion detection systems; • **General and reference** → *Reliability*; • **Applied computing** → System forensics; • **Hardware** → *Temperature monitoring*; Enterprise level and data centers power issues; **Error detection and error correction**.

KEYWORDS

Error detection, Failure detection, Syslog, Heterogeneous Clusters, Test-beds, Cross-platform Software, Monitoring, Log Analysis, Applications of Large-Language-Models

ACM Reference Format:
Andres Quan, Leah Howell, and Hugh Greenberg. 2023. Heterogeneous Syslog Analysis: There Is Hope. In *Proceedings of Make sure to enter the correct*

conference title from your rights confirmation email (SC 2023, SYSPROS-23). ACM, New York, NY, USA, 7 pages. https://doi.org/XXXXXXX.XXXXXXX

1 INTRODUCTION

In the past, monitoring of syslog data has proven useful for the detection and diagnosing of various categories of system level issues ranging from thermal imbalances or CPU throttling, instabilities or failures in power supplies, throttling on network interface cards as a result of excess heat or congestion, detection of security events and more. The problem of monitoring and processing syslog data primarily exists as a result of the quantity of data produced. In just an hour over a million messages can be produced in a small scale test-bed like Darwin [9]. The problem is further exacerbated by the heterogeneity of a test-bed and the differences in the ways that syslog messages are reported from one vendor to the next, and over time as software and firmware components are upgraded.

Previous attempts at automated systems that process these syslog messages have involved traditional machine learning methods such as "N-gram" analysis [6], and methods which are rooted in statistical detection of anomalies in syslog messages. These techniques are used to conduct categorization and anomaly detection for the logs. This has resulted in tools such as "LOGAN" [3, 4] which was used for a short while at the Darwin cluster at LANL.

Commercial solutions have also been implemented across the Tri-lab, by making use of software like Splunk which is capable of conducting some of these machine learning techniques, but not all groups within the laboratory have access to these commercial tools. Recently, the surge in development of widely available open-source large language models provides a promising means of conducting analysis on the syslog data that is collected. Large language models have been provided by companies like Meta with their release of the llama models. The community has been improving upon these models in the form of weight deltas, and also producing their own distinct models.

In in this paper we discuss our experiences with using traditional natural language processing techniques to conduct our classification, and share our results in doing so. We also evaluate large models which are publicly available such as falcon-40b [2] (which was at the top of the large language model leaderboard at the time of evaluation), and falcon-7b. to look at the performance of classifying this data using this new tool for natural language processing.

2 RELATED WORKS

System logs are important in informing administrators about a system's health, security, and resource management. Detecting and diagnosing anomalies in the system logs helps system admins maintain the efficiency and reliability of a cluster. This section explores various approaches and methodologies proposed in the literature for addressing this challenge. It also emphasizes the need for research specifically targeting heterogeneous systems, which present unique challenges due to the increased variance in log messages.

Liang et al. [16] compare several supervised methods for anomaly detection, including RIPPER (a rules-based classifier), SVM, and a custom naive Bayes model. The study finds that the custom naive Bayes model performs the best. Studiawan and Sohel [20] extend the comparison to include additional supervised methods, such as decision tree, logistic regression, SVM, LSTM, and CNN. The study shows that deep learning methods, specifically LSTM and CNN, outperform other supervised methods. Furthermore, data balancing is identified as a critical factor for improving anomaly detection performance, with ADASYN and random oversampling being recommended for oversampling and Tamek links for undersampling.

In terms of anomaly diagnosis, Tuncer et al. [22] compare decision trees, random forest, and adaptive boosting methods. The study identifies random forest as the best-performing method for diagnosing anomalies. However, the study notes a decrease in F1 score when dealing with unknown input configurations and applications, highlighting the challenge of handling a dynamic environment. Aksar et al. [1] investigate gradient boosting machines and random forest for anomaly diagnosis and shows similar findings, with the former outperforming the latter. Similar to Tuncer et al. [22], the decrease in F1 scores for unknown input configurations and applications raises concerns about the robustness of these methods in heterogeneous environments.

