# Diagnosing and Resolving Cloud Platform Instability with Multi-modal RAG LLMs

Yifan Wang
wangyifan@cs.cornell.edu
Computer Science Department, Cornell University
Ithaca, NY, USA

Kenneth P. Birman
ken@cs.cornell.edu
Computer Science Department, Cornell University
Ithaca, NY, USA

## Abstract

Today's cloud-hosted applications and services are complex systems and a performance or functional instability can have dozens or hundreds of potential root-causes. Our hypothesis is that by combining the pattern matching capabilities of modern AI tools with a natural multimodal RAG LLM interface problem identification and resolution can be simplified. ARCA is a new multimodal RAG LLM system targeting this domain. Step-wise evaluations also show that ARCA performs better than state of the art.

***CCS Concepts:*** • **Software and its engineering → System administration**; • **Information systems** → *Information retrieval*; • **Computing methodologies → Knowledge representation and reasoning**.

***Keywords:*** Root cause analysis, RAG LLM, AI-Ops

**ACM Reference Format:**
Yifan Wang and Kenneth P. Birman. 2025. Diagnosing and Resolving Cloud Platform Instability with Multi-modal RAG LLMs. In *Proceedings of The 5th Workshop on Machine Learning and Systems (EuroMLSys) (EuroMLSys '25).* ACM, New York, NY, USA, 8 pages. https://doi.org/XXXXXXX.XXXXXXX

## 1 Introduction

Incident response in complex systems entails 4 steps. (1) *Detection,* which includes the detection or prediction of an impending problem; (2) *Triage:* categorizing severity and assigning the task to a Site Reliability Engineering (SRE) team; (3) *Diagnosis:* collecting more data and pinpointing the root cause; (4) *Mitigation:* Formulating and carrying out a response and disabling any extra instrumentation that was activated.


*EuroMLSys '25, Rotterdam, The Netherlands*

ACM ISBN 978-1-4503-XXXX-X/2018/06
https://doi.org/XXXXXXX.XXXXXXX

Decades of work has given us a remarkable range of AI-assisted IT-Operation (AI-Ops) tools covering each step, like prediction-based anomaly alarming, classification-based internal support ticket assigning tool for triaging, root-cause analysis tools using language models for summarization and many more. These AI tools work on different modes of data, including user-provided bug reports in natural language, system logs in a semi-structured language and numerical performance metrics.

Our work takes the next step by offering an AI-Ops solution that can carry out cross-modality reasoning. Creating a multi-modal AI-Ops solution is complicated by a lack of high-quality training data sets. Existing public data sets in the domain of AI-Ops and IT-Ops generally reflect a single data mode, as in HPC4 [13], COM2 [16], etc.

Even for a single data mode, existing AI-Ops solutions struggle to adapt to changes in their operating environment. If the underlying data distribution shifts, for example after a hardware upgrade, the performance of threshold-based incident detection tool may degrade. Even small modifications in the log formatting can defeat log analytics implemented with regular expressions, forcing costly code changes and even model retraining.

Finally, AI-Ops tools are often proprietary and forbiddingly expensive. DevOps teams at cloud computing companies with vast GPU deployments can train new models, but this is out of the question for smaller companies.

ARCA is an *AI for Root Cause Analysis* based on a multimodal RAG (Retrieval-Augmented Generation) approach, in which an LLM is augmented by a database. Many RAG systems are limited to approximate search in document or image collections, but ARCA also supports data in structured (tabular) collections and logs. The basic idea is to focus on recurrent incidents, looking for similar past problems, summarizing prior findings, and recommending mitigation strategies that succeeded in the past.

A complicating factor is that users often report incidents in fuzzy ways, which limits label quality: a particular problem given that many AI-Ops tools are trained using labeled data. Rather than battling this reality, our work focuses on *approximate match*, but generalizes this to encompass data modalities other than text. The idea, though, is similar: RAG LLMs for search document collections treat each query as a vector database search for documents "similar" to the query. ARCA treats the multimodal signature of the incident as

a kind of query and performs approximate match against precomputed signatures from past incidents, thereby automating the search task.

Here we report on a proof-of-concept that supports three data modes: (1) *incident descriptions*, in natural language; (2) *logs* of semi-structured text generated by automated reporting components; and (3) multivaiate performance-counter time-series. ARCA is an end-to-end tool created from off-the-shelf ML models, and designed to cover incident response steps from triaging new cloud incidents to generating mitigation plans for the SREs. The ARCA multimodal RAG search mechanism (Sec. 3) is an original contribution of our effort. The future ARCA will expand these data modes and enlarge ARCA's multimodal pattern-matching capabilities.

