ARCA: AI for Root Cause Analysis

Yifan Wang [*1] **Max Koster** [*1] **Neeraj Parihar** [*1] **Shubhangi Thakur** [*1] **Tianbin Liu** [*2]

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 the pattern matching capabilities of modern AI tools, combined with a natural multimodal RAG LLM interface, could simplify problem identification and resolution. ARCA is a new multimodal RAG LLM system targeting this domain.

1. Motivation

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

Decades of work has given us a remarkable range of AI-assisted problem detection, data collection and diagnostic tools, including prediction-based anomaly alarming, classification-based internal support ticket assigning tool for triaging, root-cause analysis tools for diagnosing and strategy recommendations for mitigation.

The task is genuinely hard: even when data sets are purely numerical, proper interpretation will depend on the system component being monitored, the semantics of the monitoring output, correlations with logs, traces, software and hardware update records, etc.

We see this as a match with *Retrieval-Augmented Generative* (RAG) LLMs, in which an LLM is augmented with a database that could support many data modalities. We propose ARCA: an *AI for Root Cause Analysis*. ARCA focuses on recurrant incidents, and the intuition is simple: when faced with a problem, look for similar past problems, describe them, and recommend the mitigation strategies that succeeded in each case.

Following the intuition, we have built a Proof-of-Concept ARCA (ARCA-PoC), which works on two modes of data: (1) *incident descriptions*, which report the phenomenon of the issues from the bug reporters' perspectives in natural language and (2) *logs*, as generated by the faulty software components and the related parts. Our ARCA-PoC covers the incident response steps from triaging new cloud incidents to generating mitigation plans for the SREs. Once initialized, the ARCA-PoC can generate mitigation plans without human intervention.

To test the effectiveness of ARCA-PoC, we have created a data set of 800 bug reports collected from micro service systems in a controlled environment, which represent bugs from bug tracing tools like Bugzilla. In the evaluation, ARCA-PoC achieves 92% accuracy in triage and 72% accuracy in finding the correct mitigation plan. The evaluation methodology and results will be disclosed in Section 4, after which we conclude our work in Section 5.

2. Related Work

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

2.1. Retrieval Augmented Generation

Content generation using RAG LLM is a prominent paradigm that combines the strengths of information retrieval and generative modeling to enhance the accuracy and contextual relevance of text generation tasks (Lewis et al., 2021; Gao et al., 2024). In this approach, a retrieval module first fetches relevant information from a knowledge base or document corpus based on the input query, and the retrieved content is then provided as auxiliary input to a generative model, typically an LLM. This integration allows the model to ground its outputs in factual, up-to-date, or domain-specific information, effectively addressing limitations of standalone generative models such as hallucination or outdated knowledge. In ARCA, we design and implement the RAG system specially for IT-Ops/AI-Ops.

2.2. Prompting and Reasoning

Prompting has emerged as another powerful paradigm in natural language processing, enabling LLMs to perform diverse tasks with minimal training. Few-shot learning (Brown et al., 2020), a key technique in this paradigm, uses carefully designed prompts containing a small number of