Li et al. [15] propose a modified naive Bayes method and hidden Markov models for anomaly diagnosis, both of which incorporate temporal data to improve categorization performance. The utilization of temporal information in these models indicates its potential to capture dynamic patterns and improve the accuracy of anomaly diagnosis.

Studiawan and Sohel [20] show that supervised models outperform isolation forest and PCA. Du et al. [7] introduce DeepLog, a semi-supervised model that utilizes log key and parameter value vectors for analyzing and detecting anomalies. DeepLog is trained based on normal patterns in system logs, providing incremental updates for adaptability. Du et al. [7] show that DeepLog outperforms isolation forest and PCA as well. Zope et al. [24] compare supervised and unsupervised models for anomaly diagnosis. While PCA stands out as the best-performing unsupervised model, it is still outperformed by supervised methods.

The existing methods, including the "LogAn" tool developed at Los Alamos National Laboratory, demonstrate limitations in detecting anomalies in heterogeneous systems. LogAn takes user feedback to become more accurate, and it gives context for anomaly diagnosis [3, 4]. However, LogAn needs constant retraining on our system. LogAn was designed for a homogeneous system such as Trinity, while ours is very heterogeneous. The heterogeneity of our system causes there to be a lot of variance in messages, making the methods used in LogAn to be less effective. This finding emphasizes the need for research focusing on heterogeneous systems to overcome the challenges varying log messages pose.

Sun and Xu [21] present LogPal, a generic anomaly detection scheme designed for heterogeneous logs in network systems. LogPal adopts the synthetic attention transformer encoder network, which reduces noise influence and enhances the identification of relevant semantics in log sequences. This approach shows promise for addressing the complexities of HPC systems with diverse log messages. Hajamydeen et al. [10] demonstrate that clustering techniques can effectively reduce false positives and false negatives in heterogeneous data.

3 BACKGROUND

Previously with the Darwin cluster [9] we have had moderate success by classifying issues into category buckets using minimum edit distance based metrics like Levenshtein distance and Hamming distance [11, 13] to group syslog messages into buckets, where each bucket contains strings that are similar below a certain minimum edit distance threshold. The resulting buckets would contain messages that were all similar with the exception of a few tokens. This means that messages that state the same problem in the same way, but with slightly different identifying information would be grouped into the same bucket. The resulting buckets would have an exemplar message that can be used to collect other messages into the same bucket. Those buckets would later be grouped in to "issue" categories such as "Thermal", "Slurm", "Memory", "USB", "Network", etc. The issue categories could be set to trigger a notification email when a new message within that category has been identified. Overall, this simple approach was useful at identifying transient thermal issues and nodes that were having memory issues that we hadn't previously identified.

The problem with this approach became that as time went on, and systems received new firmware updates, or new systems would be added to the test-bed and old systems were retired, the semantics and syntax of the messages would differ slightly which would produce new buckets in the queue that needed to be classified. This continuous re-training process would consume valuable system administrator time.

As a separate effort within LANL's HPC division a tool known as LOGAN [3, 4] (short for Log Analysis) was created which served a similar purpose. Their project sought to find anomalous and interesting syslog messages and report them to the system administrators through a custom Grafana [12] interface that would allow us to mark messages as being interesting or uninteresting. Although their techniques would appear to be effective for homogeneous systems like the Trinity supercomputer, after some initial training it was clear that this system would have the same problems on Darwin with the continued need to conduct re-training that we were experiencing with the minimum edit distance techniques. There is a need for a solution that is more robust to the changes seen to these syslog messages over time.

4 METHODS

4.1 Defining Categories

As a system administrator managing a heterogeneous test-bed cluster, we don't find it beneficial to over-specify the type of problems a system may be encountering, based on the classification of syslog messages. This is primarily due to two reasons. First, we lack comprehensive syslog data on all potential hardware issues that a system may face. Second, for our purposes, a syslog serves as an initial step towards problem-solving, rather than an absolute determination of the situation. We prefer issues to be classified in a more generalized manner, at a level that prompts actionable steps.

To illustrate, let's consider a scenario where a system in our cluster repeatedly logs a specific error about memory allocation. Instead of classifying this issue as a 'segmentation fault' or 'bad memory address', which are very specific and require further data to confirm, we would categorize it under a broader category like 'Memory Issues'. This level of classification is sufficient for us to take the next step, which could involve running memory diagnostics or replacing the potentially faulty memory module.