To test the end-to-end effectiveness of ARCA, we created a data set of 800 bug reports collected from micro service systems in a controlled environment. The bug reports are typical Bugzilla incident reports of the kind users employ to request issue resolution. Each contains three components: 1) the user's incident description; 2) a log file collected from the docker container of the faulty service and 3) a time sequence of performance metrics collected from the same container during the the fault. Although the bugs have very different features, all trace to root causes associated with three widely recognized cloud computing issues: computations that exceeded time limits, memory leaks and network delays. In the evaluation, ARCA achieves 92% accuracy in triage and 72% accuracy in finding the correct mitigation plan. We have also tested the efficacy of individual components in ARCA using established data sets.

## 2 Related Work

Before we dive into the details of ARCA, we review related work that shaped our thinking.

### 2.1 Retrieval Augmented Generation

The RAG paradigm is in widespread use [5, 11]. In this approach, a query is first transformed into a vector representation and an approximate nearest neighbor search is then used to fetch relevant documents from a knowledge base. The retrieved content is then provided as auxiliary input to a generative model, typically an LLM. This extra "context" allows the model to ground its outputs in factual, up-to-date, or domain-specific information, reducing hallucinations and offering a way to continuously update the knowledge base without retraining models. RAG is effective for question answering [9], summarization [10], and code generation [14], and has been shown to significantly improve LLM performance and interpretability. Prior work on multimodal RAG has focused on the visual domain (text used to describe images). In ARCA, however, we need a RAG system specially for IT-Ops/AI-Ops. To the best of our knowledge, our work is the first to explore this form of multimodality.

### 2.2 Prompting and Reasoning

Prompt engineering is central to RAG LLM design. One prompting technique, few-shot learning [1], leverages the in-context learning capabilities of LLMs, guiding models from structure and examples in the prompt (without updates to model weights). A second, Chain of Thought [19], takes a further step by explicitly encouraging step-by-step reasoning. This has been shown to improve LLM performance on tasks requiring multi-step logical inference, arithmetic, or complex decision-making. In combination these two techniques achieve state-of-the-art performance across various domains including mathematics, common-sense reasoning, and question answering. We adopt both in ARCA.

### 2.3 AI-Ops

ARCA is also inspired by prior work in AI-Ops [2], notably for processing logs and telemetric data. LogCluster [12] introduced techniques for clustering log records to assist in bug detection using a weighted encoding, and subsequent work used LLMs to summarize abnormalities in logs [18, 22]. We drew on labelled log records from one of these efforts, LogHub[21], for our evaluation.

We noted our interested in combining application instrumentation with text records from logs. Prior studies have explored aspects of this question, notably by using DNNs for anomaly detection in multi-variant time-series data. For example, Microsoft has proposed an anomaly detector based on Convolutional Neural Network (CNN) [15], while Alibaba describes an encoder-decoder architecture in RobustTAD [4] and Tencent used a VAE network [7] for the same purpose.

The primary limitation with existing AI-Ops tools (including as the ones we cited) is that they have generally been limited to a single data mode. To the best of our knowledge, ARCA is the first multi-modal end-to-end AI-Ops solution.

## 3 How does ARCA work?

ARCA runs in two phases (Fig. 1): building the multimodal knowledge base of historical bugs and then querying it. Below we focus on a bug tracking use-case but the idea generalizes to other incident-analysis scenarios.

### 3.1 Building Phase

To deploy ARCA, we first collect and process data from existing solved bugs retrieved from bug tacking tools and then use the collected data to form a knowledge base. After creating the knowledge base, users can query ARCA for new and ongoing incidents, and the system will automatically generate a mitigation plan for each SRE.

**3.1.1 Data Sources.** We assume that software incidents are reported through tickets in a bug-tracking system such as Bugzilla. Each bug ticket contains multiple data modalities, e.g., bug descriptions (natural language), performance metrics (time sequences of numerical multi-variant data),

