System of Intelligent Actors: The DevOps chapter

Ravindra Manjunatha, Tatsuya Arai, Yinxiao Zhang, Deepanshu Rastogi, Goutham Nareddy, Nate Slater, Anshuman Mishra, Mohan Nataraj, Deap Ubhi, Sunil Mallya[*]
`sunil@flip.ai` [1]

[1]Flip AI, San Francisco and Bangalore

This report introduces a new Mixture of Experts (MoE) LLM by Flip AI that aims to automate several important tasks in the DevOps and Observability domain. Our vision is to enable developers to debug incidents by starting from a detailed analysis of what's happening in their environment rather than an alert. The report generated by Flip AI includes potential root cause analyses (RCA), allowing developers to swiftly resolve incidents in minutes as opposed to hours of debugging in a war room. Looking ahead, we anticipate reflecting on the past practice of repeatedly querying various observability storage systems for debugging production incidents as an obsolete and cumbersome approach. The intended audience for this report is chief technologists at large enterprises and developers interested in the next generation of observability capabilities.

Abstract

One could argue that the most creative aspect of a software developer's job is writing code, which they love to do. One could also argue that the least creative aspect of a software developer's job is to get the dreaded on-call 2am page to debug a production incident. The average time it takes to debug production incidents is in the order of a few hours. While there has been an abundance of research and startup activity in enabling developers to code faster, similar effort and energy have not been invested to reduce the operational pain points facing developers, devops engineers and SREs in debugging and remediating production incidents more efficiently. To address this very problem, we at Flip AI built the world's first Large Language Model (LLM) for DevOps and Observability, *Flip DevOps LLM*. Popular LLMs have been shown to be effective at coding tasks, however they aren't near ready to debug incidents in complex enterprise environments. In this technical report, we showcase how Flip AI's Mixture of Experts (MoE) style LLM, trained on 100+ billion tokens of DevOps related domain data, is vastly superior to general LLMs (open source and proprietary API-based LLMs) in debugging production incidents. Flip DevOps LLM is composed of specialized sub-networks that are experts in a few dozen tasks each. Based on extensive benchmarks that include real world production incidents, we find Flip DevOps LLM is able to debug production incidents in complex production environments with high accuracy in under a minute and on several internal benchmarks, 67.6% better than the best general open source MoE Mixtral model, and 34.5% better than GPT-4.

---

[*] CTO, Flip AI and corresponding author.

Acknowledgment: We would like to thank our team members Vinayak Rao, Vinoth Thirumalai, Subhadip Ghoshal, Vishal Gowda, Benji Haye, Harish Arora and Corey Harrison.

1 Introduction

In the modern era, virtually every large enterprise or institution in the world is dependent on software to orchestrate their core businesses. As such, each of these enterprises must also rely on a constellation of monitoring tools to determine the health of their software applications and underlying infrastructure at any given point in time. Today, technical leaders and developers have an abundance of choice when it comes to observability platforms, from commercial platforms to open source alternatives; and yet, the practice of debugging incidents remains far from an efficient, methodical and timely process. In almost every organization, the critical process of observing systems and bringing them back to health has devolved into a convoluted process, resembling a porridge of procedures aimed at restoring operational health. For a truly vexing and potentially catastrophic incident, you bring in the grizzled veteran warrior — a DevOps Achilles, a skilled professional well-versed in navigating the intricate maze of logs, performance metrics, and intricate system interactions - in other words, someone who knows where all the secrets reside. These specialists bear the weight of swift resolution on their shoulders, imparting their expertise in the war rooms of Slack, Teams and Polycoms. There is just no compression algorithm (yet!) for that kind of experience in debugging production incidents. Flip AI is changing this. We set on a journey to compress the operational knowledge of those "set of warriors" that exist on your teams, but at 10x the speed. In other words, we are building the world's foremost subject matter expert at debugging incidents across any permutation of application architectures, technologies, coding languages and infrastructure in the world.

So why haven't observability platforms gone the extra mile to solve this problem in the last 20 years? Simply, it's because observability tools have been reduced to expensive big data storage and retrieval systems, all of which have their own unique, arcane query syntaxes and quirks that lock in your data. They obsess over shaving off 500 milliseconds on a query that accesses 100GB of data, while all along the true bottleneck in the debugging process are humans and their ability to rapidly interpret that data. The ability to analyze and root cause incidents are not dependent on the faster retrieval of your observability data; rather it is because there are human experts that have an existing hunch based on past experience. They are able to author intelligent queries into these observability systems to retrieve the appropriate telemetry data, and sift through the data to find patterns and causal elements, all while keeping the production incident as context. Incident debugging and root cause analysis is not a big data problem, yet existing observability vendors' businesses are dependent on projecting it as one. In reality, the incident analysis process is a collaborative exercise among experts that can be broken down into four distinct phases.

Phases of Incident Debugging

1. Triage: Incident Triage and On-call Mitigation
2. Hypothesis: Hypothesis Formulation and Localization
3. Debug: Collaborative Debugging with Experts
4. Correction of Error: Resolution and Corrective Actions

A typical workflow of the incident debugging process begins with the creation of an incident report and/or a page to relevant on-call engineers after some pre-configured alert or alarm fires. Next, a war room of developers, subject matter experts and other stakeholders gets set up with the objective of getting back to normal operation of production software by finding the root cause and mitigating the incident. The first responder or the operator on-call in this group focuses on incident triage and problem identification, i.e understanding what is wrong, what is the impact, and what may be the fastest way to alleviate the incident. This is not quite the root cause fix, but a temporary band-aid. Mitigation Steps allow for the production system to restore to normal or acceptable operational state. Hence we carve out an explicit fourth phase for resolution and corrective actions that pertains to steps taken to restore to forward operation, i.e. production systems now run an improved version of the software. Phase two involves hypothesis formulation based on collected data, narrowing down the search for the underlying problem. Within the war room, many individual subject matter experts from cross-functional teams engage in intense communication and information-sharing to delve deeper into the identified hypotheses.

Although phase 2 and 3 appear distinct, they are often commingled and happen in parallel. The final phase, Resolution and Corrective Actions, focuses on developing a targeted action plan to address the root cause, implementing fixes, and monitoring the system's response to ensure effective resolution.

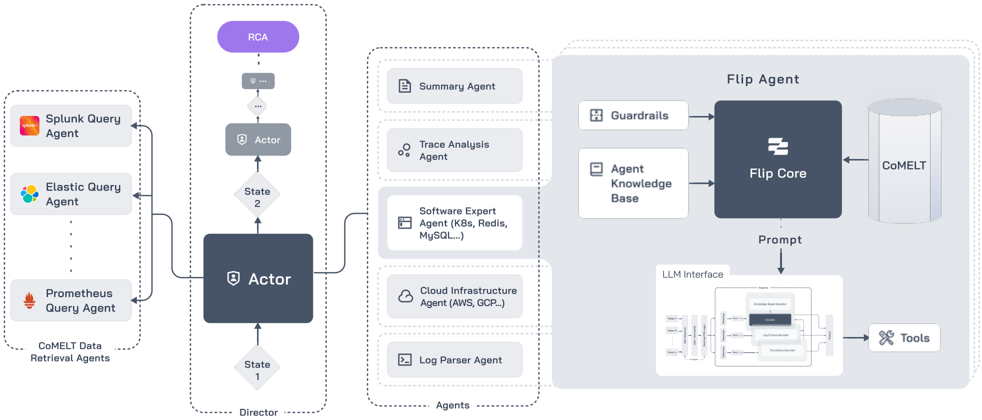


Figure 1: Flip AI System of Intelligent Actors

To automate the above incident debugging analysis, mitigation and root cause process, we at Flip AI have developed a Large Language Model-driven end to end workflow automation that mimics the war room process, which we refer to as “System of Intelligent Actors,” as illustrated in Figure 1 that are trained in a synthetically generated chaos environment, the details of which are in section 6. These actors are akin to subject matter experts and leverage the Flip DevOps LLM to query observability systems, reason through the data, perform analysis and summarize the potential root cause of an incident. Please refer to Appendix A for sample RCA reports from production systems that Flip AI has debugged.

