Detect failure root cause and predict faults from software logs

Stanford CS224N Project

Name Sandip Pal
Department of Computer Science (NDO)
Stanford University
sandipal@stanford.edu

Abstract

We aim to find anomalies and their root causes from system log data. Log files from computers speak a technical language that is sometimes limited to a series of 80-character sentences, but provides insights into what leads to a possible failure. We intend to map "anomaly" log detection as a problem in the domain of NLP analysis of computer components conversing and emitting signals of health, despair, and possibly panic. As a first step we aim to leverage zero shot classifiers to identify log anomalies and subsequently aim to leverage Large Language Models and the Retrieval Augmented Generation to devise a question-and-answer system.

1 Key Information to include

- Mentor: Tony Wang
- External Collaborators (if you have any):None

2 Introduction

We aim to find anomalies in computer logs and subsequently discover the root cause that led to that anomaly. The anomalies in this context are "failures" or fatal errors. Software engineers write code to build systems and they insert logs (print them to the console or file or stream) to debug in case of any error. As the systems grow in complexity the amount of log that accumulates is of considerable volume that can stretch over hours to days and possibly months. When an error occurs in a system, it becomes a necessary activity to find the root cause, and the logs are studied to identify when such an error occurred and what led to the error. The job of manually going through such logs is laborious and time-consuming and requires an understanding of the system context. Due to the difficulty in analyzing such voluminous logs, automation is the best choice for repeated log analysis.

There are existing methods that have been tried out to find the "root cause" by analyzing log sections by conducting a semantic search. The semantic search and root cause associations are heavily dependent on the type of log that it is built for. With advances in NLP, the problem can be mapped to "processing" computers/systems **talking** and broadcasting information, warnings, pain, fatalities etc. "Sentiment" detection is a similar problem space as compared to our primary problem of log fault or anomaly analysis.

With the recent progress with LLMs, log anomaly identification can leverage the existing relationship of words already present in the base large language model. The technique of fine-tuning a pre-existing language model is a candidate to solve our problem. Retrieval Augmented Generation (RAG) techniques to create a prompt for the large language model with a local vector "database" and answer the "anomaly" question will be another approach.

The problem becomes hard when the log output doesn't follow conventional "English" language rules. Some systems output cryptic logs that follow an encoding scheme, which implies the training system

and the query system need the decoder to understand the logs. Zero shot classifier techniques will not work on an encoded log system. Log systems that produce "non-english" logs will need a translator (auto or manual) to be present to feed an analyzer system.

3 Related Work

This section helps the reader understand the research context of your work by providing an overview of existing work in the area.

Pan et al. (2023) authors have carried out a similar investigation by trying to solve the detection of log anomaly problems by using retrieval augmented generation and building a vector database with normal logs. The architecture was to populate a RAG vector database with normal logs and use chatGPT as the base LLM and subsequently answer the log anamoly context

He et al. (2016) describes various methods and the experience around studying different logs and extracting features from them. Conventional log parsing, with context windows, feature extractions and event detection were primarily used to study the logs. Statistical methods involving logistic regression , Principal component analysis and supervised learning methods like support vector machines were used to detect anomalies and study their effectiveness.

4 Approach

- **Approach 1** NLI(Natural langiuage Inference) based zero shot classifier
  To detect an "anomaly" log, we will use the pre-trained "facebook/bart-large-mnli" and classify each line of log as [’error’, ’normal’]. This analysis will give us insights into using a pre-trained model (Williams et al. (2018)) to classify sections of the log entries. We’ll experiment with different sentences used in the log entries to get an insight into this "zero shot" classification. The zero-shot classification works under the premise of a sequence and a hypothesis, results in probabilities of "Contradiction", "Entailment" or "Neutral". The table below has examples of how it works.

Premise	**Label**	**Hypothesis**
Memory in location 0x224568 has overflow and a sigfault has been generated, shutting down	Contradiction	The system is healthy
user xyz has logged in successfully from IP 2.2.2.4 and port 23	Neutral	Multiple users have registered for the system
module unit phy has malfunctioned and is restarting	Entailment	Some entity has crashed

Table 1: Zero shot method

  The Python source file "zeroshot_log.py" uses "facebook/bart-large-mnli" to conduct a hypothesis testing of "This example is an error" and the "sequence" as one line from the file. We will detail some of the results that we obtained on different files and their precision, recall and F1 scores.

- **Approach 2** RAG based vector DB and pre-trained LLM
  The next approach is detailed in Figure1 and Figure 2. For our experiments, we use "Faiss" Huggingface Vector db format to build our vector database. We split the log file into manageable chunks of 100 lines and prepare it as an input for the next stage. The python source code "src/createtxtdb.py" uses the "HuggingFaceEmbeddings" module to create the Faiss database disk file.