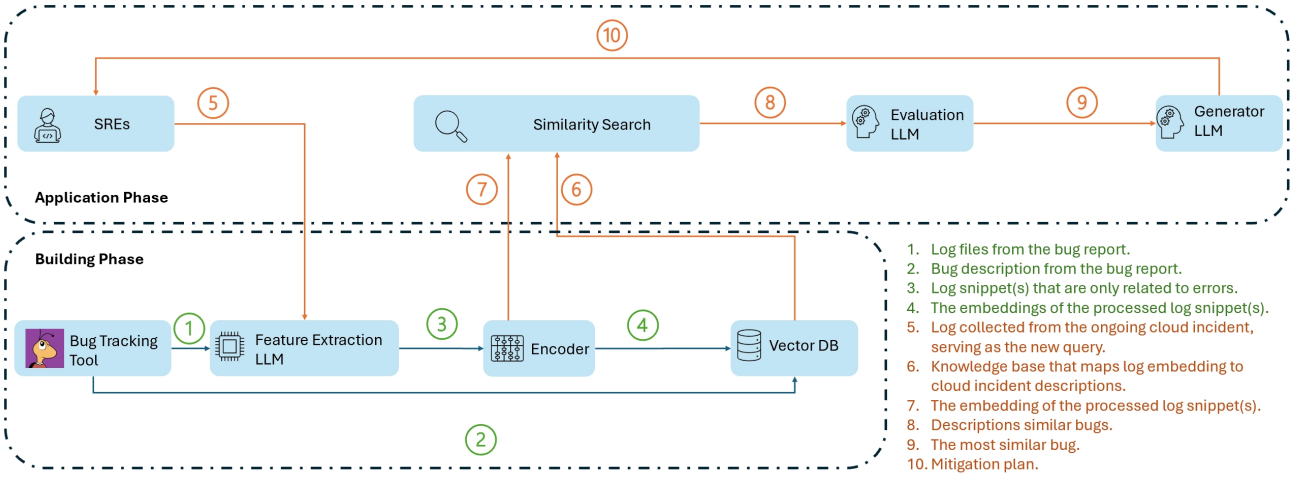


Figure 1. The workflow of ARCA in the building phase and query phase.

task-specific examples to guide the model's behavior. This approach leverages the in-context learning capabilities of LLMs, where models learn from the structure and examples in the prompt without requiring updates to their weights. Building on this foundation, Chain of Thought (CoT) (Wei et al., 2023) prompting enhances the reasoning abilities of LLMs by explicitly encouraging step-by-step reasoning in their outputs. Further advancing this, ReAct (Yao et al., 2022) and Reflexion (Shinn et al., 2023) includes user feedbacks in the loop. In ARCA, we use Few-shot learning and CoT to prompt the LLMs in our system.

2.3. AI-Ops

ARCA is also inspired by the pioneering work in AI-Ops. In the work of XPert (Jiang et al., 2023), Jiang et al. discusses using LLM code generation capability to generate database queries for IT-Ops. In UniLog (Zhu et al., 2021), Xu et al. studies how to use an LLM to parse and analyze logs. Meanwhile, Xiong et al. creates a validation tool for AI infrastructure called SuperBench (Xiong et al., 2024).

3. Method

As shown in Fig. 1, ARCA works in two phases: building the multimodal knowledge base for fixed bugs and then querying it. Below we focus on the case of data from a bug tracking system, but the idea generalizes to the other data modalities ARCA will support.

3.1. Building Phase

To build ARCA, we first collect and process data from existing solved bugs retrieved from bug tacking tools and then we use the collected data to form a knowledge base.

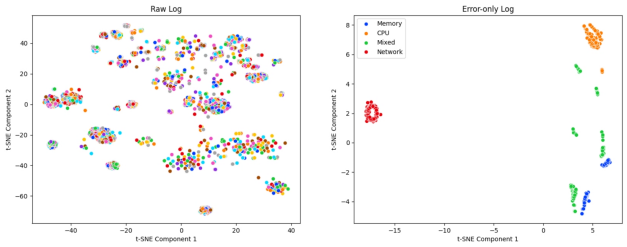


Figure 2. t-SNE of the embedded log content.

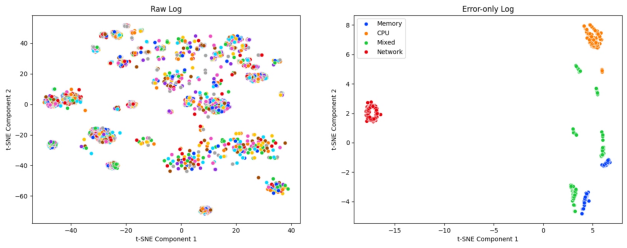


Figure 3. t-SNE of the embedded log content.

Then ARCA can query the knowledge base with new data collected from ongoing incidents. Experiments discussed in our evaluation are based solely on synthetic datasets created from the Deathstar workbench. We also constructed a dataset of real-world Linux kernel bugs via scraping from Kernel.org Bugzilla; however, we have not yet applied RAG techniques to this dataset.

3.1.1. Build Knowledge Base

To build a knowledge base that enables fast similarity search, ARCA embeds the log snippets to a high-dimensional latent space so that it can use cosine similarity to quantify the dif-

ference between two log snippets. We use GPU to calculate cosine similarity. To further expedite the similarity search on very large scale of data sets, ARCA uses approximate K-nearest neighbors to organize the log embeddings in two tiers. So to find the most similar log embeddings, we first look for the closest centroid and then we can spot our answer from the centroid groups that contain only a fraction of all the embeddings.