2 Introducing CoMELT for Observability

Navigating incident debugging and root cause analysis involves quite a journey due to several complexities. First off, figuring out the root cause of an incident can be a bit like solving a puzzle – it might be one thing or a mix of different factors all tangled up. During an incident, you get hit with a bunch of alerts – some showing the way, others caused by chain reactions, and some just giving symptoms without revealing the actual root cause. Plus, software systems are a bit like moving targets, with hidden code paths that make it tricky to pinpoint a fixed root cause because things change so fast.

Now, the proof for why an incident happened is often hidden in the MELT data (Metrics, Events, Logs, Traces) collected during operations. But here’s the catch – observability tools usually stash this data in different places without a clear way to connect the dots, and there’s no magical key to connect it all in a meaningful way. This means humans have to step in, dive into each source, and piece the puzzle together. Oh, and developers don’t stop there – they also dig into build systems and actual code, leading us to coin the term "CoMELT" to amalgamate code and MELT.

Considering all these challenges, hunting for the "real root cause" can feel like being stuck in a bit of a maze. Sometimes, it’s not just about the final answer but understanding all the twists and turns in the evidence that lead us there.

3 Related Work

The exploration of automated incident debugging has predominantly emerged as a subset focused on the automatic identification of abnormal behavior within software systems. Throughout the industry’s history, automation has played a pivotal role, manifesting in statistical analysis to pinpoint anomalies and scripting to execute mitigation actions. Efforts have been made to automate mitigations directly[27], but these attempts have fallen short of providing a comprehensive solution due to the diverse nature of incidents, which varies across teams and evolves over time [1][11].

With the growing popularity of machine learning, this field has evolved, adopting the term AIOps within the industry. Key works in the industry have concentrated their automation endeavors on specific use cases, such as debugging applications [24], log analysis [17], system logs [9][7], application performance with traces [21], network anomaly detection [4], and software-defined network debugging [23]. These approaches leverage historical data and system logs to train models capable of detecting deviations from normal operation.

The integration of automated incident debugging into continuous integration and continuous deployment (CI/CD) pipelines has been explored to expedite the identification and resolution of issues in software development workflows, thereby enhancing the reliability of software systems [13]. Several software vendors in the industry, including Big Panda, Moogsoft, Datadog, Dynatrace, among others, have introduced their own variations of AIOps, tailored for anomaly detection in log and metric time series data. Practically speaking, AIOps has faced challenges in delivering value to its customers, particularly due to its emergence during a time when substantial quantities of clean and labeled data were required to train older generations of machine learning models

More recently, with the advent of large language models, there have been attempts to automate RCA generation by fine tuning these models on specific use cases such as incidents in the cloud pertaining to network issues [6], database specific incidents [32] and IT operations [12]. These efforts concentrate on a limited spectrum of use cases and are predominantly assessed through question answering and multiple-choice question scenarios, rather than achieving a comprehensive, fully automated incident debugging solution.

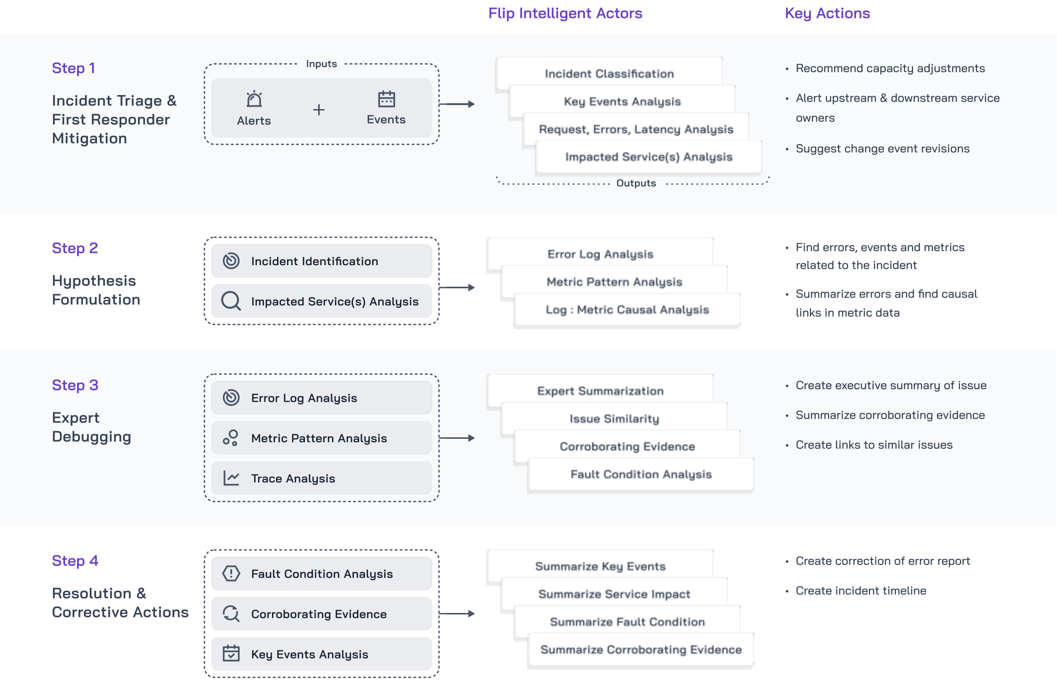

Flip Intelligent Actors
Key Actions
Step 1
Incident Triage & First Responder Mitigation
Inputs
Alerts
+
Events
Incident Classification
Key Events Analysis
Request, Errors, Latency Analysis
Impacted Service(s) Analysis
Outputs
Recommend capacity adjustments
Alert upstream & downstream service owners
Suggest change event revisions
Step 2
Hypothesis Formulation
Incident Identification
Impacted Service(s) Analysis
Error Log Analysis
Metric Pattern Analysis
Log : Metric Causal Analysis
Find errors, events and metrics related to the incident
Summarize errors and find causal links in metric data
Step 3
Expert Debugging
Error Log Analysis
Metric Pattern Analysis
Trace Analysis
Expert Summarization
Issue Similarity
Corroborating Evidence
Fault Condition Analysis
Create executive summary of issue
Summarize corroborating evidence
Create links to similar issues
Step 4
Resolution & Corrective Actions
Fault Condition Analysis
Corroborating Evidence
Key Events Analysis
Summarize Key Events
Summarize Service Impact
Summarize Fault Condition
Summarize Corroborating Evidence
Create correction of error report
Create incident timeline

Figure 2: Phases of Incident Debugging

4 Problem Statement

In the context of software production systems, an incident signifies any occurrence leading to service disruptions or a decline in operational quality. In response to such incidents, the process of root cause analysis becomes imperative for uncovering the underlying causal issues responsible for the disruptions. The RCA process in production systems encompasses diverse stages, starting with the collection of incident-related data, sourced from logs, metrics, traces, and/or alerts. Subsequently, a thorough analysis of the gathered data is undertaken to identify patterns, anomalies, or correlations that may offer insights into the root cause. Following this, hypotheses regarding potential root causes are formulated and rigorously verified, akin to the process described in cloud services incident resolution. The problem that Flip AI focuses on is the automation of root cause analysis for production incidents by debugging the issue using data in multi-modal sources, thus minimizing the time to detect incidents, improving recovery time, and the number of expert hours required to resolve production incidents.

