RCA-Agent for ZAIOPs

Rishabh Jain[1], **Devika Sondhi**[2], **Sandeep Hans**[2]

[1]IIT Delhi,
[2]IBM Research, India

Abstract

This project focuses on developing an AI-powered Root Cause Analysis (RCA) agent for IBM Z systems—mainframes that support high-volume, mission-critical workloads. Currently, RCA is performed manually by on-call engineers, requiring deep domain expertise and significant effort to analyze vast amounts of log and metric data. The goal of this work is to automate and augment the RCA process. In the initial phase, the project explored various RAG approaches using static data sources such as product documentation, blogs, and transcripts from technical videos. After collecting and preparing multi-modal data via scraping techniques, early experiments revealed significant shortcomings in generation quality. The two core challenges that are attributed to this is: ineffective retrieval and suboptimal generation approach. Subsequent work focused on improving retrieval performance through experimentation with different chunking strategies and embedding models, leading to a significant improvement in retrieval accuracy. Following efforts are aimed at optimizing the generation component of the pipeline and establishing an evaluation framework for the generations for real-world Z system issues. This project demonstrates the potential of AI agents in reducing the manual effort involved in RCA and sets the foundation for building a more dynamic, data-driven reasoning system for enterprise-scale operations.

Introduction

Modern cloud-based software systems, particularly those supporting enterprise-scale operations, are growing in complexity. Characterized by multi-tiered architectures and distributed components, these systems are prone to frequent and costly production incidents. Timely and accurate identification of the underlying causes is essential to mitigate downtime and ensure system reliability. Root Cause Analysis (RCA) plays a critical role in this process.

Conventional RCA techniques—such as the 5 Whys, Fishbone Diagrams, Fault Tree Analysis, and Pareto Analysis—depend heavily on human expertise and manual inspection. While these methods have been effective in the past, they struggle to keep up with the sheer volume and diversity of data generated in modern environments, including logs, metrics, and traces. The effort required to sift through this information makes RCA a time-consuming and error-prone task.

Recent advancements in large language models (LLMs) offer new opportunities to augment the RCA process. However, current automated solutions typically operate only on static data and lack the ability to access and reason over real-time diagnostic signals, which are often critical for accurate analysis. This limitation hampers their effectiveness in dynamic, high-stakes production environments.

This project aims to build an LLM-powered RCA agent for IBM Z systems—mainframes that handle mission-critical workloads at scale. In its initial phase, the agent leverages retrieval-augmented generation (RAG) using static sources such as product documentation, blogs, and technical transcripts. After collecting and processing multimodal data through web scraping, early experiments revealed challenges in both retrieval and generation. Through systematic enhancements, including refined chunking strategies and improved embedding models, retrieval accuracy was increased.

Subsequent efforts focus on extending the system to handle real-world incident scenarios and optimizing its reasoning capabilities. Lastly, an evaluation framework which primarily relies on LLM-as-a-Judge was deployed to grade the LLM generations.

Our contributions are:

- A custom RAG pipeline tailored for IBM Z mainframe logs and artifacts.
- An evaluation framework combining LLM-based scoring with human judgment.
- A new dataset of RCA examples drawn from system tools and human queries.

Related Work

Root Cause Analysis (RCA) in cloud systems has gained traction with the advent of Large Language Models (LLMs). Traditional AIOps approaches [1] have struggled with limitations such as data quality issues, poor generalizability, and significant human supervision. To mitigate these problems, recent works have proposed LLM-augmented frameworks that better align with the dynamic and heterogeneous nature of cloud environments.

RCAgent [2] introduces a tool-augmented autonomous agent framework that leverages internally hosted LLMs for