As a result, we chose the following categories for our initial syslog classification scheme. "Hardware Issue", "Intrusion Detection", "Memory Issue", "SSH-Connection", "Slurm Issues", "Thermal Issue", "USB-Device", "Unimportant".

The classification was done in such a way that if the syslog appears to be Unimportant noise or just application specific information, then it is classified as "Unimportant". Then we reviewed the data for any indications of a thermal issue, a Slurm issue, a memory issue, or indications that the log would be useful for intrusion detection. Issues that indicate some kind of hardware issue that doesn't fit into any of the explicitly mentioned issue categories were classified as "Hardware Issue".

4.2 Infrastructure

To store syslog data from all the nodes in the Darwin cluster, we built a small Opensearch [19] cluster using standard server hardware. We call this system Tivan. Fluentd [8] was used to collect data from a syslog server and forward it to the Opensearch cluster. Grafana [12] is used to visualize the data store in Opensearch. This system has allowed us to store and search over thirty million log records a month.

4.2.1 Hardware. Our current hardware includes 8 Dell R530 servers with 128GB of DRAM and 4TB of storage per Opensearch node. The nodes are connected via a 1Gbps gigabit link.

Inference timings were collected from a single system consisting of four A100 SXM4 Nvidia GPUs each with 40GB of VRAM connected via NVLink with two AMD EPYC 7742 Rome processors.

4.2.2 Software. The system is composed of only open source components. Database support is provided by an Opensearch service deployed across 6 of the Dell servers. Data collection, filtering, and translation is implemented using Fluentd running on a dedicated server. Analysis and dashboard support is implemented using Grafana. Grafana is hosted on a separate server with additional network connectivity to support data analysis and dashboard access. The syslog data stream is forwarded from all our compute nodes to a primary syslog server which then forwards the stream to Fluentd. Forwarding is managed by rsyslogd's builtin support.

4.3 Preprocessing and Feature Engineering

4.3.1 TF-IDF. One of the things that causes trouble with regards to grouping together messages of similar meaning is that although messages of the same category might use the same words, they may use different syntax to describe the words that are unique to a particular category. This is one of the places where previously utilized techniques like Levenshtein distance grouping have been inefficient, and needed retraining. As an example consider the following two sentences: "CPU temperature above threshold, cpu clock throttled.", and "CPU 1 Temperature Above Non-Recoverable - Asserted. Current temperature: 95C". It's easy for a human to look at these and say that these two systems are experiencing similar issues, but our previous grouping technique would say that this has a Levenshtein

Table 1: The top 5 tokens from each category after TF-IDF

Category	Top Tokens
Hardware Issue	timestamp, sync, clock, system, event
Intrusion Detection	root, session, user, started, boot
Memory Issue	size, real_memory, low, cn, node
SSH Connection	closed, preauth, connection, port, user
Slurm Issue	version, update, slurm, please, node
Thermal Issue	processor, throttled, sensor, cpu, temperature
USB Device	usb, device, hub, number, new
Unimportant	error, lpi_hbm_nn, job_argument, slurm_rpc_node_registration

distance of 7 and thus needs to be grouped into a seperate bucket to later be assigned a category.

TF-IDF stands for Term Frequency Inverse Document Frequency [17] and is a technique for finding the words in a text that are most relevant for a particular set of text, amongst a corpus of text. In our specific case we can think of the "particular set of text" as being all of the messages within a certain category we are interested in classifying. Therefore the "corpus of text" is the combined set of messages in all of the categories. The resulting top words from running this process on our dataset is given in table 1. These words can now be used as features in the input vectors to our machine learning methods. In addition these words allow us as humans to look at the words that will be used as features and provide a degree of explain-ability for the classifications. In later sections we will discuss how these words are introduced in prompts to large language models to encode information about many syslog messages into a small prompt.

4.3.2 Lemmatization. To deal with differences in how certain vendors use different parts of speech with the same words to describe the same issue. We use a process called Lemmatization to group together these different parts of speech for a particular word in a message into the same root of the word also known as a "lemma".