Challenges in automation of RCA arise from diversity in technology stacks, programming languages, architectural patterns, deployment environments, and configurations across organizations, impeding the development of a universally applicable automated solution. The variability in how incidents happen, the nature of their root causes coupled with the lack of standardized data formats and inconsistent logging practices, further complicates automation. In short, there's no lack of creativity when it comes to how humans can induce errors into production systems knowingly or unknowingly. Furthermore, most organizations that manage large scale systems capture different system telemetry in different observability systems of record, adding another layer of complexity to the debugging process. Addressing these challenges demands deep general domain expertise coupled with knowledge of nuanced differences in how software systems are architected for specific use cases. Two companies could be using the same canonical piece of technology (cache, database, etc.) in completely different ways, even if they were using the same technology and vendor, yet they could be on different software versions! In order to attempt automation, it is essential to understand all of the complexities involved in the software system before hypotheses can be formulated and explored in a programmatic way, which is why almost all of the incident and root cause analysis process today is inherently human-driven with or without the help of some tools.

5 Architecture

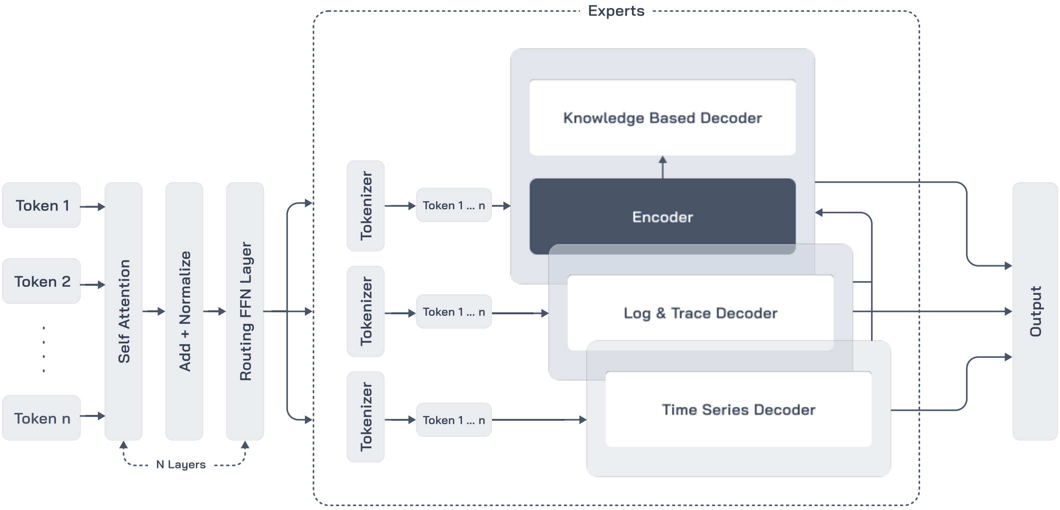


Figure 3: Flip DevOps LLM (MoE) Architecture

While the majority of the world is captivated by the power of general purpose LLMs, we believe in the power of domain specific language models (DLMs) to solve specific problems within an enterprise. The only argument for increasing the size of the language models beyond a certain threshold is to fit on low frequency patterns found in large noisy data corpora. We strongly disagree with the proposition of "scale is all you need." Instead, we argue that the scale is for those who do not want to invest in cleaner and disciplined approaches to training data. Our experiments repeatedly show that smaller models are easier to steer, tune, and more practical because they can be deployed with a smaller compute footprint. Recent results from smaller LLMs such as Phi2 [14], Mixtral[15] support these claims.

Given the complexity of the problem we are solving at Flip AI, our model needs to be good at a multitude of tasks across modalities (e.g. language, log, code, graphs, time series, etc). We started our journey with encoder-decoder models as the base model and saw high performance on a number of tasks particularly on log, language, and graphs. However some tasks, particularly code and time series related, were not at acceptable accuracy. Based on experiments, we found tokenization and decoder to be the most influential in cause and effect. It prompted us to build a single encoder multi-decoder architecture where each modality had a specialized decoder. With the popularity of mixture-of-expert architectures such as switch transformer [10], MoE from Meta[2] etc., we recently transitioned closer to a MoE style architecture. However, in our case each expert is independently trained, has its own tokenizer and decoder. Once experts are trained, we do a final training pass on the routing layer to help direct the input to the specialized expert in the network.

At the heart of our innovation is the abstraction layers we have built on top of our LLM that are responsible for planning, task decomposition and execution, Directors, Actors, and Agents, respectively, illustrated in Figure 1. Models and model architectures may evolve over time, and hence are not the correct abstractions to standardize or expose for external consumption. Many of the tasks that Flip needs to solve are multi-modal, i.e. each observability tool speaks a different language and reveals a different part of the story. This story can only be completed by connecting the dots across the data modalities and hence require coordination amongst many experts or chaining of results by many experts.

The automation of the debugging process starts with an incident alert, based on which the Director (planner) creates a runbook specific to debugging the incident and invokes the Actors to carry out the steps. The Director generates the plan using a custom domain specific language (DSL), specifically developed by Flip AI for incident debugging. An Actor is a self-sufficient module that uses one or more agents to automate a step in the runbook with clearly defined input and output schema. Actors are workflow specific. Simply put, it's equivalent to an API. Actors utilize Agents, that are a set of specialized toolkits for Actors to complete a specific task on CoMELT data such as log summarization, generating queries for observability platforms etc. Since they are the interface layer into the model, Agents are responsible to utilize a set of tools[26] from the library such as code interpreter, output guardrails, knowledge bases. Agents use different decoding techniques to reduce hallucinations [28], adhere to specific output formats, and ensure high quality results. In the workflow described, A Director might decide to back-track a particular path if the result is negative or there is a lack of supporting evidence from a particular Actor. Finally, all of the results get assembled to create a detailed RCA report for the developers to consume. This approach is similar to compositional reasoning put forward in the Chameleon[19] and tree of thought[30] paper albeit at the model task level abstraction.

6 Model Training

Given the exposure to diverse technologies and the edge cases that they unveil, It is challenging to bring LLMs that are production ready with common training regimes. At Flip AI, we are able to perform robust and scalable incident debugging thanks to our novel technology of chaos training which we call as *Otrashi*.[2]

6.1 Pre-Training: Continued domain pre-training

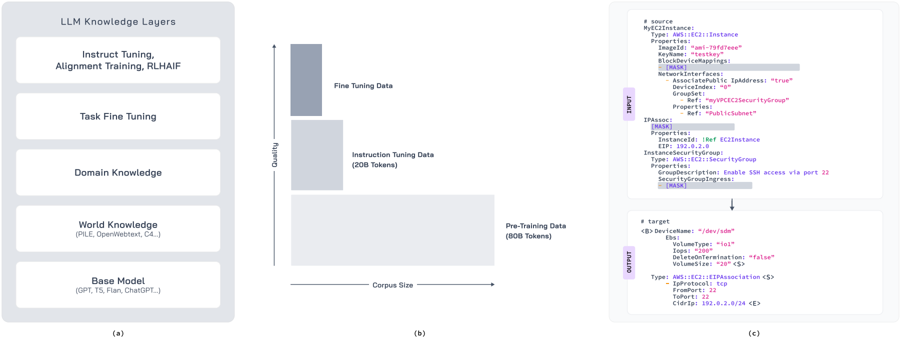
We leverage pre-trained (instead of zeroed-weights) general-purpose base models trained on trillions of tokens with exaflops of computing power. In constructing the DevOps knowledge base, we extend the pre-training phase using these released checkpoints. We meticulously curate a dataset comprising over 100 billion tokens sourced from diverse platforms, including open source code datasets, log datasets, scraping technical blogs, linux forums, github projects, etc., all possessing permissible commercial licenses. A significant proportion of these datasets are scraped from the internet and are inherently noisy. We strongly believe that artifacts from incorrect parsing or low data quality can affect outcomes, hence have developed a meticulous pipeline to filter low quality data.