ARCA keeps the bug descriptions in natural language. We made this decision because bug descriptions 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.

In ARCA-PoC, the embedding model is "text-embedding-3-large" from OpenAI. The knowledge base in ARCA-PoC is made of 2 object stores: one for the log embeddings and the other for bug descriptions. We also maintain a mapping relationship between them.

3.1.2. Process Log Files

Logs are text files of system maintenance messages, warnings and errors, anomaly notations, etc. 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 very similar. For example, heartbeat messages for the same component only differ in their timestamps.
3. Even within a log, we find multiple data modalities: text, tabular data, time-series data, etc.

So we use a Feature Extraction LLM to filter the log records by keeping the warning, error and fatal messages only. The prompt is in the appendix.

To illustrate the efficacy of log preparation, we processed 800 log files from our data set. For the raw log content, we don't make any change and embed every line from the log. For the processed log content, we embed the outputs of the Feature Extraction LLM. Then we used t-distributed Stochastic Neighbor Embedding (t-SNE) (Johnson et al., 2017) to project the high dimensional embedding space to a 2-D image. Doing so yielded the images seen in Fig. 3, where each dot represents an embedding. As we can see, the embedding of processed logs (the right picture) reveals a much more clustered pattern than the raw log (the left picture), which is desirable for triage.

On performance metric sequences, We experimented with Granger Causality and the Vector Autoregression method to identify causal and correlation relationships among performance metrics. Our analysis revealed that network input, network output, and memory usage could be the potential drivers of average latency, highlighting their role as potential root causes. Further experimentation with larger datasets is required to validate and strengthen these findings.

In ARCA-PoC, we use "gpt-4o" from OpenAI as the Feature Extraction LLM.

3.2. Application Phase

The application phase can be divided into two sub-phases: the query phase and the generating phase. In the query phase, we first query the populated knowledge base by carrying out the similarity search. This phase is analogous to the triage step and the output are the textual descriptions of similar bugs. The descriptions are 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 query prompts. The idea is similar: we extract features from the logs and again form an embedding reusing the same software component as we build the knowledge base. Then we perform an approximate KNN search first for cluster centroids closest to the query embedding, and then for individual bug.

In ARCA-PoC, we use FAISS library (Johnson et al., 2017) to carry out the similarity search so that we can enjoy the acceleration brought by GPUs.

3.2.2. Generating Phase

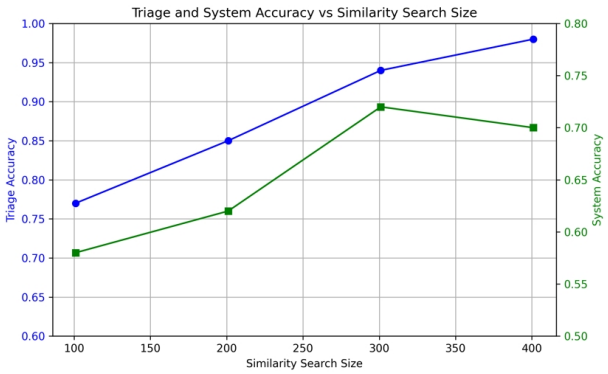
In the generating phase, we first use an Evaluation LLM to find the bug whose description shows the closest resemblance to the ongoing issue. Then we send the chosen description of the fixed bug, which contains the mitigation plan, to the Generator LLM to create a new plan for the SREs.

The Evaluation LLM is built on the tested idea of LLM-as-a-judge (Zheng et al., 2024) and we format the prompt in the same manner. To improve the accuracy, we have also introduced multiple CoT examples in the context of the Evaluating LLM. A prompt example is in the appendix.