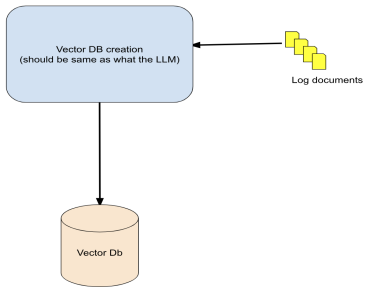

Figure 1: Vector db

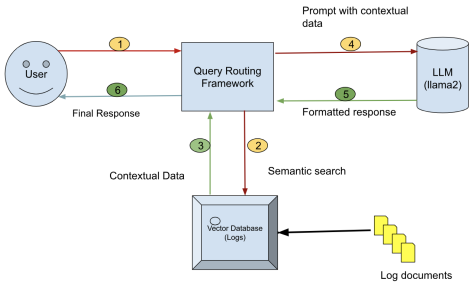

Figure 2: RAG with llama2 LLM

5 Experiments

5.1 Data

- **BGL** Oliner and Stearley (2007)
  BGL is an open dataset of logs collected from a BlueGene/L supercomputer system at Lawrence Livermore National Labs (LLNL) in Livermore, California. The non-alerts are marked with "-" at the beginning of the log. This data will used in our study for experiments. A snapshot of the data :

```
KERNMNTF 1136322675 2006.01.03 R05-M0-N8-I:J18-U01 2006-01-03-13.11.15.856535 R05-M0-N8-I:J18-U01 RAS KERNEL FATAL Lustre mount FAILED : bglio90 : block_id : location
KERNMNTF 1136322676 2006.01.03 R01-M0-NC-I:J18-U01 2006-01-03-13.11.16.068209 R01-M0-NC-I:J18-U01 RAS KERNEL FATAL Lustre mount FAILED : bglio28 : block_id : location
KERNMNTF 1136322676 2006.01.03 R04-M1-N4-I:J18-U11 2006-01-03-13.11.16.422780 R04-M1-N4-I:J18-U11 RAS KERNEL FATAL Lustre mount FAILED : bglio71 : block_id : location
- 1136326357 2006.01.03 R10-M0-N7-C:J17-U01 2006-01-03-14.12.37.696447 R10-M0-N7-C:J17-U01 RAS KERNEL INFO 1 ddr error(s) detected and corrected on rank 0, symbol 14 ove
- 1136326357 2006.01.03 R10-M0-N7-C:J17-U01 2006-01-03-14.12.37.823416 R10-M0-N7-C:J17-U01 RAS KERNEL INFO total of 1 ddr error(s) detected and corrected over 2687 secon
- 1136326357 2006.01.03 R06-M1-N6-C:J15-U01 2006-01-03-14.12.37.944165 R06-M1-N6-C:J15-U01 RAS KERNEL INFO 2 torus receiver x+ input pipe error(s) (dcr 0x02ec) detected
- 1136326358 2006.01.03 R06-M1-N6-C:J15-U01 2006-01-03-14.12.38.050765 R06-M1-N6-C:J15-U01 RAS KERNEL INFO 3 tree receiver 2 in re-synch state event(s) (dcr 0x019a) dete
```

Figure 3: BGL log

- **Synthetic microservices log**
  A simulated log with failure pattern injected at random times. We wrote a Python code to simulate three micro-services that continuously print logs to their output files. Error events were injected with 10% probability. Each event comprises 3-10 such error logs that are printed in the log file. Statistically, the entire log file will have around 90% normal logs and 10% error(or anomalous) logs Data snapshot:

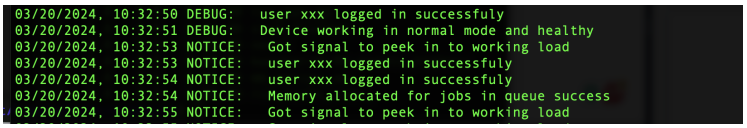

Figure 4: Our synthetic log

- **SSHLog:** OpenSSH log $github$: $Zhuet\ al.$ (2023) Data snapshot:

Figure 5: SSH log

5.2 Evaluation method

As the problem datasets are binary, we used the following definitions of Precision, Recall and F1 score.

Precision = $\frac{TP}{TP+FP}$
Recall = $\frac{TP}{TP+FN}$
F1 = 2 x $\frac{Precision x Recall}{Precision+Recall}$

The RAG and LLM evaluation will be qualitative by examining a few results by hand for this research. More extensive evaluation will be discussed in the "future work" of the conclusion section.

5.3 Experimental details

- **Synthetic log creation with anomaly event**
  The data section details the experiments that we did to create relevant synthetic logs. The difficulty in creating such simulated logs was the randomness and relevance of an anomaly event that we triggered. We used a fixed set of log template lines to generate almost similar lines, but with a new timestamp. The generation logic turned out to be complicated and will be investigated in future.
- **Experiment NLI zero shot classification**
  As described earlier, "zeroshot_log.py" was used to conduct the hypothesis testing of "This example is an error" and the "sequence" as one line from the file.

Lines to classify	Time	compute
11k	162 mins	24 core cpu
1k	13 mins	24 core cpu

Table 2: NLI classification time

- **RAG Vector db creation and RAG with Llama2_7b**
  For the first proof of concept we created a very small corpus of seven text documents and created a small "Faiss" vector db. We then devised a very simple query to try to access logical deduction from the new set of documents. The answer given was as expected and can be rated as "good" if we're to rate the qualitative aspect of it.