Our training data pipeline incorporates minhash-based functions to eliminate duplicate content, binary data filters, classifiers assessing content relevancy, and document quality. To ensure the inclusion of diverse data modalities and comprehensive coverage of domains deemed vital by our customers, we maintain a careful balance. For the continuous pre-training phase, we construct self-supervised

---

[2] Otrashi means Chaos in the Indian language of Konkani and Kannada

datasets with different pre-training objectives. Our findings reveal that a singular masking or denoising technique falls short, prompting the adoption of specific corruption objectives tailored to distinct data domains or modalities. A majority of documents undergo transformation with the UL2[29] objective, while log and trace modalities use causal masking. For code and configuration related content, a novel corruption method is introduced, focusing on masking the most significant parts of the code to aid the model to learn the more challenging aspects as shown in 4c. The pre-training phase uses roughly 80 billion tokens and mirrors a curriculum training process, where simpler denoising-based datasets take precedence in the initial stages, progressing to more complex samples towards the end of training. We use PyTorch as our LLM training framework and find that to be the most convenient and robust in comparison to other brittle open source training wrappers.



```
# source
MyEC2Instance:
  Type: AWS::EC2::Instance
  Properties:
    ImageId: "ami-79fd7eee"
    KeyName: "testkey"
    BlockDeviceMappings:
    - [MASK]
    NetworkInterfaces:
      - AssociatePublic IpAddress: "true"
        DeviceIndex: "0"
        GroupSet:
          - Ref: "myVPCEC2SecurityGroup"
        Properties:
          - Ref: "PublicSubnet"
IPAssoc:
  [MASK]
  Properties:
    InstanceId: !Ref EC2Instance
    EIP: 192.0.2.0
InstanceSecurityGroup:
  Type: AWS::EC2::SecurityGroup
  Properties:
    GroupDescription: Enable SSH access via port 22
    SecurityGroupIngress:
    - [MASK]
```

```
# target
<B> DeviceName: "/dev/sdm"
       Ebs:
         VolumeType: "io1"
         Iops: "200"
         DeleteOnTermination: "false"
         VolumeSize: "20" <S>

    Type: AWS::EC2::EIPAssociation <S>
       - IpProtocol: tcp
         FromPort: 22
         ToPort: 22
         CidrIp: 192.0.2.0/24 <E>
```

Figure 4: (a) LLM Knowledge Stack (b) Training Data Hierarchy (c) Advanced Denoising

6.2 Instruction tuning and Fine tuning

We curate from existing instruction tuning datasets available in the open source world [20] [8] [31]. However, we mostly rely on creation of our own instruction tuning datasets using experts in the loop and evol-instruct[22] methodologies on high quality LLMs. This dataset, while it may appear to be synthetically generated, is much closer to real data because the generation is conditioned on existing content rather than prompting the LLMs to generate data without much context. We call this RAG-based[16] synthetic data generation. This generated data is also subjected to data cleaning and quality pipelines. Finally, the instruction data is varied based on the techniques that invert inputs [18] [8] and add prompt variations. For the task specific fine-tuning phase, we create the high quality of data that is subjected to double blind expert annotation passes, followed by an expert annotator arbitration [3] for conflicted labels. This phase of data annotation is expensive in cost and time but allows us to increase our model accuracy and trustworthiness in evaluation data.

Our annotation process and data pipelines have been refined over the past 18 months. This deliberate investment has proven to be crucial, contributing substantially to the high accuracy achieved in our tasks and Root Cause Analysis (RCA) process.

6.3 Chaos Training

The model acquires extensive knowledge during the initial three stages of training pertaining to DevOps and specific tasks associated with Root Cause Analysis (RCA) debugging. This is followed by pivotal phase of Chaos training. The knowledge accumulation for the model during the first three phases is analogous to the educational journey of an engineer progressing through high school, a computer science degree, and culminating in a PhD. However, the practical challenges encountered in the real-world scenario necessitate experiential learning. In order to build proficiency in incident debugging, a reinforcement learning-based approach is used on synthetically generated incidents, termed as *RLHAIF*, involving both human and AI feedback. Using OpenAI Gym [5] as a reference for reinforcement learning training environment, we developed an internal chaos gym for DevOps, named Otrashi. This chaos gym facilitates the generation of synthetic incidents in diverse environments such as Kubernetes, AWS, GCP, Azure, Serverless, VMWare, introducing chaos at the infrastructure, orchestration, or code level by intentionally injecting bugs. The subsequent Flip debugging process leads to the creation of a Root Cause Analysis report. Since the incident is intentionally generated by Otrashi, it passes all the metadata and scenario information to the RLHAIF trainer, enabling feedback-based training for enhanced model proficiency. This is the invention that sets us apart from

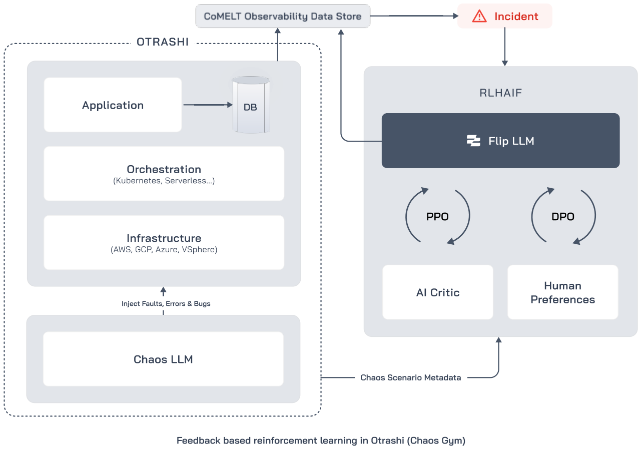

CoMELT Observability Data Store
Incident
OTRASHI
Application
DB
Orchestration
(Kubernetes, Serverless...)
Infrastructure
(AWS, GCP, Azure, VSphere)
Inject Faults, Errors & Bugs
Chaos LLM
Chaos Scenario Metadata
RLHAIF
Flip LLM
PPO
DPO
AI Critic
Human Preferences
Feedback based reinforcement learning in Otrashi (Chaos Gym)

Figure 5: Otrashi: Chaos Gym for Feedback Based Training

7 Evaluation

The primary focus of our evaluation centers on assessing the process of end-to-end automated incident debugging. Recognizing the complexity of the problem at hand, we believe that solely evaluating the ultimate goal of root cause generation is inadequate. Instead, our approach involves a comprehensive evaluation of all essential components necessary for the execution of root cause analysis. In the realm of true academic evaluation, the absence of direct baselines is notable, as no existing observability product or LLM is specifically tailored to address the end-to-end problem of RCA analysis. Consequently, the breakdown of tasks not only aids in assessing the baseline performance but also helps quantify the impact on accuracy from our training process. The true baseline for our evaluation remains as the expert humans in each sub-category who have annotated the data.

To establish a robust evaluation framework, we define domain-specific tasks integral to the root cause analysis process and rigorously assess the Flip model's performance against these tasks. Our evaluation methods adhere to statistically sound approaches, diverging from the prevalent trend of relying on GPT-4 as an automated analysis benchmark, which we deem flawed. Our commitment to robust evaluation is evident through meticulous curation of high-quality test data, majority of which are from real but anonymized production environments. We further ensure that all test samples used for evaluation are verified to be distinct from the training data. Upholding the principle of blind evaluation, the test sets are predominantly crafted by human subject matter experts by undergoing a double-blind annotation pass, and an arbitration step to guarantee high quality. These test set samples are meticulously chosen for semantic diversity, free from duplicates or near-duplicates, aiming to represent the broad spectrum of the sample space for a given task. Our intention is to ensure that our evaluation reflects the all the complexities of a given task, it's nuances thus instilling confidence in the ability our model to generalize across use cases for that given task.