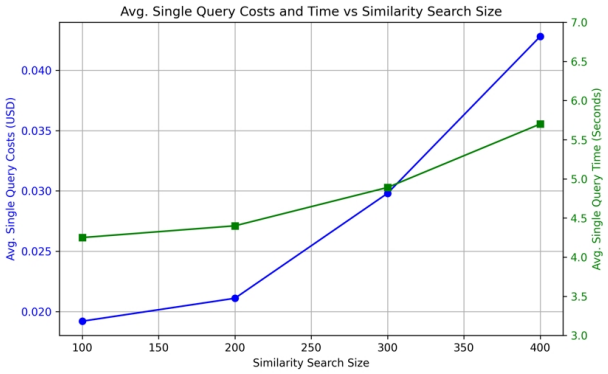
In ARCA-PoC, we use "gpt-4o" from OpenAI to serve as both the Evaluating LLM and the Generator LLM.

4. Evaluation

To evaluate our work, we first build build our data set of bug tickets containing descriptions, logs and performance



(a) Accuracy



(b) Cost

Figure 4. ARCA-PoC accuracy and cost analysis.

metrics. Then we build ARCA-PoC knowledge base using 80% bug tickets and evaluate the PoC with the held-out 20% tickets.

4.1. Data Set

We use a micro-service workload generator, called "DeathStar" (Gan et al., 2019) to run different micro-service applications, like "HotelReservation", "SocialNetwork", etc. During the application runtime, our synthesized errors fall in four categories: not enough CPU computation power, memory leak, network delay and mixed. All together we have 200 bug reports for each category and 800 in total.

For each bug, we use "gpt-4o" to generate a human readable bug report using prompt like "the issue is caused by a random delay in every invocation of the call back function XXX" to enable the bug report contains meaningful mitigation plans.

In this way, we have created 800 bug tickets that contain the bug descriptions with mitigation plans, the time-series of performance metrics and the logs.

4.2. Results

We first study the effect brought by 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-PoC needs to include the closest bug in the output of the triage step. For the whole system to be accurate, ARCA-PoC needs to pick the labeled closest bug as the output of the Evaluation LLM. The results are shown in Fig. 4-a.

In our test, we have increased the similarity search output size from 100 to 400. As we can see, the triage accuracy keeps increasing with the raise of the triage group size, when the system accuracy is limited by the triage accuracy since the Evaluation LLM will be wrong if the right answer is not given. However, the system accuracy drops when we increase the similarity search size from 300 to 400 because having too many choices greatly increases the input noise for the Evaluation LLM and thus lowered the performance.

It's also worth noting that we cannot increase the similarity search output size without a limit. GPT-4o, has a context window size limit of 30,000 tokens and we will reach that limit if the triage group size is slightly more than 400.

Meanwhile, we also study the time and financial cost per query. The gpt-4o uses 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 number of tokens computed. Taking all these considerations together, we have reached our results shown in Fig. 4-b. As we can see, for large group size, the generation is significantly slower and the cost mounts up easily.

5. Conclusion

ARCA is a work in progress, but already revealing promise for a new multimodal RAG LLM approach to searching incident report databases for events that reflect prior troubleshooting of complex cloud-hosted applications. This is the first trial of creating end-to-end software incident management system per our knowledge.

References

Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., and et al. Language models are few-shot learners, 2020. URL https://arxiv.org/abs/2005.14165.

Gan, Y., Zhang, Y., Cheng, D., Shetty, A., Rathi, P., Delimitrou, C., and et al. 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, ASPLOS '19, pp. 3–18, New York, NY, USA, 2019. Association for Computing Machinery. ISBN 9781450362405. doi: 10.1145/3297858.3304013. URL https://doi-org.proxy.library.cornell.edu/10.1145/3297858.3304013.

Gao, Y., Xiong, Y., Gao, X., Jia, K., Pan, J., Bi, Y., and et al. Retrieval-augmented generation for large language models: A survey, 2024. URL https://arxiv.org/abs/2312.10997.

Jiang, Y., Zhang, C., He, S., Yang, Z., Ma, M., Zhang, D., and et al. Xpert: Empowering incident management with query recommendations via large language models, 2023. URL https://arxiv.org/abs/2312.11988.

Johnson, J., Douze, M., and Jégou, H. Billion-scale similarity search with gpus, 2017. URL https://arxiv.org/abs/1702.08734.

Lewis, P., Perez, E., Piktus, A., Petroni, F., Karpukhin, V., Goyal, N., and et al. Retrieval-augmented generation for knowledge-intensive nlp tasks, 2021. URL https://arxiv.org/abs/2005.11401.