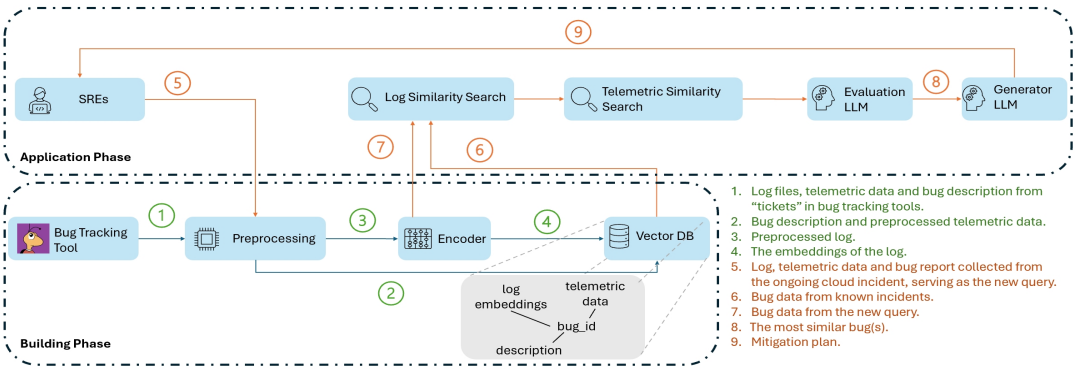


**Figure 1.** ARCA workflow in its building and query phases.

logs (semi-structured machine-generated event reports), etc. In ARCA, we strive to find a mitigation plan by reasoning across the different modes of data.

A bug tracking system works like an online bulletin board similar to Reddit. Progress towards resolving bugs is tracked as follow-up posts to the original post initiated by the staff member who found the incident. To collect data to form a knowledge base, we keep our attention to the following steps within the life cycle of a bug ticket: (1) The first post, which includes a textual description of the problem. (2) The ticket assignment post, which reflects the judgment of a human triage specialists and has a fixed format. (3) Data collection posts with attachments: these are often data collected by the SRE team using tools they found relevant and is the step at which ARCA can learn from data modalities other than natural language. (4) The last post: the last post of a closed ticket is usually the diagnosis of the issue and the following mitigation. Notice that each category of posts and data hints at a its own similarity metric: rather than a single metric for all types of data, we need a unified metric spanning multiple modalities and robust against missing data (some reports may cite data that other related reports omit).

**3.1.2 Build A Multi-modal Knowledge Base.** With the logs, performance metrics and bug descriptions retrieved from the prior step, we can build a knowledge base that associates related information. The idea behind the knowledge base is to do up front work so that later, we can quickly find similar bugs by comparing their logs and telemetric data during a triage step and then rapidly retrieve the corresponding bug descriptions to help create a mitigation plan.

To enable a fast similarity search among logs instead of directly searching the text space of the logs, ARCA embeds processed log snippets to a high-dimensional latent space, which we will refer to as the embedding space. The system will later use cosine similarity to quantify the difference between two log snippets. Calculating cosine similarity only involves calculating the product of two matrices, which can be carried out at very high speeds, particularly with the help of a GPU. The task is much quicker than searching in the text space. To further accelerate the similarity search on very large data sets, ARCA uses approximate K-nearest neighbors to organize the log embeddings in two tiers. To find the most similar log embeddings, we first look for the closest centroids. The assumption is that these event clusters will contain the embeddings most relevant to the incident report. We then use cosine similarity again, but now include performance metrics in our approximate similarity test. To enable this we first convert the performance metrics to a vector during our log preprocessing step by aligning the telemetric data of variant lengths and sources. Then, we store the vector in the knowledge base and via the bug id, can we associate it with other pieces of information collected from the same bug.

ARCA keeps bug descriptions in natural language because they may contain important details that stood out to the human observer of the issue and hence are likely to be of high value to the tasks performed out by the Evaluation LLM in later steps. Additionally, bug resolution descriptions contain mitigation plans which proved effective in the past, and the Generator LLM can use those to propose a new plan to mitigate the ongoing issues.

In ARCA, the log embedding space contains 3072-dimensional vectors of 32-bit floating point numbers, and we embed the logs using the "text-embedding-3-large" model from OpenAI. The preprocessed performance metrics are represented as 21-dimensional vectors of 32-bit floating point numbers. The knowledge base in ARCA is made of 3 object stores, with one for each of the log embedding, vectorized performance metrics and bug descriptions. We also maintain a mapping relationship between them. In the future we plan to allow dynamic additions to the database, but the PoC works with a static data set.

**3.1.3 Process Log Files.** ARCA supports two data modalities: logs of semi-structured text and bug descriptions containing human inputs in natural language. We preprocess the logs prior to improve the accuracy of ARCA's triage technique.

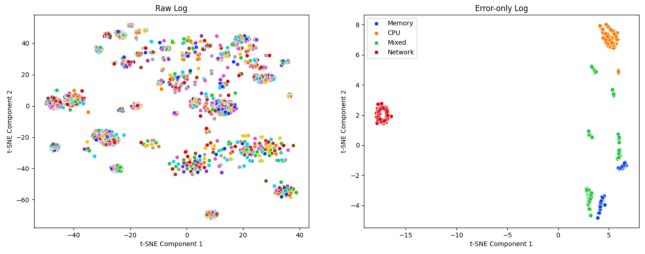


**Figure 2.** t-SNE of the embedded log content.

The logs we consider are created by a variety of applications and systems services, and take the form of text files in which system maintenance messages, warnings and errors, anomaly notations, and other reporting can be intermixed. After examining the log files in our evaluation data set, we have found that

1. A relatively small subset of log lines are relevant to any given incident.
2. Log records of a given type are formatted in similar ways. For example, heartbeat messages for the same component only differ in their timestamps.
3. For any single incident, a log may contain multiple relevant data modalities: text, tabular data, time-series data, etc.