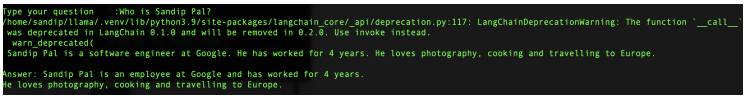

Figure 6: RAG LLM initial exp

5.4 Results

Report the quantitative results that you have found. Use a table or plot to compare results and compare against baselines.

- Approach 1: NLI zero shot log classification as "error" or "normal" for two synthetic logs

	Precision	Recall	F1
microserv NLI (ours)	0.949	0.969	0.958
RAGLog (Pan et al. (2023))	0.91	0.88	0.89
LogPrompt(Liu et al. (2024))	0.25	0.83	0.38

Table 3: NLI zeroshot and comparison with RAGLog and LogPrompt

The Precision, Recall and F1 scores are based on our simulated log data , and the comparative scores are based on the corresponding research papers.

- Approach 2: RAG and LLM (llama2)
  Our investigation with RAG and LLM was exploratory and we sampled a few answers from the combined vector database and LLM.
  With the synthetic microservices log file as an input for the vector db , and the question "When did the wrong memory location corruption happen? How many times?"
  The answer was satisfactory as it pointed to one correct date and log, and was not able to give the count. yes, it did hallucinate, but that is how the inherent transformer and attention mechanism works.

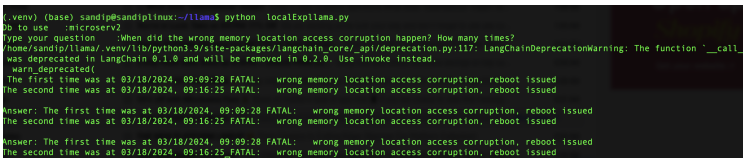

Figure 7: RAG QA

6 Analysis

With the advent of LLMs the models have successfully mapped the relationship between words used in common English language conversations and can answer questions using the underlying transformer architecture. Log analysis using RAG can be further fine-tuned to identify the specific log conversations of a particular system.

As we study diverse log systems, the intuition behind log entries and their association with natural language becomes evident. Logs are written by developers/engineers who speak a mix of technical and natural language conversation. It may happen the logs are written by multiple diverse developers, yet the overall system can be thought to be independent conversations about individual health, happiness, warning, despair and finally fatality.
The zero-shot method is very effective in identifying a "cluster" of "anomalous" logs given an effective "anomalous" label is used in the hypothesis. It may need a few iterations to test the hypothesis that works best to identify most of the anomalous log lines or clusters of log lines.
Log analysis can have real time needs and LLM with RAG may turn out to be a bottleneck. Our

experiments with "llama2" 7 billion parameters 8 bit LLM had a high latency (5-10 secs) when executed on a twenty-four core CPU. The 7 billion parameter LLM model was very large in terms of loading it to a GPU memory less than 24Gigabytes.

7 Conclusion

Simple models like NLIYin et al. (2019) which are trained on far less corpus can serve as classifiers for unseen text and classes. On the other hand, LLM models trained with a generic huge corpus of text provide the base for word association. This helps with transfer learning techniques like RAG that save cost and time in terms of computations when a model is trained from scratch.

Log analyzers can be more precise if the underlying log message schema is agreed on and the same is used by the analyzer. But most of the systems are designed without this fixed template, and thus there is a need to have a generic solution. As the nature of logging varies vastly from system to system, a globally general fixed model/algorithm to detect failures for any kind of log is very hard to design.

Log analyzers need to learn from the local system that it is trying to analyze and the inference model built for such a system may not be fairing well for another system. The primary limitation of our study was getting enough diverse logs that could be used to train a system to find anomalies. There needs to further experiments to decipher relationships between multiple components of the system to trace the original root cause of that failure.

References

Shilin He, Jieming Zhu, Pinjia He, and Michael R. Lyu. 2016. Experience report: System log analysis for anomaly detection. In *2016 IEEE 27th International Symposium on Software Reliability Engineering (ISSRE)*, pages 207–218.

Yilun Liu, Shimin Tao, Weibin Meng, Jingyu Wang, Wenbing Ma, Yanqing Zhao, Yuhang Chen, Hao Yang, Yanfei Jiang, and Xun Chen. 2024. Interpretable online log analysis using large language models with prompt strategies.

A. Oliner and J. Stearley. 2007. What supercomputers say: A study of five system logs. In *LLM-Parser: An Exploratory Study on Using Large Language*, page 575–584, 37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN'07). IEEE.

Jonathan Pan, Swee Liang Wong, and Yidi Yuan. 2023. Raglog: Log anomaly detection using retrieval augmented generation. In *RAGLog: Log Anomaly Detection using Retrieval Augmented Generation*, Online.

Adina Williams, Nikita Nangia, and Samuel Bowman. 2018. A broad-coverage challenge corpus for sentence understanding through inference. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 1112–1122. Association for Computational Linguistics.

Wenpeng Yin, Jamaal Hay, and Dan Roth. 2019. Benchmarking Zero-shot Text Classification: Datasets, Evaluation, and Entailment Approach. In *Proc. of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*.

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

A Appendix

-