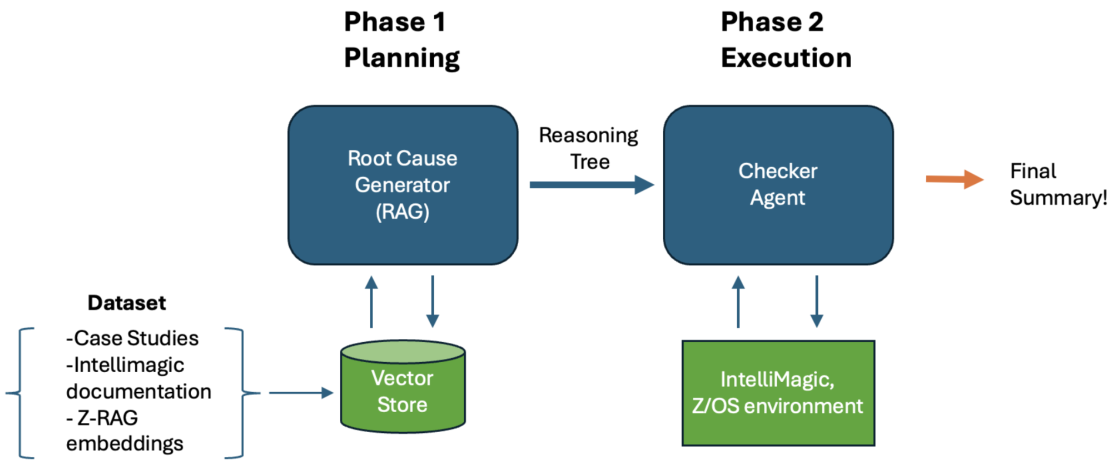

Figure 1: Our end-to-end workflow

privacy-aware RCA. It enhances the classic ReAct framework [3] by introducing several mechanisms: an Observation Snapshot Key (OBSK) to handle long-context limitations, domain-specific expert agents (for log and code analysis), and a novel trajectory-level self-consistency mechanism for reliable multi-step reasoning. RCAgent is evaluated on Alibaba Cloud's real-world Flink platform, showing consistent improvements over prior methods on root cause, solution, and evidence prediction tasks.

AgentRCA [1] follows a similar tool-augmented LLM agent paradigm but operates primarily with an externally hosted GPT-based architecture. It focuses on the flexible integration of multiple tools including metric, log, and topology-based RCA tools. The authors demonstrate that decomposing RCA tasks into subtasks (e.g., identifying the affected component and RCA type) can significantly improve performance. Unlike RCAgent, AgentRCA emphasizes human-agent collaboration and introduces task-specific planning modules for better controllability.

While both RCAgent and AgentRCA represent major advancements in LLM-based RCA, they differ in their assumptions regarding deployment constraints and tool interfaces. Our work builds upon these foundations by extending RCA to multi-modal cloud data, including product documents, case study blogs, videos, tuning letters as well as dynamic log data in the future. Our focus lies in the RCA for IBM Z mainframes, and we intend to adapt our agent-based approach to effectively handle domain-specific data sources.

Method

The RCA-Agent is built on a modular, API-driven architecture. The core components work together to deliver accurate root cause analysis.

System Architecture

Our RCA agent is built on a modular, API-driven architecture designed to support dynamic, retrieval-augmented root cause analysis. The system comprises the following components:

1. **UI (Streamlit)**: A web-based user interface allows users to input queries and interact with the RCA agent.
2. **APIs (Flask)**: The interface communicates with multiple backend APIs developed using Flask. Each endpoint is linked to a specific RAG pipeline or a combination of models.
3. **RAG Pipelines** : These pipelines form the core of the system. Built using LangGraph, they support adaptive and stateful reasoning workflows including:
   - Query routing and transformation
   - Document retrieval from vector databases
   - Relevance and hallucination grading
   - Final response generation
4. **LLMs & Embedding Models**: The system integrates large language models (LLMs) from IBM's WatsonX and RITS. We use various `SentenceTransformers` models (e.g., `bge-large-en-v1.5`, `granite-embedding-30m-english`) for generating embeddings.
5. **Vector Store (ChromaDB)**: All knowledge sources are chunked, embedded, and stored in a `ChromaDB` vector store to support efficient semantic search and retrieval.

RAG Pipelines

We implemented three RAG pipelines tailored for different use cases:

- **RAG2**: Utilizes only custom-collected data (blogs, docs, transcripts) for both retrieval and generation.
- **ZRAG**: Employs embeddings and knowledge provided directly by the IBM Z team.
- **RAG1**: Combines both custom-collected data and ZRAG embeddings for hybrid performance.

RAG Workflow

The RAG pipeline is implemented using LangGraph and follows the workflow illustrated in Figure 2. The workflow consists of several key states:

- **retrieve**: Fetches relevant documents from the vector store.
- **grade_documents**: Ranks the documents by relevance and filters out hallucinated responses.
- **transform_query**: Refines the query based on the graded documents.
- **generate**: Produces the final response, conditioned on the refined query and retrieved context.