To establish benchmarks, we will utilize the best performing open source model, namely Mixtral-MoE v0.1 [15] and closed source API-based LLM namely GPT-4. However, it's crucial to underscore that our primary goal in the RCA process is to surpass or at least match human experts, rather than being fixated on outperforming general-purpose LLMs. A practical consideration worth noting is that observability data is commonly classified as Personally Identifiable Information (PII) in nearly every enterprise organization. Consequently, comparing with API-based LLM products can be challenging, as they are inherently multi-tenant, potentially infringing upon enterprise data governance and compliance requirements. Furthermore, the open source LLMs we evaluate pose an additional challenge related to packaging and deployment in enterprise environments. These complexities underscore the need for a careful and nuanced evaluation approach that accounts for the specifics of the RCA process and the practical challenges associated with enterprise settings.

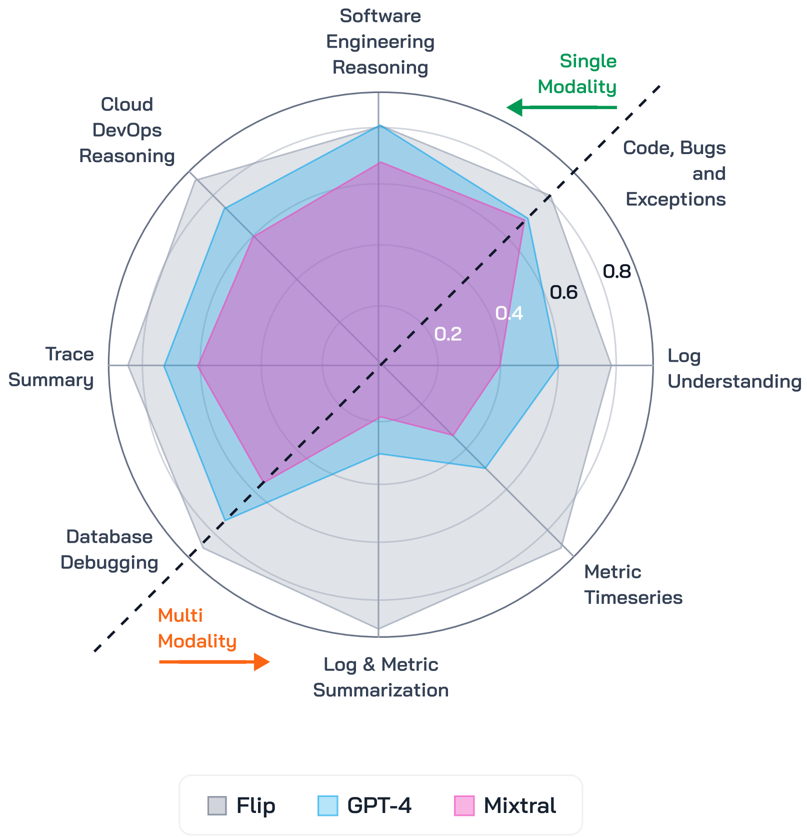


Figure 6: Flip DevOps LLM Performance Benchmark

7.1 Results

We evaluate the model across 30+ different tasks of utility that span all of the CoMELT modalities (code, metrics, events, logs and traces), complexity levels and task types. However, while reporting, we aggregate the granular tasks into higher level categories that reflect either the war room personas or the knowledge they encompass and present results below in spider chart 11. The categories are listed below with example tasks within each category, refer Appendix B for illustration of the tasks. All tasks in the benchmark are reported against expert human annotations with the majority of the test data being collected from real and diverse production environments. To ensure fairness in benchmarking, we take the best of 5 prompts and temperature settings for both Mixtral and GPT-4 baselines and report the numbers in our evaluation. As an additional note, these are the same prompts we use in our training as well, hence we believe that the prompts being used are of high quality and should not be a cause of underperformance in baselines. For each task, we use the best metric applicable; for example, for classification, we use F1 score; summarization, Q&A and reasoning related tasks, we use Rouge-2. For tasks such as SQL and code generation, we do compilation and execution check along with exact match scores; and for time series forecasting related tasks, we use mean absolute percentage error. For the convenience of the readers, we normalize all the scores to a scale of 0 to 1. For more details on the normalization, please refer to Appendix C. For tasks such as summarization that are prone to hallucinations, we do additional checks to capture factual correctness using automated entity-level factual consistency method[25] and human evaluations.

The Flip DevOps LLM consistently outperforms Mixtral by an average of 67.6% and surpasses GPT-4 by 34.5% across various tasks, as detailed in Table 1. There is a noticeable step-function improvement in performance, particularly on tasks related to Log Understanding, Log & Metric Summarization, Metric Timeseries, and Database Debugging. This observation strongly supports our assertion regarding the necessity of domain-specific training for effectively addressing challenges that require expert human insights. The Flip DevOps LLM attains a performance range between 0.77 to 0.95 compared to human expert accuracy. It is crucial to emphasize that while the baselines fail to have satisfactory performance on tasks that involve multiple pieces of evidence which we term as multi-modality. The multi-modality here is akin to code-mixing in languages where a mix of languages tell a coherent story. It's noteworthy that even in single-modality tasks, the performance of the baselines does not approach that achieved by the Flip DevOps LLM except for the tasks of Software Engineering Reasoning where GPT-4 is marginally better. Our training process at Flip AI ensures the absence of test set contamination. However, we are unable to verify whether GPT-4 or Mixtral have been trained on the test samples, particularly in tasks related to Software Engineering, Cloud DevOps Reasoning, and Code, Bugs, and Exceptions that are likely to be in the public domain. Additionally, it is essential to note that GPT-4 responses tend to be verbose and include unnecessary details, leading to higher recall for those outputs. In a preference study focused on Trace Summarization, Log & Metric Summarization, and Software Engineering Reasoning, expert annotators expressed a preference for the outputs from Flip DevOps LLM 65.7% of the time and tied with GPT-4 14.2% of the time. All of these results underscore the superiority of the Flip DevOps LLM over baseline models.

Table 1: Model Evaluation Performance Scores (normalized scale 0-1) for Categories

Category	**Flip**	**GPT-4**	**Mixtral**	**Sample Tasks**
Log Understanding	**0.77**	0.58	0.4	*Log Parsing, Classification, Generative Log NER*
Code, Bugs, Exceptions	**0.80**	0.71	0.68	*Code Explain, Exception Cause Reasoning*
Software Engineering Reasoning	0.81	**0.84**	0.68	*Software Engineering Q&A with Explanation*
Cloud DevOps Reasoning	**0.88**	0.76	0.6	*Cloud Certification, DevOps Scenario Reasoning*
Trace Summary	**0.85**	0.73	0.62	*Summarize Multiple Flame Graphs*
Database Debugging	**0.85**	0.7	0.56	*Database Performance, Slow Query Debugging*
Log & Metric Summarization	**0.95**	0.26	0.17	*Summarize RCA from Metric & Log evidence*
Metric Timeseries	**0.86**	0.45	0.33	*Summarize Cause of Spikes, Forecast Latency Spikes*

8 Discussion

Among the largest institutions in the world, all of which are deeply engaged in the development, maintenance, and management of extensive software systems, the imperative to ensure the robust health and optimal performance of these systems has emerged as a one of their most urgent priorities. Consequently, these institutions make substantial investments in both technical expertise and cutting-edge tools aimed at sustaining the well-being of their software ecosystems. It has become a norm to witness sizable teams comprising software developers, DevOps engineers, and site reliability engineers (SREs) operating within the precincts of every major enterprise. To cater to the needs of these subject matter experts, enterprises channel significant resources into observability and monitoring tools—platforms designed to accumulate and store telemetry data emanating from software applications and their underlying infrastructures. Given the intricate nature of large enterprises, it is increasingly commonplace to encounter a diverse array of observability platforms concurrently in use. In instances where undesired or anomalous behavior manifests within enterprise software systems, the resolution often hinges on the collaborative efforts between technical subject matter experts and the observability platforms. This collaboration is crucial for deciphering the root cause of issues and implementing corrective measures to restore the system to its desired state.