To illustrate this technique consider the following sentences. "The system has failed", "There was a failure in the system", "The system is failing". These sentences use three different parts of speech for the verb "fail". Lemmatization turns all of these instances of the word into just the root lemma which is "fail"

Implementations of Lemmatization can differ slightly from one to the other, in our case we use the NLTK WordNet Lemmatizer [5] to perform this preprocessing step.

4.4 Trained Datasets

4.4.1 Levenshtien Distance Dataset. The dataset that we used was composed of a set of approximately 196 thousand unique messages that were collected and classified using the previously mentioned Levenshtein distance based metric with the distance threshold for message similarity set to 7. The data was classified over the course of a year, and the advantage to building our dataset in this way is that we really only needed to classify about 3415 messages which were the exemplars for a particular "bucket", and then the remaining messages were categorized by way of similarity to a pre-classified bucket exemplar.

Table 2: Number of unique messages per category

Category	Number of Messages
Hardware Issue	3582
Intrusion Detection	6599
Memory Issue	12449
SSH-Connection	3615
Thermal Issue	59411
Slurm Issues	46
USB-Device	4139
Unimportant	106552

4.4.2 Dataset Imbalance. As a result of the transient nature of issues in the system and the relative rarity of certain categories of issue it becomes difficult to capture and classify a dataset which is perfectly balanced. This furthers the important of techniques like TF-IDF to extract the features that are most relevant for identifying a particular class, because it allows us to equalize the number of features that identify a particular class, and reduce the bias on the side of the classifier.

4.5 Monitoring and Suggested UI

While the classification and identification of syslog issues is at the core of this research, it is also important to consider the different techniques for viewing and organizing the syslog data, and other performance metrics that help to diagnose issues in the test-bed. We will discuss four different categories of techniques that we have found to be helpful for finding and diagnosing issues in our test-bed, and some example use-cases for each technique where it could potentially be helpful to have such a view into the data.

4.5.1 Frequency and Temporal Analysis. Certain categories of issues, in particular memory issues, and thermal issues can result in behavior that causes surges of repeated messages to appear. A sudden influx of a large quantity of new syslog messages can be indicative of an issue. By visualizing syslog data as a graph that shows number of messages on one axis, and time in the other axis, you can identify points in time where something may have been going wrong and use this as a starting point to further investigate.

One can later use that point in time to correlate the issues to a potential cause such as: Someone leaving the door open in the cold isle containment system, and it resulting in thermal shutdown to various systems. Potentially from a security standpoint you could correlate someones access control to the data center room with a log that is identified as a security event, such as someone plugging in a USB device. You could also group syslog messages by node in the frequency analysis to get an idea for which machines specifically are suddenly being much more noisy. Alternatively you can group syslog by service to see which services are suddenly being noisy.

4.5.2 Positional Analysis. The physical placement of compute nodes within a data center can significantly influence system performance and the identification of problems. This is especially relevant in terms of networking and thermal management considerations. All

nodes within a single rack are typically connected to the same edge switch, which can impact network performance due to the shared bandwidth and potential bottlenecks. Thus understanding the physical topology can help in identifying potential issues. Thermal management is another important aspect that is influenced by the placement of compute nodes. Nodes within a rack share a similar micro-climate, with the cooling effectiveness and temperature being more or less the same for all nodes in the rack. Thus, physical location of compute nodes is a key factor in both network and thermal performance, and can play a vital role in monitoring and investigation of problems in a heterogeneous environment.

4.5.3 Per-architecture Analysis. One way of finding out if system behavior is anomalous in a test-bed is by comparing a given node to others within the cluster that have the same architecture. Occasionally if a node is experiencing an issue that appears to be interesting or relevant it may be a false indication. It's worth checking to see if the same message or data is appearing on other compute nodes with the same architecture to verify that the message or status is anomalous. One example where this has shown to be helpful is with data coming from IPMI. Fans or thermal sensors will occasionally report through IPMI that they are not functioning or the reading for those sensors are unusually high or low, however when comparing readings from other nodes from the same architecture the readings are exactly the same. As a result of the hardware being early access in a test-bed we tend to find instances where although chassis sensors are reporting that there is an issue or that something is missing or has failed, in reality the system is operating nominally.