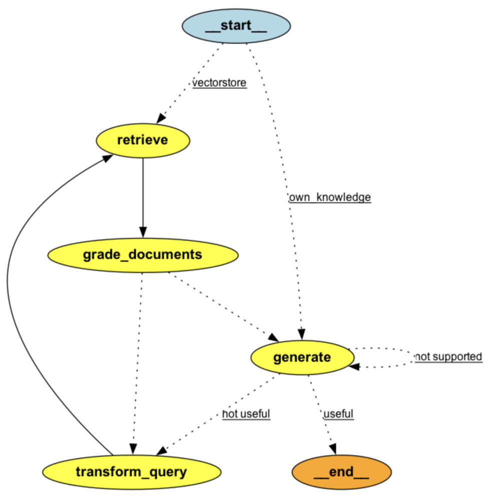

Figure 2: RAG Workflow using LangGraph

Data Collection

The data used to train and evaluate the RAG pipelines consists of diverse, multimodal sources (within IBM Community resources) processed into text format:

- **IBM Z Blogs**: Scraped from the blog url pages. Image captions were removed due to unreliability.
- **Performance and Capacity Blogs**: Same procedure as above.
- **Technical Videos**: MP4 files were downloaded and transcribed using OpenAI's `Whisper` model.
- **Intellimagic Documentation**: Parsed from PDF using pypdf library.
- **Tuning Letters**: Extracted similarly from PDF using the same parser tool. Data in these PDFs was more elaborate and unstructured, so it has not been included in the experiments mentioned later. Though, its inclusion (after processing) might be quite relevant to our use-case.

All data sources were chunked and embedded before being stored in the vector database for downstream use in RAG pipelines.

Experiments

A subset of five documents was selected from the dataset and the following queries were created based on them (one query corresponding to each document).

- There is an increased disk connect time. What is the cause?
- There is an application slowdown. What is the cause?
- There are more TCP received packets discarded than expected. What is the cause?
- There is a DB2 slowdown. What is the cause?
- Spike in XCF (cross-system coupling facility) traffic. What is the cause?

A RAG (pipeline explained earlier) was developed using just these five documents. Then these queries were given to the RAG as well as LLMs including `granite-34b-code-instruct` and `llama-3-3-70b-instruct`. The LLM responses in both cases were incomplete, incorrect or ambiguous. On the other hand, the RAG responses were relevant and to the point.

The motivation of this experiment was just to evaluate the domain-knowledge expertise of the two LLMs (one being `granite-34b-code-instruct` which is IBM's own LLM) on their knowledge of issues arising in mainframes (IBM Z in this case). It was concluded that the LLMs themselves are lacking the know-how, hence, would require additional information from a relevant dataset to answer the queries.

ZRAG

ZRAG is a pre-existing RAG that which contains a large amount of data (mostly product documentations related to IBM Z). Although ZRAG is not tuned to perform Root cause analysis on IBM Z related errors, the same set of five queries (mentioned above) were given to it. To summarize the findings, the ZRAG responses were very abstract and it became clear that it is not suitable for performing RCA on IBM Z related issues. However, the theoretical knowledge-base of ZRAG is critical, and utilized in some of our experiments.

RAG1

Once all the data was collected and converted to text format, embeddings were trained using the `granite-embedding-30m-english` model. This embeddings model was chosen because the ZRAG embeddings were trained on it. In this experiment, ZRAG data has been clubbed with all the data that was collected (mentioned in the Data section).

A variable chunk size was observed in ZRAG embeddings (with maximum chunk size being nearly 2000 tokens) and zero overlap. The logic of tokenization for ZRAG embeddings could have been related to semantic similarity, however it was not completely clear. So, the collected data was embedded with a chunk size of 512 and an overlap of 20%. Finally, all the embeddings (of collected data as well as ZRAG) were now fed to the RAG (RAG1).

A query benchmark of 30 questions was created. In order to do so, 30 chunks were sampled amongst all the chunks fed to the RAG (with representation from all data sources), and one query-answer pair was created for each chunk, using llama-70b LLM. For evaluation of the RAG1 responses of these queries Granite-3.1-8b was used as a judge, with the following prompt:

> Evaluate the model's answer based on the following criteria:
>
> 1. **Factual accuracy** – Does it contain correct and relevant information?
> 2. **Completeness** – Does it fully address all parts of the question?
> 3. **Clarity** – Is the answer well-written, concise, and easy to understand?
> 4. **Alignment** – Does it match the tone and intent of the expected answer?
>
> Your task is to assign a **score from 1 to 10**, where:
>
> - **10** = Perfect: matches or exceeds the expected answer in quality.
> - **7–9** = Good to Very Good: mostly correct and complete with minor issues.
> - **4–6** = Fair to Moderate: partially correct or incomplete.
> - **1–3** = Poor: mostly incorrect, irrelevant, or unclear.