The tasks of building, maintaining, and managing software applications and infrastructure are no longer exclusive to technology companies. Instead, software has evolved to become as indispensable as human labor in the endeavors to construct and expand institutions of various kinds—whether they be large or small, in the private sector, governments, or beyond. And given the increasingly complex nature of software systems, things will go awry. We at Flip AI see a clear lane for domain specific LLMs (DLMs) to play a commanding role in the effort to help anyone and everyone who has ever answered that early morning page, or responded to that flurry of war room messages, or succumbed to a storm of system alerts; to provide for them the world's foremost set of technical experts whose sole job is to help them rapidly solve the Rube Goldberg-like puzzles of debugging production incidents. Given how urgent of a priority this is for any institution that is dependent on software for the smooth functioning of their business, Flip AI has found a high degree of resonance with our approach to solving this ubiquitous problem. Our next set of investments is to allow our customers the ability to customize the underlying LLMs using internal documentation and rich expertise they have built up in their enterprise environments.

9 Conclusion

With the advent of large language models, we see a clear application of this new technology to augment the technical subject matter experts by treating the telemetry data stored in observability systems as a semantic language. Flip AI DevOps LLM rapidly reasons through multiple modalities of CoMELT data (i.e. code, metrics, events, logs, traces), and synthesizes a narrative of what these languages are telling us about unwanted or anomalous system behavior. Because an LLM can interpret large sets of fractious, diverse data in a fraction of the time that a human could, this approach allows Flip AI to very successfully augment technical subject matter experts in debugging incidents and getting systems back to health much faster than before. It is worth noting that one of the key reasons why our approach has been successful is in our belief that a concerted, collaborative system of domain specific models (DLMs) can produce vastly superior results to a traditional, generalist large language model (LLM). This belief is validated by the results, not only with our customers, but also in the benchmarking discussed in this paper. Flip's system of intelligent actors that encompass the Flip DevOps LLM and observability integration are available today to be deployed in a customer's virtual private cloud, on-premises or as Software-as-a-Service (SaaS).

References

[1] Abdulaziz Alaboudi and Thomas D. Latoza. An exploratory study of debugging episodes. *ArXiv*, abs/2105.02162, 2021.

[2] Mikel Artetxe, Shruti Bhosale, Naman Goyal, Todor Mihaylov, Myle Ott, Sam Shleifer, Xi Victoria Lin, Jingfei Du, Srinivasan Iyer, Ramakanth Pasunuru, Giri Anantharaman, Xian Li, Shuohui Chen, Halil Akin, Mandeep Baines, Louis Martin, Xing Zhou, Punit Singh Koura, Brian O'Horo, Jeff Wang, Luke Zettlemoyer, Mona Diab, Zornitsa Kozareva, and Ves Stoyanov. Efficient large scale language modeling with mixtures of experts, 2022.

[3] Ron Artstein. Inter-annotator agreement. 2017.

[4] Monowar H. Bhuyan, Dhruba Kumar Bhattacharyya, and Jugal Kumar Kalita. Network anomaly detection: Methods, systems and tools. *IEEE Communications Surveys & Tutorials*, 16:303–336, 2014.

[5] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym, 2016.

[6] Yinfang Chen, Huaibing Xie, Minghua Ma, Yu Kang, Xin Gao, Liu Shi, Yunjie Cao, Xuedong Gao, Hao Fan, Ming Wen, Jun Zeng, Supriyo Ghosh, Xuchao Zhang, Chaoyun Zhang, Qingwei Lin, Saravan Rajmohan, Dongmei Zhang, and Tianyin Xu. Automatic root cause analysis via large language models for cloud incidents, 2023.

[7] Zhuangbin Chen, Jinyang Liu, Wenwei Gu, Yuxin Su, and Michael R. Lyu. Experience report: Deep learning-based system log analysis for anomaly detection, 2022.

[8] Hyung Won Chung, Le Hou, Shayne Longpre, Barret Zoph, Yi Tay, William Fedus, Yunxuan Li, Xuezhi Wang, Mostafa Dehghani, Siddhartha Brahma, Albert Webson, Shixiang Shane Gu, Zhuyun Dai, Mirac Suzgun, Xinyun Chen, Aakanksha Chowdhery, Alex Castro-Ros, Marie Pellat, Kevin Robinson, Dasha Valter, Sharan Narang, Gaurav Mishra, Adams Yu, Vincent Zhao, Yanping Huang, Andrew Dai, Hongkun Yu, Slav Petrov, Ed H. Chi, Jeff Dean, Jacob Devlin, Adam Roberts, Denny Zhou, Quoc V. Le, and Jason Wei. Scaling instruction-finetuned language models, 2022.

[9] Min Du, Feifei Li, Guineng Zheng, and Vivek Srikumar. Deeplog: Anomaly detection and diagnosis from system logs through deep learning. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, CCS '17, page 1285–1298, New York, NY, USA, 2017. Association for Computing Machinery.

[10] William Fedus, Barret Zoph, and Noam Shazeer. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity, 2022.

[11] Supriyo Ghosh, Manish Shetty, Chetan Bansal, and Suman Nath. How to fight production incidents? an empirical study on a large-scale cloud service. In *Proceedings of the 13th Symposium on Cloud Computing*, SoCC '22, page 126–141, New York, NY, USA, 2022. Association for Computing Machinery.

[12] Hongcheng Guo, Jian Yang, Jiaheng Liu, Liqun Yang, Linzheng Chai, Jiaqi Bai, Junran Peng, Xiaorong Hu, Chao Chen, Dongfeng Zhang, Xu Shi, Tieqiao Zheng, Liangfan Zheng, Bo Zhang, Ke Xu, and Zhoujun Li. Owl: A large language model for it operations, 2023.

[13] Shilin He, Pinjia He, Zhuangbin Chen, Tianyi Yang, Yuxin Su, and Michael R. Lyu. A survey on automated log analysis for reliability engineering. *ACM Comput. Surv.*, 54(6), jul 2021.

[14] Mojan Javaheripi and Sébastien Bubeck. Phi-2: The surprising power of small language models, 2023.

[15] Albert Q. Jiang, Alexandre Sablayrolles, Antoine Roux, Arthur Mensch, Blanche Savary, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Emma Bou Hanna, Florian Bressand, Gianna Lengyel, Guillaume Bour, Guillaume Lample, Lélio Renard Lavaud, Lucile Saulnier, Marie-Anne Lachaux, Pierre Stock, Sandeep Subramanian, Sophia Yang, Szymon Antoniak, Teven Le Scao, Théophile Gervet, Thibaut Lavril, Thomas Wang, Timothée Lacroix, and William El Sayed. Mixtral of experts, 2024.

[16] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. Retrieval-augmented generation for knowledge-intensive nlp tasks, 2021.

[17] Yilun Liu, Shimin Tao, Weibin Meng, Jingyu Wang, Wenbing Ma, Yanqing Zhao, Yuhang Chen, Hao Yang, Yanfei Jiang, and Xun Chen. Logprompt: Prompt engineering towards zero-shot and interpretable log analysis, 2023.

[18] Shayne Longpre, Le Hou, Tu Vu, Albert Webson, Hyung Won Chung, Yi Tay, Denny Zhou, Quoc V. Le, Barret Zoph, Jason Wei, and Adam Roberts. The flan collection: Designing data and methods for effective instruction tuning, 2023.

[19] Pan Lu, Baolin Peng, Hao Cheng, Michel Galley, Kai-Wei Chang, Ying Nian Wu, Song-Chun Zhu, and Jianfeng Gao. Chameleon: Plug-and-play compositional reasoning with large language models, 2023.

[20] Shuai Lu, Daya Guo, Shuo Ren, Junjie Huang, Alexey Svyatkovskiy, Ambrosio Blanco, Colin Clement, Dawn Drain, Daxin Jiang, Duyu Tang, Ge Li, Lidong Zhou, Linjun Shou, Long Zhou, Michele Tufano, Ming Gong, Ming Zhou, Nan Duan, Neel Sundaresan, Shao Kun Deng, Shengyu Fu, and Shujie Liu. Codexglue: A machine learning benchmark dataset for code understanding and generation, 2021.