ARCA filters log contents by retrieving the bugs that show a "similar pattern" in logs in the query step. To keep the LLM focused on important features, it is important that the log contents visible to the LLM be relevant to the issues flagged in the problem report, and free of irrelevant information if that information might be of value for maintaining the system or other purposes. Accordingly, we run a Feature Extraction LLM that we configure to remove repetitious content and extract data that distinguishes each record from the others occuring at the same time, like the error messages, special events, performance metric readings, etc. We additionally convert all the data modalities that we encounter to text. Length considerations precluded reproducing the prompt here, but we do include it as Appendix A, Fig. 6.

To assess the efficacy of this approach to log preparation, we processed 800 log files from our data set using OpenAI's gpt-4o as the Feature Extraction LLM. First, we generated embeddings from the raw log content with no preprocessing. Then, we preprocessed the logs and embedded only the filtered and aggregated outputs of the Feature Extraction LLM. We used t-distributed Stochastic Neighbor Embedding (t-SNE) [8] to project the high dimensional embedding space to a 2-D image while maintaining relative Euclidian distances. Doing so yielded the images seen in Fig. 2, where each dot represents an embedding. As we can see, the embedding of processed logs (the right picture) resolves more clearly, showing a cleaner clustering pattern with far fewer clusters than for the raw log (the left picture): evidence that this step achieved its goals. We additionally colored the dots in both images to signify root cause labels. As we can easily see, the dots from the same root cause, i.e., memory, CPU and network, are correctly clustered after preprocessing but were jumbled before doing so. Especially interesting are the green dots, for incidents in which a mix of CPU and memory issues simultaneously caused degraded system performance. These green data points are correctly located between the clusters for CPU issues and those for memory issues.

#### 3.1.4 Align Telemetric Data.

To enable a similarity search, it is necessary to convert telemetry data to a fixed length vector. Raw data can be highly platform-specific: a matrix with one row per time stamp and a column for each performance counter (CPU utilization, memory utilization, etc), but potentially with missing data due to faults and timeouts, idiosyncratic formats and units, and including hardware-specific metrics. To overcome these issues, ARCA uses a set of 7 docker performance counters, all of which are commonly available when diagnosing cloud microservice incidents. These track CPU and memory utilization, network I/O, block device I/Os, average operation latency, and socket errors. Servers are highly heterogeneous, hence raw values are not directly comparable. Accordingly, we calculate the normalized first order gradient, the average value and the standard deviation for each time series. In this way, we can convert the matrix of performance counter readings to a vector of 21 floating point numbers.

### 3.2 ARCO-PoC Phases

ARCO-PoC runs in two sub-phases: the query phase and the generating phase. In the query phase, we interrogate the populated knowledge base by carrying out the similarity search on log embeddings and vectorized performance metrics. This phase is analogous to the triage step and the output are the textual descriptions of similar bugs. The descriptions are then sent to the generating phase to create a mitigation plan for the SREs.

#### 3.2.1 Query Phase.

Once our knowledge base has been populated, ARCA performs an approximate match query using posts associated with a new incident as its query prompts. The methodology used to extract the relevant aspects of the incident is quite similar to the one used to build the knowledge base, and yields an embedding vector that we can understand as an abstract representation of the new incident in the knowledge space. Our goal is to perform an approximate nearest neighbor (ANN) search. We do this in two steps: first, we identify cluster centroids closest to the query embedding, and then within those clusters perform a search for known prior incidents with similar characteristics. Here, ARCA departs slightly from common RAG approaches that only retrieve the top tens of documents based on the similarity score. Instead, ARCA retrieves the top hundreds of bugs as reported from the similarity search. This is because ARCA

treats similarity search as a triage step for the purpose of coarsely categorizing a bug by placing it within a family of issues so that corresponding SREs can chime in. For example, if a bug seems to be CPU-related, it could be assigned to SREs working on performance issues, ones working on scheduling, and ones investigating disruptions associated with locking. With just a small number of approximate matches we might miss some relevant categories, but with hundreds of approximate machines, we have a high likelihood of routing the issue to all SREs that might have insight into the issue.