Following is a table containing the average scores (datasource-wise):

Source	Score
Transcripts	5.6/10
IBMZ-Blogs	7.8/10
Performance-Blogs	7.8/10
Tuning-letters	6.0/10
ZRAG-emeddings	8.6/10

Table 1: Evaluation scores for RAG1 across different document sources.

Our analysis, supported by human feedback, revealed significant room for improvement in the system's responses. To systematically address this, the problem was decomposed into two key components: *Retrieval* and *Generation*. The subsequent experiments are designed to improve each of these components in isolation. Furthermore, the focus will be exclusively on the data collected from this point forward—excluding the earlier ZRAG embeddings. The new setup is referred to as **RAG2**.

Retrieval

The objective is to retrieve the most relevant chunks for a given query. The first step toward this was to synthesize query–chunk pairs. Once these pairs are prepared, we evaluate whether the relevant chunk(s) are retrieved when the query is submitted to the vector database (in our case, `ChromaDB`). The experiments were categorized into three types: retrieval of queries derived from one chunk, multiple chunks, and case studies.

Single Chunk Retrieval The motivation of this experiment is to test the retrieval accuracy of various retrieval specifications. Specifically, different embeddings models and chunking strategies were tried out to achieve the maximum possible retrieval accuracy. The setup of this experiment includes 100 chunks (randomly sampled from all the chunks that were fed to the RAG). Then for each chunk, a description and a query were generated using `llama-3-3- 70b` model. The following prompts were used:

> Based on the following text, generate one relevant question and a brief answer for it. Text: {chunk_text} Generate a question that can be answered using this text, followed by a concise answer. Format your response as: Query: [question] Answer: [answer]

Now, we tested the retrieval on these queries. For each query top 10 most relevant chunks were retrieved from the vector database. Since, each query was created with one chunk as reference, we will consider the retrieval successful if that chunk appears in the top 10 retrieved chunks. In the observation table (add reference), the frequency of appearance of correct chunk in top 10 relevant chunks and well as the top most relevant chunk has been noted.

A set of five experiments were performed with different embeddings models, chunking strategies and retrieval methods. These have been listed in the observation table (Table 3). The embeddings models used are available on the `SentenceTransformers` library in python.

Multi Chunk Retrieval The motivation of this experiment was to validate the retrieval of chunks for queries which might have relevant information spread across various chunks.

Ten chunks were randomly sampled from the set of all chunks. Then, for each chunk, the top four most semantically similar chunks were retrieved (based on cosine similarity of chunk embeddings vectors). So, we ended up with 10 clusters each consisting of five related chunks. Now, a query was created for each cluster using the same prompt as given in the single chunk query experiment and the query was given to the database. Then, the number of chunks (out of five in the cluster) that were retrieved among the top 10 chunks retrieved was noted. An average of **4 out of 5** chunks (from the correct cluster) were retrieved in the top 10 relevant chunks retrieved.

Generation

A set of five case study queries were considered for the following experiments.

- What could be the root cause of db2 slowdown?
- Why could there be a spike in the TCP/IP address space demand?

Table 2: Retrieval performance on RAG2 data across different chunking and embedding strategies

ID	Model	Chunking Strategy	Retrieval Method	Score (k=10)	Score (k=1)
db_retrieval_1	granite-30m	max_chunk_size=512, sentence break (no overlap)	vector similarity	74%	42%
			mmr	67%	42%
db_retrieval_2	all-MiniLM-L6-v2	max_chunk_size=512, sentence break (no overlap)	vector similarity	76%	43%
			mmr	70%	43%
db_retrieval_3	all-MiniLM-L6-v2	semantic chunking (TextTiling)	vector similarity	65%	40%
			mmr	74%	40%
db_retrieval_4	**bge-large-en-v1.5**	**semantic chunking (TextTiling)**	**vector similarity**	**85%**	**53%**
db_retrieval_5	granite-125m	semantic chunking (TextTiling)	vector similarity	75%	47%

- How should I identify and reduce excessive XCF traffic that's impacting CPU usage?
- There is a sudden increase in connect time. What could be the root cause and how can I fix it?
- The system experienced application slowdown overnight. What could be the root cause and how can I fix it?