[21] Shutian Luo, Huanle Xu, Chengzhi Lu, Kejiang Ye, Guoyao Xu, Liping Zhang, Yu Ding, Jian He, and Chengzhong Xu. Characterizing microservice dependency and performance: Alibaba trace analysis. In *Proceedings of the ACM Symposium on Cloud Computing*, SoCC '21, page 412–426, New York, NY, USA, 2021. Association for Computing Machinery.

[22] Ziyang Luo, Can Xu, Pu Zhao, Qingfeng Sun, Xiubo Geng, Wenxiang Hu, Chongyang Tao, Jing Ma, Qingwei Lin, and Daxin Jiang. Wizardcoder: Empowering code large language models with evol-instruct, 2023.

[23] Syed Akbar Mehdi, Junaid Khalid, and Syed Ali Khayam. Revisiting traffic anomaly detection using software defined networking. In Robin Sommer, Davide Balzarotti, and Gregor Maier, editors, *Recent Advances in Intrusion Detection*, pages 161–180, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

[24] Vijayaraghavan Murali, Edward Yao, Umang Mathur, and Satish Chandra. Scalable statistical root cause analysis on app telemetry, 2021.

[25] Feng Nan, Ramesh Nallapati, Zhiguo Wang, Cicero Nogueira dos Santos, Henghui Zhu, Dejiao Zhang, Kathleen McKeown, and Bing Xiang. Entity-level factual consistency of abstractive text summarization. In Paola Merlo, Jorg Tiedemann, and Reut Tsarfaty, editors, *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume*, pages 2727–2733, Online, April 2021. Association for Computational Linguistics.

[26] Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach themselves to use tools, 2023.

[27] Manish Shetty, Chetan Bansal, Sai Pramod Upadhyayula, Arjun Radhakrishna, and Anurag Gupta. Autotsg: Learning and synthesis for incident troubleshooting. In *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ESEC/FSE 2022, page 1477–1488, New York, NY, USA, 2022. Association for Computing Machinery.

[28] Weiwei Sun, Zhengliang Shi, Shen Gao, Pengjie Ren, Maarten de Rijke, and Zhaochun Ren. Contrastive learning reduces hallucination in conversations, 2022.

[29] Yi Tay, Mostafa Dehghani, Vinh Q. Tran, Xavier Garcia, Jason Wei, Xuezhi Wang, Hyung Won Chung, Siamak Shakeri, Dara Bahri, Tal Schuster, Huaixiu Steven Zheng, Denny Zhou, Neil Houlsby, and Donald Metzler. Ul2: Unifying language learning paradigms, 2023.

[30] Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Thomas L. Griffiths, Yuan Cao, and Karthik Narasimhan. Tree of thoughts: Deliberate problem solving with large language models, 2023.

[31] Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Li, Qingning Yao, Shanelle Roman, Zilin Zhang, and Dragomir Radev. Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-sql task, 2018.

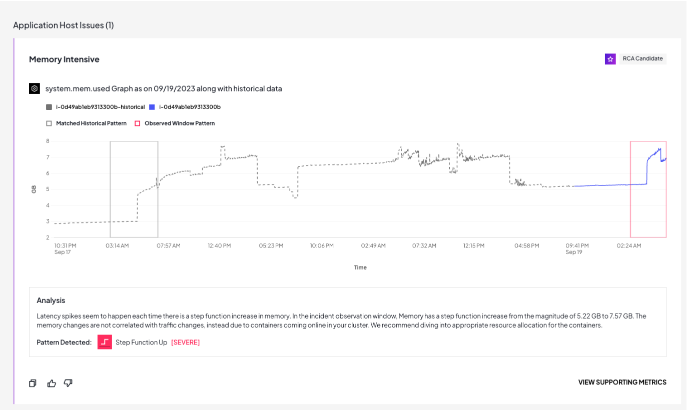
[32] Xuanhe Zhou, Guoliang Li, and Zhiyuan Liu. Llm as dba. 2023.

A Appendix: Sample RCA Reports

In this section, we share few incident RCA reports that Flip AI has debugged in production. All incidents have been anonymized to protect customer data. The list is not structured to be exhaustive, instead intended to give the reader a flavor of Flip AI's capabilities. It is noteworthy that, On average Flip AI takes under 60 seconds to debug these incidents.

A.1 Memory Exhaustion

Incident: P90 latency on service has increased beyond the threshold of 5 seconds.



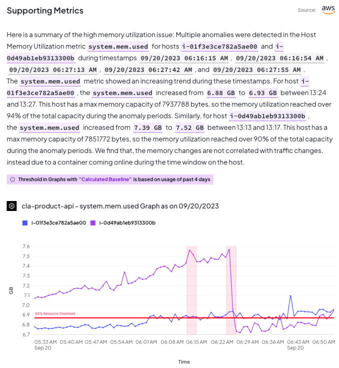
system.mem.used Graph as on 09/19/2023 along with historical data

Analysis

Latency spikes seem to happen each time there is a step function increase in memory. In the incident observation window, Memory has a step function increase from the magnitude of 5.22 GB to 7.57 GB. The memory changes are not correlated with traffic changes, instead due to containers coming online in your cluster. We recommend diving into appropriate resource allocation for the containers.

Pattern Detected: Step Function Up [SEVERE]

(a) Reasoning Memory Spikes by Flip AI



Here is a summary of the high memory utilization issue: Multiple anomalies were detected in the Host Memory Utilization metric `system.mem.used` for hosts `i-01f3e3ce782a5ae00` and `i-0d49ab1eb9313300b` during timestamps `09/20/2023 06:16:15 AM`, `09/20/2023 06:16:54 AM`, `09/20/2023 06:27:13 AM`, `09/20/2023 06:27:42 AM`, and `09/20/2023 06:27:55 AM`. The `system.mem.used` metric showed an increasing trend during these timestamps. For host `i-01f3e3ce782a5ae00`, the `system.mem.used` increased from `6.88 GB` to `6.93 GB` between 13:24 and 13:27. This host has a max memory capacity of 7937788 bytes, so the memory utilization reached over 94% of the total capacity during the anomaly periods. Similarly, for host `i-0d49ab1eb9313300b`, the `system.mem.used` increased from `7.39 GB` to `7.52 GB` between 13:13 and 13:17. This host has a max memory capacity of 7851772 bytes, so the memory utilization reached over 90% of the total capacity during the anomaly periods. We find that, the memory changes are not correlated with traffic changes, instead due to a container coming online during the time window on the host.