From the bugs with similar log patterns, we additionally perform a second-round KNN search in the high-dimensional space of the vectorized performance metrics. Here, an issue of cost arises: our work uses OpenAI language-generation APIs that are billed on a per-use basis. Accordingly, we only use one tenth of our prior report candidates for generation of the bug explanation hypotheses that the developer will be shown. In the evaluation, we will show that this step of filtering will not hurt the overall accuracy.

In ARCA, we use FAISS library [17] to carry out the similarity search so that it will run on GPU accelerators. We have tried to retrieve from top 100 bugs to top 500 bugs and we can reach a triage success rate as high as 92%. We will discuss the effect brought by different number of retrieved bugs in evaluation section.

**3.2.2 Generating Phase.** In the generating phase, we first use an Evaluation LLM to find the bug whose description most closely fits each incident. We pass the description of the bug fix (which contains the mitigation plan) to the Generator LLM, which in turn produces text explaining the choice and suggesting a new mitigation plan to the SREs. The approach is similar to a concept sometimes referred to as *LLM-as-a-judge* [20] (the corresponding prompt details are included in Appendix A, Fig. 7). To improve accuracy, we employ a Chain of Thought prompting style (Appendix A, Fig. 8), using a series of similar CoT prompts in accordance with standard practice in few-shot learning. The output of this step is the closest resolved bug. We then prompt the Generator LLM with the input shown in Fig. 9. The generated mitigation plans are sent back to the SREs. ARCA uses gpt-4o as its Evaluating and Generator LLMs.

## 4 Experimental Results

To evaluate our work, we first build a data set for 800 bug tickets containing descriptions, logs and performance metrics. Then we build ARCA knowledge base using 700 bug tickets and evaluate the PoC with the held-out 100 tickets.

### 4.1 Data Set

Our data set of bugs arising in micro service systems is typical of modern cloud infrastructures. To keep our data set as general as possible, we keep our attention only to the bug features reported from the docker container, including the

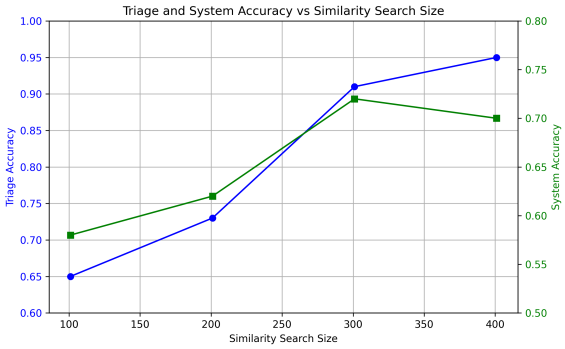


**Figure 3.** ARCA accuracy.

docker logs and the performance metric readings from the "top" command, without any application-level features.

We use a micro-service workload generator, "DeathStar" [3] to run different micro-service applications, like "Hotel-Reservation", "SocialNetwork", etc. As the application executes, we inject errors. To load the CPU, we modified the benchmark so before processing a new request, the application performs a CPU-intensive operation. We also increase the number of requests per second during runtime until the application crashes from overload. To simulate a memory leak we modify the benchmark by introducing a memory allocation in the call back function but intentionally not freeing the memory. Finally, to increase network delays, we introduce a random sleep in the call back function. To make the challenge harder, we have introduced a fourth category of error that causes both a memory leak and a long-running computation, resulting in two possible crash types.

Each of the four categories of injected errors are used to create 100 experiments, which we diversify by tweaking settings. We run each experiment twice so that we can use the data set we can automatically label an experiment run with its closest bug, which is the run generated from the same experiment configuration, yielding 4*100*2=800 bug incidents. For each bug, we use gpt-4o to generate a human readable bug report. In the generating prompt, we have provided the root causes like "the issue is caused by a random delay in every invocation of the call back function XXX" to ensure that the bug report contains meaningful mitigation plans. We have also instructed the LLM to decribe the bug by summarizing the performance metric readings and the logs. We thus obtain 800 bug tickets that contain the bug descriptions with mitigation plans, the time-series of performance metrics and the logs.

To evaluate the efficacy of our similarity search in the log embedding space, which is the key of the RAG system, we use public data sets from four supercomputing systems: BGL, Thunderbird, Liberty, and Spirit [13].