5 RESULTS

In consideration of the following results. It's important to consider not only the classification accuracy of a given technique and its corresponding explainability, but also the time that it takes to execute a particular classification technique. Again, this is due to the extremely high volume of messages that are being ingested into the database at any given moment. Techniques that are able to perfectly classify a message with a great explanation for why it belongs in that category are useless to us, if they require so much computational power that we can only afford to classify a single message every 30 seconds.

5.1 Traditional NLP Techniques

We evaluate the traditional NLP models by comparing weighted-average F1-scores, training time, and testing times. The F1 scores for each class are calculated as the harmonic mean of the precision and recall scores. The weighted-averaged F1 score is better for imbalanced data, like ours, and is calculated by taking the mean of all per-class F1 scores while considering each class's support. Confusion matrices were also made for each model to indicate which categories were frequently mixed up.

We observe that all of the traditional NLP models achieve a high weighted F-1 score, (over 0.95), with the highest being Random Forest at 0.9995. The KNN approach achieves the fastest training time at about 0.0107 seconds, while Linear SVC takes the longest, around 211.78 seconds. Complement Naïve Bayes has the lowest testing time at around 0.0018 seconds.

In producing the confusion matrices, we observed that the most frequently confused category was "Unimportant," so we also experimented with removing those messages from the training and testing data. This caused all of the weighted F-1 scores to increase, with the highest being Linear SVC at around 0.99994, followed closely by Random Forest and kNN with both at around 0.99989. The training and testing times also decreased, with the training time for Linear SVC dropping the most, from 211.78 seconds to 2.213 seconds.

In figure 3 we see the confusion matrix for Linear SVC.

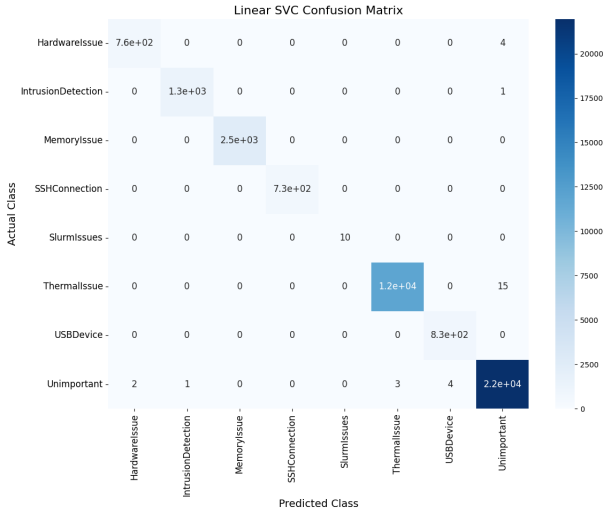

Figure 2: Confusion matrix for Linear SVC

We attribute the higher degree of confusion along the "Unimportant" category to be a result of messages that use significant words from other categories, but that aren't actually an interesting issue from the perspective of the system administrators that classified it. These results suggest a potential need to use a preprocessing step that is able to filter out this category of messages prior to classification. This could be done with the previously utilized minimum-edit distance techniques using a lower value for the categorization threshold. This could allow system administrators to "blacklist" specific kinds of messages while allowing the remaining messages and any new messages to use the more general classifier.

5.2 Large Language Models

Our initial expectations for open source LLMs were based on our experiences working with tools like ChatGPT [18] in other contexts and we were perhaps overly optimistic about the performance we could achieve with smaller models that are able to operate on a single node. Our plan was to feed a generative LLM a prompt that contained a classification task that was worded in the form of a multiple choice question. The multiple choice question was worded to include information about the various categories, and then provides the prompt that needs to be classified, with instructions that state that the resulting classification must be one of the given categories. In our evaluation we chose Falcon-40b [2] and falcon-7b because

Classifier	Weighted F1-Score	Training Time (sec)	Testing Time (sec)
Logistic Regression	0.9992	15.37882	0.005369
Ridge Classifier	0.9987	4.723393	0.00425
kNN	0.998475	0.010657	4.908732
Random Forest	0.9995	9.101409	0.605475
Linear SVC	0.99925	211.7842	4.82377
Log-loss SGD	0.987794	0.468163	0.002344
Nearest Centroid	0.952334	0.012665	0.007403
Complement Naïve Bayes	0.99751	0.022864	0.001801

Figure 3: Results from traditional classification techniques with TF-IDF Pre-processing

at the time of evaluation, Falcon-40b was at the top of the large language model leader-board, and it was distributed with an open access Apache2 license.