For generation, two LLMs were tested which include Granite-8b and Llama-405b. The following prompt was used:

> You are an expert in Root Cause Analysis (RCA) for IBM Z Mainframe related incidents.
> Query: {question}
> Context: {context}
> Instructions:
> Based on the query and the provided context, identify all possible root causes.
> For each cause, provide the following:
> 1. A concise description of the cause.
> 2. A detailed explanation of how it contributed to the issue.
> 3. The source(s) in the context that support this cause, specified as chunk_id(s).
> Format your response as a numbered list like this:
> 1. Cause: <cause description> Explanation: <detailed explanation> Source(s): <chunk_id(s)>

Evaluation

There does not exist a precise ground truth as the answers to the case study queries. For this reason, LLM-as-a-judge approach has been explored in this study. Various LLMs were used as judges, and a **Mixture of Experts** paradigm was established. The following LLMs were used as judge:

- `DeepSeek-V2.5`
- `llama3-3-70b`
- `mixtral-8x22b`
- `mixtral-8x7B`
- `qwen2-5-72b`

And the following prompt (after refinement) was used for all LLMs:

> You are an expert troubleshooting AI for enterprise systems. Your task is to evaluate how well a provided **cause and explanation** address a specific **query** about a system issue.
> You will receive:
> – A query about a potential issue
> – A proposed cause for that issue
> – An explanation of the cause
> Assess the following aspects when giving your evaluation:
> – **Relevance**: Does the cause and explanation directly address the issue raised in the query?
> – **Correctness**: Is the cause technically sound and factually accurate?
> – **Clarity**: Is the explanation easy to understand and well-articulated?
> Based on your assessment of these dimensions, assign a score from **1 (poor)** to **10 (excellent)**, and provide a **short reason** (1–2 sentences) justifying the score.
> Return your response **strictly** in the following JSON format (do not include any other text):
>
> ```
> {
>   "score": <number>,
>   "reason": "<short reason>"
> }
> ```
>
> Query: query
> Cause: cause
> Explanation: explanation

Once all the responses were generated for the five case queries, the above evaluation was performed and each reason (for each response) was classified into low, mid or high quality based on majority vote from the mixture of experts.

Table 3: Reason generation response scores

	Granite-8b	**Llama-405b**
low	1	1
mid	42	45
high	17	14

Note: Low = majority judge scores in 1–3, Mid = 4–7, High = 8–10.

The majority of responses were rated as having medium (mid) quality in terms of relevance, correctness, and clarity. However, validating these scores remains a limitation of our evaluation pipeline. The LLM-as-a-judge approach provides only a coarse estimate of response quality, as the models often lack the domain-specific knowledge required to assess nuanced technical details. One notable limitation is the model's tendency to hesitate in decisively rejecting incorrect responses, which can lead to overestimation of quality.

Conclusion

This work presents an AI-powered RCA agent tailored for IBM Z mainframe systems. Through a combination of custom data collection, modular RAG pipelines, and iterative evaluation, we identified key challenges in both retrieval and generation. Our findings emphasize that standalone LLMs—even state-of-the-art models—lack the domain expertise needed for nuanced RCA in enterprise environments. By narrowing focus to curated domain-specific data and refining both embedding strategies and prompting techniques, we significantly improved retrieval precision and response relevance.

We also introduced a hybrid evaluation methodology combining LLM-as-a-Judge and human feedback, exposing current limitations in automated judgment for technical accuracy. While the majority of generations were of moderate quality, this signals a strong baseline for future improvements.

While the current work is heavily based on a RAG pipeline, the future work shall target strengthening the agent's capabilities by consuming SMF data reports to derive more nuanced reasoning in addition to the knowledge base existing in RAG.

Also, in the future, we plan to extend our agent's capabilities to IBM's tuning letters as well as dynamic data sources including logs and real-time signals. This expansion, paired with human-in-the-loop validation, will help evolve the agent into a trustworthy assistant for RCA in production-grade mainframe systems. Ultimately, this work lays a foundation for scalable, AI-driven diagnostics in enterprise computing.

References

[1] Z. Yu *et al.*, "Agentrca: Tool-augmented llm agents for root cause analysis," *arXiv preprint arXiv:2403.04123*, 2024.

[2] Z. Wang, Z. Liu, Y. Zhang *et al.*, "Rcagent: Cloud root cause analysis by autonomous agents with tool-augmented large language models," in *Proceedings of the 33rd ACM International Conference on Information and Knowledge Management (CIKM)*, 2024.

[3] S. Yao *et al.*, "React: Synergizing reasoning and acting in language models," *arXiv preprint arXiv:2210.03629*, 2022.