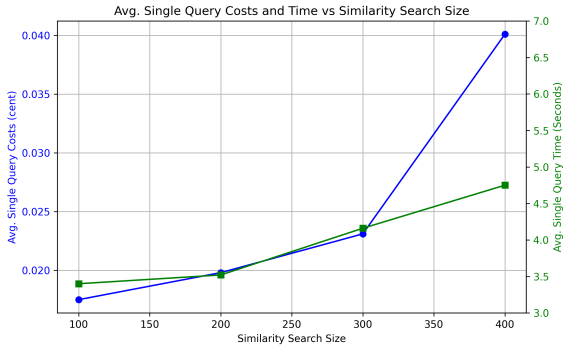


**Figure 4.** ARCA cost.

### 4.2 End-to-end Evaluation

We first study the effect of using different numbers of nearest neighbors reported from the similarity search module. This is also the size of the output from the triage step. So we compare both the triage accuracy and the system accuracy. For a triage operation to be accurate, ARCA needs to include the closest bug in the output of the triage steps. For the whole system to be accurate, ARCA needs to pick the labeled closest bug as the output of the Evaluation LLM. The results are shown in Fig. 3. To account for the randomness introduced by the LLMs, we repeated each test 300 times and report the average values in Fig. 3 and 4

In our test, we increase the log similarity search output size from 100 to 400, and we filter out 20% of the chosen bugs in the similarity search using telemetric data. As we can see, triage accuracy increases steadily with the raise of the triage set size. However, the overall system accuracy drops when we increase the similarity search size from 300 to 400. Upon inspection we found that when the similarity search size is small (less than 200), the right answer is not presented in the input prompt. This ceases to be an issue with larger set sizes. Interestingly, however, although triage accuracy at set size 400 is significantly higher than that for size 300, overall system accuracy drops: the Evaluation LLM apparently becomes overwhelmed by choices.

It's also worth noting that we cannot increase the similarity search output size without limit. GPT-4o, the LLM we use for our Evaluation LLM, has a context window size limit of 30,000 tokens and the input cannot be longer than that. This token window limit corresponds to a triage output set size of slightly more than 400.

We also evaluated time and financial cost per query. Gpt-4o uses a decoder-only neural network structure and hence the longer the input in tokens, the more time it will take to generate an answer. Also, OpenAI charges clients on the basis of the number of tokens computed. Taking all these considerations together, we arrive at the results shown in Fig. 4. As we can see, for large group size, generation is significantly slower and cost mounts substantially.

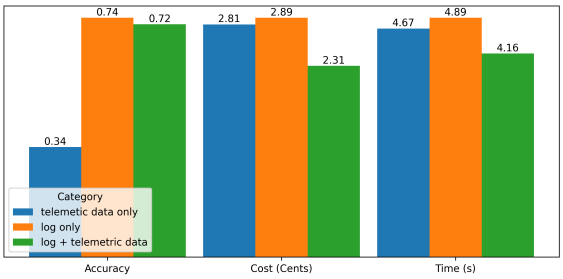


**Figure 5.** Evaluation of each similarity search step.

### 4.3 Evaluation of Similarity Search

There are two similarity search steps in ARCA, with one in the high-dimensional space of log embedding and the other in telemetric encoding. We have evaluated the efficacy of each step against the combined performance, and reported the results in 5, where we pick the triage size to be 300. The key finding is that multi-modal similarity search can save time and financial cost, while maintaining nearly the same efficacy.

### 4.4 ARCA as A Log Clustering Tool

Very similar to log clustering tools, ARCA's RAG-LLM based log processing module can be used alone to detect anomalies in logs. In our evaluation, we use public log data sets reported from 4 supercomputing labs and report the results in 1. The numbers before '/' are from ARCA and the one after are the state-of-the-art numbers reported in [6, 18], which are achieved through proprietarily fine-tuned LLMs. From the results, ARCA-PoC outperforms on all data sets despite requiring only off-the-shelf embedding LLMs.

| Data Set | F1-Score | Recall | Precision |
|---|---|---|---|
| BGL | **0.995**/0.976 | **0.99**/0.982 | **1**/0.970 |
| Thunderbird | **0.984**/0.97 | 0.975/**0.99** | **1**/0.97 |
| Spirit | **0.993**/0.992 | 0.986/**0.999** | **1**/0.984 |
| Liberty | 0.986/* | 0.986/* | 0.986/* |

**Table 1.** Evaluation as a log clustering tool.

## 5 Conclusions and Future Work

ARCA is a work in progress, but already reveals promise for a new multimodal RAG LLM approach to searching incident report databases created when troubleshooting complex cloud-hosted applications. In the future, we plan to support more data modalities into ARCA, with a focus on performance metrics and traces. Synthesis of generated answers that incorporate observations from multiple modalities raises especially interesting questions for study.