The first noticeable failures we were having with this approach were that we would frequently get a "generated classification" from the model where the chosen classification for the syslog message was an entirely new category that we hadn't previously defined, but that makes sense in the context of the message provided. This was frustrating because it makes the process of automating the parsing of the result more difficult.

Beyond the alignment issues of getting the models to correctly respond to multiple-choice questions we also had issues with excessive generation. This excessive generation could appear in the form of an unsolicited justification for why a certain classification was chosen. In one of the more interesting cases we had a result from a classification where the response included not only a classification for the given syslog message, but went on to also produce a new prompt that introduces a system administrator character, and then produced an artificial syslog message with instruction for the fictional character to conduct classification. This issue persisted on both Falcon-40b and Falcon-7b despite the inclusion of instructions that stated to only respond with one of the categories given. We resolved these particular issues by placing a limit on the number of new tokens that could be generated by the model. This is an important problem to ensure is resolved in future efforts because excessive generation can add unnecessary computational cost to the classification.

Ultimately the prompt that generated the most success in our testing contained the following elements: An introduction of the problem. a list of the potential categories. A list of the most commonly used words generated via TF-IDF for each category. A specification of the output format, and finally we also included an example syslog message with its corresponding classification in the output format expected, but the results were frequently incorrect, even for "larger" models like falcon-40b.

Above all, the computational cost is what makes using Large Language Models for this task infeasible. In Table 3 we show the inference timing information that we collected from the two generative models that we tested, along with one zero-shot model we looked at.

Table 3: Time in seconds for LLM generative classification of a single message

Hugging Face Model Name	Inference time	Messages per hour
Falcon-7b	.639	5633
Falcon-40b	2.184	1648
facebook/Bart-Large-MNLI	.13359	26948

Zero shot text classification [23] is an interesting alternative technique because it fixes the problems with "generated classification", and the need to format the classification in the form of a prompt but it still makes use of the power behind language models to analyze the text. Zero shot text classification involves feeding a model a piece of text, and a given set of categories, with no prior knowledge of those categories, and asking it to rate the text's classification into those categories based on the name of the category alone. Although zero-shot may be easier to use, one of the advantages behind open ended prompts for generative models over zero shot techniques is that we can still encode category specific details from feature extractors like TF-IDF [17] within the prompts. For our study we looked at the facebook/bart-large-mnli [14]. This particular model has a much faster execution time as you can see in Table 3, but is still insufficient for our purposes.

Despite the computational cost the ability that generative large language models have for providing a human readable explanation for a given classification is an interesting ability that we would like to explore further in the future.

6 CONCLUSION

Our work with applying natural language processing techniques and traditional machine learning techniques show that its possible to obtain acceptable classification performance for heterogeneous syslog classification. Using traditional techniques we are able to obtain a much higher inference throughput compared to large language models.

Timings taken from large language models on our inference node show that the computational cost involved makes their use extremely costly for our smaller cluster, since classification of the syslog messages via this technique on a single node will not be able to keep up with the continuous flow of messages without dedicating significantly more resources.

Higher degrees of confusion with the traditional models relative to the "Unimportant" category. Show a need to filter out messages that have been designated as acceptable to ignore, prior to classification of the remaining messages.

7 FUTURE WORK

The next immediate goal to work on in this project will be deploying our trained models on the new data we stored in our collection system and making use of those classifications to build new views into our system status. We will specifically be interested in how well this particular classification/pre-processing technique combination holds up to changes in our cluster's environment.

Although Large Language Models are too computationally expensive to deploy for this particular task, and there were difficulties as far as alignment on the task. We are hopeful that as progress is

made on large language models that there still might be use-cases for these tools in the context of a test-bed cluster. Some examples could be summarizing the system status, explanation of groups of syslog messages within a given node, generating recommended responses to admin emails based on system specific information, and other low frequency tasks. These models excel in tasks that involve unstructured text.