Shinn, N., Labash, I., Atanasova, P., and Ribeiro, L. Reflexion: Language agents with verbal reinforcement learning. *arXiv preprint arXiv:2303.11366*, 2023. URL https://arxiv.org/pdf/2303.11366.

Wei, J., Wang, X., Schuurmans, D., Bosma, M., Ichter, B., Zhou, D., and et al. Chain-of-thought prompting elicits reasoning in large language models, 2023. URL https://arxiv.org/abs/2201.11903.

Xiong, Y., Jiang, Y., Yang, Z., Qu, L., Zhou, L., and et al. Superbench: Improving cloud ai infrastructure reliability with proactive validation, 2024. URL https://arxiv.org/abs/2402.06194.

Yao, S., Cao, J., Yu, D., Narasimhan, K., Callison-Burch, C., and Clark, P. React: Synergizing reasoning and acting in language models. *arXiv preprint arXiv:2210.03629*, 2022. URL https://arxiv.org/pdf/2210.03629.

Zheng, L., Chiang, W.-L., Sheng, Y., Zhuang, S., Wu, Z., Stoica, I., and et al. Judging llm-as-a-judge with mt-bench and chatbot arena. In *Proceedings of the 37th International Conference on Neural Information Processing Systems*, NIPS '23, Red Hook, NY, USA, 2024. Curran Associates Inc.

Zhu, Y., Meng, W., Liu, Y., Zhang, S., Han, T., Tao, S., and Pei, D. Unilog: Deploy one model and specialize it for all log analysis tasks, 2021. URL https://arxiv.org/abs/2112.03159.

A. Future Exploration

To further enhance ARCA's capabilities in mitigating site reliability issues, we propose developing a comprehensive mitigation strategies dataset that enables the AI-based system to provide targeted, accurate responses to recurring incidents.

The dataset will include detailed mappings of common SRE challenges (e.g., CPU constraints, memory leaks, and network delays) to mitigation strategies, performance metrics, and success rates observed in prior cases. By integrating this dataset into ARCA, the system can proactively predict and automate mitigation actions, such as resource reallocation, traffic throttling, or service restarts. Additionally, by learning from historical logs and performance traces, ARCA can recommend long-term solutions that address root causes and minimize future occurrences.

This approach will allow ARCA to serve as a dynamic decision-making assistant, empowering SREs with real-time incident responses and actionable insights, thus improving reliability and operational efficiency in complex cloud systems. Future work will also explore cross-modality search mechanisms that unify logs, traces, and metrics into a singular actionable response, ensuring ARCA delivers holistic incident resolution.

Sometimes, the underlying causes of bugs can vary significantly, even when users report similar issues. To improve ACRA's bug diagnosis capabilities, we plan to develop methods for identifying the root causes of bugs, enabling us to offer more effective and targeted solutions for resolving them. Tracing back the cause of the error in the code with reference to ReAct (Yao et al., 2022) and Relfecxion (Shinn et al., 2023), or requesting a portion of the log for diagnostic purposes as the user did before the error occurred, or even re-giving the correct Instruction based on the main claim, are all potentially feasible in the course of our experiments. We will distinguish their applicability conditions and optimize the debug approach in future work.

B. Team Breakdown

I worked on building phase, retrieving system perfor- mance metrics from bug reports and performing Root Cause Analysis (RCA) to identify key factors impacting system behavior. As part of this analysis, I experimented with Granger Causality and the Vector Autoregression (VAR) methods. In this class, I gained a comprehensive understanding of every stage involved in building and deploying Large Language Models (LLMs). The course was structured around in-depth discussions of cutting-edge research papers, where we critically analyzed, wrote reviews, and presented these papers to our peers. This experience not only deepened my theoretical knowledge of LLMs but also honed my research and presentation skills, fostering a collaborative environment to explore complex topics like model architectures, training strategies, and evaluation methods. I deserve an A+ grade as analyzing time series data for performance metrics is inherently complex, requiring extensive research and careful study of academic papers to develop an approach, that accurately fits our spe- cific scenario.

C. Prompts

```
{
    "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 5. 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 6. The 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 7. 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 8. The prompt for Generator LLM.