## References


[1] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, and et al. 2020. Language Models are Few-Shot Learners. arXiv:2005.14165 [cs.CL] https://arxiv.org/abs/2005.14165

[2] Qian Cheng, Doyen Sahoo, Amrita Saha, Wenzhuo Yang, Chenghao Liu, Gerald Woo, Manpreet Singh, Silvio Saverese, and Steven C. H. Hoi. 2023. AI for IT Operations (AIOps) on Cloud Platforms: Reviews, Opportunities and Challenges. arXiv:2304.04661 [cs.LG] https://arxiv.org/abs/2304.04661

[3] Yu Gan, Yanqi Zhang, Dailun Cheng, Ankitha Shetty, Priyal Rathi, Christina Delimitrou, and et al. 2019. An Open-Source Benchmark Suite for Microservices and Their Hardware-Software Implications for Cloud & Edge Systems. In *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems* (Providence, RI, USA) *(ASPLOS '19)*. Association for Computing Machinery, New York, NY, USA, 3–18. doi:10.1145/3297858.3304013

[4] Jingkun Gao, Xiaomin Song, Qingsong Wen, Pichao Wang, Liang Sun, and Huan Xu. 2021. RobustTAD: Robust Time Series Anomaly Detection via Decomposition and Convolutional Neural Networks. arXiv:2002.09545 [cs.LG] https://arxiv.org/abs/2002.09545

[5] Yunfan Gao, Yun Xiong, Xinyu Gao, Kangxiang Jia, Jinliu Pan, Yuxi Bi, and et al. 2024. Retrieval-Augmented Generation for Large Language Models: A Survey. arXiv:2312.10997 [cs.CL] https://arxiv.org/abs/2312.10997

[6] Hongcheng Guo, Jian Yang, Jiaheng Liu, Jiaqi Bai, Boyang Wang, Zhoujun Li, Tieqiao Zheng, Bo Zhang, Junran peng, and Qi Tian. 2024. LogFormer: A Pre-train and Tuning Pipeline for Log Anomaly Detection. arXiv:2401.04749 [cs.LG] https://arxiv.org/abs/2401.04749

[7] Tao Huang, Pengfei Chen, and Ruipeng Li. 2022. A Semi-Supervised VAE Based Active Anomaly Detection Framework in Multivariate Time Series for Online Systems *(WWW '22)*. Association for Computing Machinery, New York, NY, USA, 10 pages. doi:10.1145/3485447.3511984

[8] Jeff Johnson, Matthijs Douze, and Hervé Jégou. 2017. Billion-scale similarity search with GPUs. arXiv:1702.08734 [cs.CV] https://arxiv.org/abs/1702.08734

[9] Vladimir Karpukhin, Barlas Oğuz, Sewon Min, Patrick Lewis, Ledell Wu, Wen tau Yih, and et al. 2020. Dense Passage Retrieval for Open-Domain Question Answering. arXiv:2004.04906 [cs.CL] https://arxiv.org/abs/2004.04906

[10] Philippe Laban, Alexander R. Fabbri, Caiming Xiong, and Chien-Sheng Wu. 2024. Summary of a Haystack: A Challenge to Long-Context LLMs and RAG Systems. arXiv:2407.01370 [cs.CL] https://arxiv.org/abs/2407.01370

[11] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, and et al. 2021. Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks. arXiv:2005.11401 [cs.CL] https://arxiv.org/abs/2005.11401

[12] Qingwei Lin, Hongyu Zhang, Jian-Guang Lou, Yu Zhang, and Xuewei Chen. 2016. Log Clustering Based Problem Identification for Online Service Systems. In *2016 IEEE/ACM 38th International Conference on Software Engineering Companion (ICSE-C)*. 102–111.

[13] Adam Oliner and Jon Stearley. 2007. What Supercomputers Say: A Study of Five System Logs. In *37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN'07)*. 575–584. doi:10.1109/DSN.2007.103

[14] Md R. Parvez, Wasi U. Ahmad, Saikat Chakraborty, Baishakhi Ray, and Kai-Wei Chang. 2021. Retrieval Augmented Code Generation and Summarization. arXiv:2108.11601 [cs.SE] https://arxiv.org/abs/2108.11601

[15] Hansheng Ren, Bixiong Xu, Yujing Wang, Chao Yi, Congrui Huang, Xiaoyu Kou, Tony Xing, Mao Yang, Jie Tong, and Qi Zhang. 2019. Time-Series Anomaly Detection Service at Microsoft. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (KDD '19)*. ACM, 3009–3017. doi:10.1145/3292500.3330680

[16] Bianca Schroeder and Garth A. Gibson. 2007. Disk Failures in the Real World: What Does an MTTF of 1,000,000 Hours Mean to You?. In *5th USENIX Conference on File and Storage Technologies (FAST 07)*. USENIX Association, San Jose, CA. https://www.usenix.org/conference/fast-07/disk-failures-real-world-what-does-mttf-1000000-hours-mean-you