8 CONTRIBUTION STATEMENT

Leah Howell conducted research into currently available classification methods and wrote the code that uses traditional natural language processing techniques to classify the syslog messages. She conducted an evaluation of the performance of those traditional techniques and conducted an initial evaluation of the use of Falcon-40b and Falcon-7b as potential syslog classifiers. She also wrote the Related Works and results for Traditional NLP Techniques sections.

Hugh Greenberg helped us to stand up the infrastructure for the Tivan syslog and he wrote the infrastructure section of the paper.

Andres Quan wrote the remaining sections of this paper, and steered the direction of this research project. He also created the initial Levenshtein distance based classifier which was used to generate the dataset that was used for this study.

ACKNOWLEDGMENTS

We would like to thank David Rich for his help on this project. David Rich worked with Hugh Greenberg to properly configure the cluster to send messages into OpenSearch. David is also the PI for the team that manages the Darwin cluster at Los Alamos National Laboratory in the CCS-7 group (Computer Computational and Statistical Sciences Division - Applied Computer Science Group). He helped to classify the syslog messages from the initial dataset, and helped provide information for the infrastructure section.

We would like to thank Hugh Greenberg for helping us to setup the Tivan Opensearch cluster within Darwin, and for his work in setting up the Grafana front-end to view the syslog data as it flows into Tivan.

We would also like to thank Wesley Mason for his help in getting the network configured properly, and for his help in feeding network data into our syslog collection system.

This research used resources provided by the Darwin testbed at Los Alamos National Laboratory (LANL) which is funded by the Computational Systems and Software Environments subprogram of LANL's Advanced Simulation and Computing program (NNSA/DOE).

This work was supported by the U.S. Department of Energy through the Los Alamos National Laboratory. Los Alamos National Laboratory is operated by Triad National Security, LLC, for the National Nuclear Security Administration of U.S. Department of Energy (Contract No. 89233218CNA000001).

REFERENCES

[1] Burak Aksar, Benjamin Schwaller, Omar Aaziz, Vitus J. Leung, Jim Brandt, Manuel Egele, and Ayse K. Coskun. 2021. E2EWatch: An End-to-End Anomaly Diagnosis Framework for Production HPC Systems. In *Euro-Par 2021: Parallel Processing*, Leonel Sousa, Nuno Roma, and Pedro Tomás (Eds.). Springer International Publishing, Cham, 70–85. https://doi.org/10.1007/978-3-030-85665-6_5

[2] Ebtesam Almazrouei, Hamza Alobeidli, Abdulaziz Alshamsi, Alessandro Cappelli, Ruxandra Cojocaru, Merouane Debbah, Etienne Goffinet, Daniel Heslow, Julien Launay, Quentin Malartic, Badreddine Noune, Baptiste Pannier, and Guilherme Penedo. 2023. Falcon-40B: an open large language model with state-of-the-art performance. (2023).

[3] Elisabeth Baseman, Sean Blanchard, Zongze Li, and Song Fu. 2016. Relational Synthesis of Text and Numeric Data for Anomaly Detection on Computing System Logs. In *2016 15th IEEE International Conference on Machine Learning and Applications (ICMLA)*. 882–885. https://doi.org/10.1109/ICMLA.2016.0158

[4] Elisabeth Baseman and Lissa. 2016. Interpretable Anomaly Detection for Monitoring of High Performance Computing Systems. https://api.semanticscholar.org/CorpusID:51763551

[5] Steven Bird, Edward Loper, and Ewan Klein. 2009. *Natural Language Processing with Python*. O'Reilly Media Inc.

[6] William Cavnar and John Trenkle. 2001. N-Gram-Based Text Categorization. *Proceedings of the Third Annual Symposium on Document Analysis and Information Retrieval* (05 2001).

[7] Min Du, Feifei Li, Guineng Zheng, and Vivek Srikumar. 2017. DeepLog: Anomaly Detection and Diagnosis from System Logs through Deep Learning. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security (CCS '17)*. Association for Computing Machinery, New York, NY, USA, 1285–1298. https://doi.org/10.1145/3133956.3134015

[8] Fluentd. 2023. Fluentd: Open Source Data Collector - Unified Logging Layer. https://fluentd.org/ Accessed: 2023-07-31.