[17] Laurens v. d. Maaten and Geoffrey Hinton. 2008. Visualizing Data using t-SNE. *Journal of Machine Learning Research* 9, 86 (2008), 2579–2605. http://jmlr.org/papers/v9/vandermaaten08a.html

[18] Yuqing Wang, Mika V. Mäntylä, Jesse Nyyssölä, Ke Ping, and Liqiang Wang. 2025. Cross-System Software Log-based Anomaly Detection Using Meta-Learning. arXiv:2412.15445 [cs.SE] https://arxiv.org/abs/2412.15445

[19] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Denny Zhou, and et al. 2023. Chain-of-Thought Prompting Elicits Reasoning in Large Language Models. arXiv:2201.11903 [cs.CL] https://arxiv.org/abs/2201.11903

[20] Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Ion Stoica, and et al. 2024. Judging LLM-as-a-judge with MT-bench and Chatbot Arena. In *Proceedings of the 37th International Conference on Neural Information Processing Systems* (New Orleans, LA, USA) *(NIPS '23)*. Curran Associates Inc., Red Hook, NY, USA, Article 2020, 29 pages.

[21] 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 *2023 IEEE 34th International Symposium on Software Reliability Engineering (ISSRE)*. 355–366. doi:10.1109/ISSRE59848.2023.00071

[22] Yichen Zhu, Weibin Meng, Ying Liu, Shenglin Zhang, Tao Han, Shimin Tao, and Dan Pei. 2021. UniLog: Deploy One Model and Specialize it for All Log Analysis Tasks. arXiv:2112.03159 [cs.NI] https://arxiv.org/abs/2112.03159

## Appendix A: LLM Prompts

Received 11 February 2025

```
{
    "role": "system",
    "content": "You are a Site Reliable
Engineering (SRE) working on processing
system logs from cloud vendors, like AWS."},
{
    "role": "user",
    "content": "Please examine the following
log content line by line and keep only the
lines that contains warnings, errors, fatal
messages, special events like user inputs
and performance counter readings. The
performance counter readings should be in
the format of "[METRIC_NAME] is x", e.g.,
cpu_utilization_rate is 90%." + LOG_CONTENT
}
```

**Figure 6.** Prompt for LLM to process log file.

```
{
    "role": "system",
    "content": "You are a Site Reliable
Engineering (SRE) working in information
technology operation group (IT-Ops) of a cloud
vendor, like AWS."},
{
    "role": "user",
    "content": "You will receive one new incident
description and many existing descriptions from
incident reports. Please let me know which one of
the existing descriptions are mostly likely to
share the same root cause as the new incident
description.
In the input, the new incident description is
labeled as [New] and the existing descriptions
are labeled as [X], where X is the index of the
description, like [1], [2], etc.
In the answer, first you need to explain your
thoughts, and then you need to give your answer
of picking the most similar incident, following
this format: "[[X]], where X is the index of the
chosen description." + [NEW_DESCRIPTION] +
[DESC_1] + [DESC_2] + ...}
```

**Figure 7.** Prompt for the Evaluation LLM.

```
{
    "role": "system",
    "content": "You are a Site Reliable
Engineering (SRE) working in information
technology operation group (IT-Ops) of a cloud
vendor, like AWS."},
{
    "role": "user",
    "content": "To determine if two bug
descriptions describe incidents originated from
the same root cause, we can look at the
similarity between the performance metrics. For
example, in the following 3 sample description
texts. A has CPU utilization rate of 90%, B has a
CPU utilization rate of 92% and C has a CPU
utilization rate of 10%. Because |92% - 90%| <
|10% - 90%|, we feel A and B have similar
performance metrics and hence they are likely to
be caused the same root cause." + [A_Desc] +
[B_Desc] + [C_Desc]
}
[other similar CoT prompts for few-shot learning]
```

**Figure 8.** The CoT contexts for the Evaluation LLM.

```
{
    "role": "system",
    "content": "You are a Site Reliable
Engineering (SRE) working in information
technology operation group (IT-Ops) of a cloud
vendor, like AWS."},
{
    "role": "user",
    "content": "You will receive an incident
report from resolved bugs in the past labeled as
[Resolved], together with the descriptions of
the description of the current incident, labeled
as [Current].
You need to read the mitigation plan from the
resolved bug report and create a new plan for
the current incident. Also, please explain your
thoughts." + [RESOLVED_BUG_DESC] + [NEW_DESC]
}
```

**Figure 9.** Prompt for the Generator LLM.