[9] Charles Kristopher Garrett. 2018. The Darwin Cluster. (6 2018). https://doi.org/10.2172/1441285

[10] Asif Iqbal Hajamydeen, Nur Izura Udzir, Ramlan Mahmod, and Abdul Azim Abd. Ghani. 2011. Filtering Events using Clustering in Heterogeneous Security Logs. *Information Technology Journal* 10, 4 (04 2011). https://doi.org/10.3923/itj.2011.798.806

[11] Karen Kukich. 1992. Techniques for Automatically Correcting Words in Text. *ACM Comput. Surv.* 24, 4 (dec 1992), 377–439. https://doi.org/10.1145/146370.146380

[12] Grafana Labs. 2023. Grafana: The open observability platform. https://grafana.com/ Accessed: 2023-07-30.

[13] Vladimir I. Levenshtein. 1965. Binary codes capable of correcting deletions, insertions, and reversals. *Soviet physics. Doklady* 10 (1965), 707–710. https://api.semanticscholar.org/CorpusID:60827152

[14] Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Veselin Stoyanov, and Luke Zettlemoyer. 2020. BART: Denoising Sequence-to-Sequence Pre-training for Natural Language Generation, Translation, and Comprehension. *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics* (2020). https://doi.org/10.18653/v1/2020.acl-main.703

[15] Tao Li, Feng Liang, Sheng Ma, and Wei Peng. 2005. An Integrated Framework on Mining Logs Files for Computing System Management *(KDD '05)*. Association for Computing Machinery, New York, NY, USA, 776–781. https://doi.org/10.1145/1081870.1081972

[16] Yinglung Liang, Yanyong Zhang, Hui Xiong, and Ramendra Sahoo. 2007. Failure Prediction in IBM BlueGene/L Event Logs. In *Seventh IEEE International Conference on Data Mining (ICDM 2007)*. 583–588. https://doi.org/10.1109/ICDM.2007.46

[17] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schutze. 2019. *Introduction to information retrieval*. Cambridge University Press.

[18] OpenAI. 2023. Chatbot: GPT-4, Language Model by OpenAI. https://www.openai.com/chat-gpt/. Accessed: 2023-07-30.

[19] OpenSearch. 2023. OpenSearch. https://opensearch.org/ Accessed: 2023-07-31.

[20] Hudan Studiawan and Ferdous Sohel. 2020. Performance Evaluation of Anomaly Detection in Imbalanced System Log Data. In *2020 Fourth World Conference on Smart Trends in Systems, Security and Sustainability (WorldS4)*. 239–246. https://doi.org/10.1109/WorldS450073.2020.9210329

[21] Lei Sun and Xiaolong Xu. 2023. LogPal: A Generic Anomaly Detection Scheme of Heterogeneous Logs for Network Systems. *Security and Communication Networks* 2023 (Apr 2023). https://doi.org/10.1155/2023/2803139

[22] Ozan Tuncer, Emre Ates, Yijia Zhang, Ata Turk, Jim Brandt, Vitus J. Leung, Manuel Egele, and Ayse K. Coskun. 2019. Online Diagnosis of Performance Variation in HPC Systems Using Machine Learning. *IEEE Transactions on Parallel and Distributed Systems* 30, 4 (2019), 883–896. https://doi.org/10.1109/TPDS.2018.2870403

[23] Wenpeng Yin, Jamaal Hay, and Dan Roth. 2019. Benchmarking zero-shot text classification: Datasets, evaluation and entailment approach. *arXiv preprint arXiv:1909.00161* (2019).

[24] Kalyani Zope, Kuldeep Singh, Sri Harsha Nistala, Arghya Basak, Pradeep Rathore, and Venkataramana Runkana. 2019. Anomaly Detection and Diagnosis In Manufacturing Systems: A Comparative Study Of Statistical, Machine Learning And Deep Learning Techniques. In *Proceedings of the Annual Conference of the PHM Society, 11(1)*. https://doi.org/10.36001/phmconf.2019.v11i1.815

Received 20 February 2007; revised 12 March 2009; accepted 5 June 2009