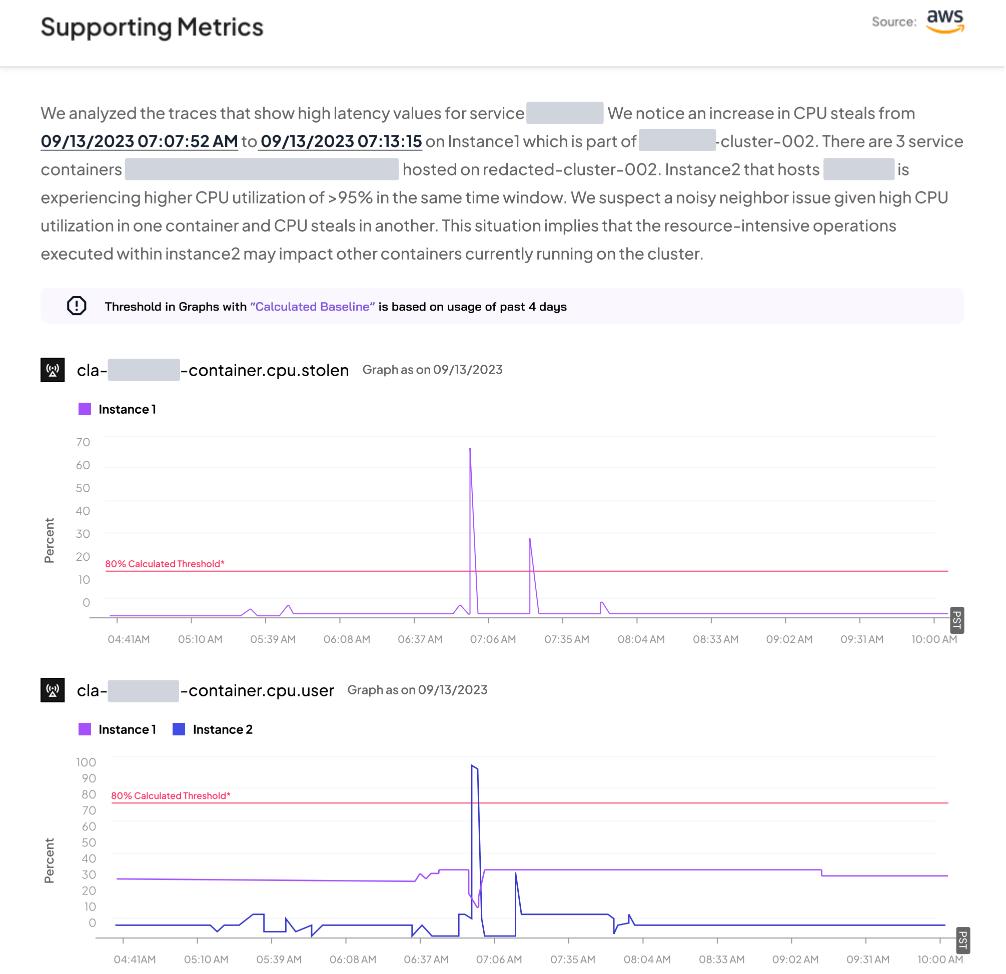
Threshold in Graphs with "Calculated Baseline" is based on usage of past 4 days

cla-product-api - system.mem.used Graph as on 09/20/2023

(b) Supporting Evidence

A.2 Insufficient Resource Allocation

Incident: Service is experiencing latency issues occasionally due to poor resource allocation in Kubernetes



Supporting Metrics

Source: aws

We analyzed the traces that show high latency values for service [redacted] We notice an increase in CPU steals from **09/13/2023 07:07:52 AM** to **09/13/2023 07:13:15** on Instance1 which is part of [redacted]-cluster-002. There are 3 service containers [redacted] hosted on redacted-cluster-002. Instance2 that hosts [redacted] is experiencing higher CPU utilization of >95% in the same time window. We suspect a noisy neighbor issue given high CPU utilization in one container and CPU steals in another. This situation implies that the resource-intensive operations executed within instance2 may impact other containers currently running on the cluster.

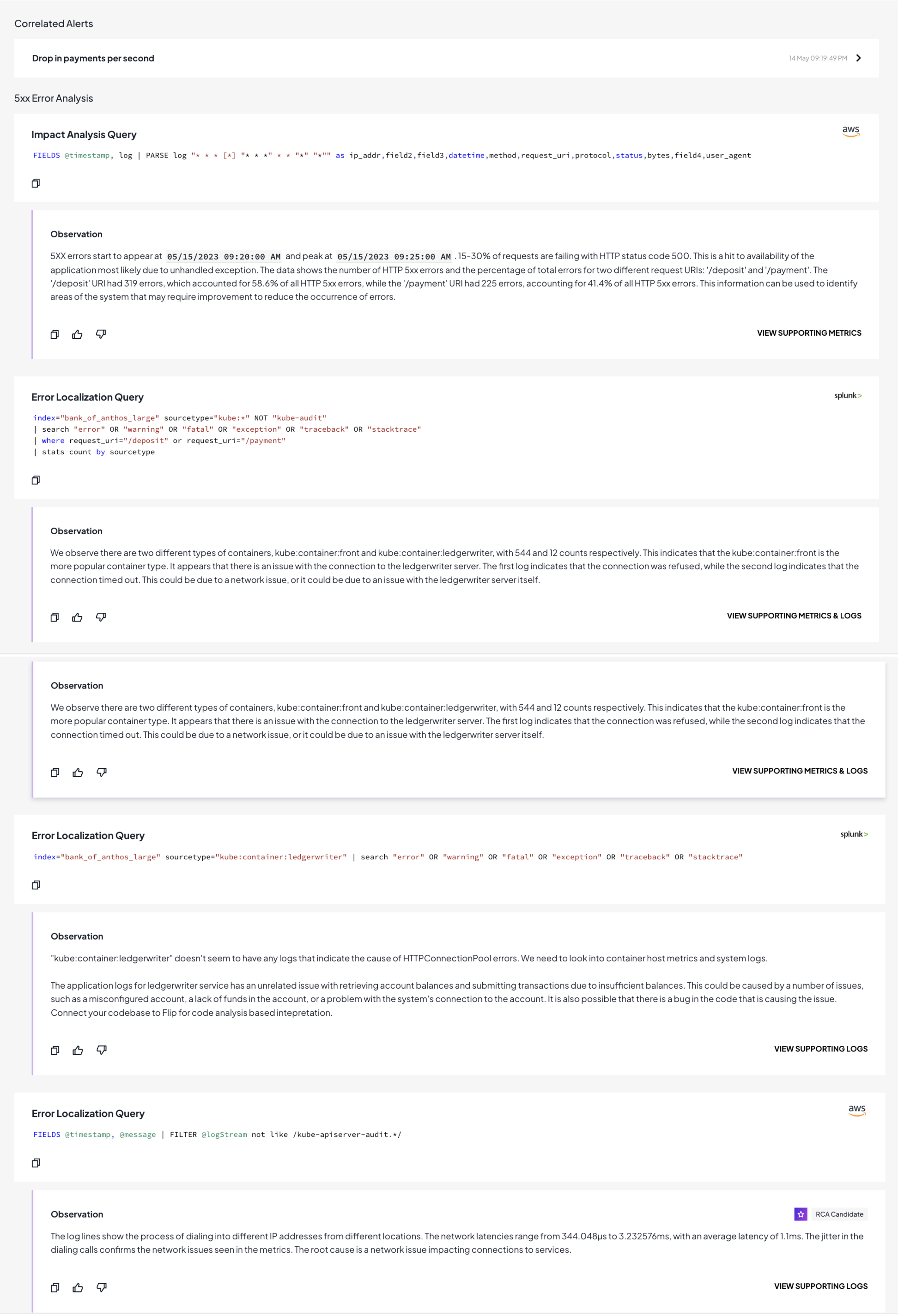
ⓘ Threshold in Graphs with "Calculated Baseline" is based on usage of past 4 days

cla-[redacted]-container.cpu.stolen Graph as on 09/13/2023

cla-[redacted]-container.cpu.user Graph as on 09/13/2023

A.3 HTTP Error Debugging

Incident: Intermittent 5xx Errors in Access Logs



Correlated Alerts

Drop in payments per second

5xx Error Analysis

Impact Analysis Query

```
FIELDS @timestamp, log | PARSE log "* * * [*] "* * *" * * "*" "*"" as ip_addr,field2,field3,datetime,method,request_uri,protocol,status,bytes,field4,user_agent
```

Observation

5XX errors start to appear at `05/15/2023 09:20:00 AM` and peak at `05/15/2023 09:25:00 AM` . 15-30% of requests are failing with HTTP status code 500. This is a hit to availability of the application most likely due to unhandled exception. The data shows the number of HTTP 5xx errors and the percentage of total errors for two different request URIs: '/deposit' and '/payment'. The '/deposit' URI had 319 errors, which accounted for 58.6% of all HTTP 5xx errors, while the '/payment' URI had 225 errors, accounting for 41.4% of all HTTP 5xx errors. This information can be used to identify areas of the system that may require improvement to reduce the occurrence of errors.

Error Localization Query

```
index="bank_of_anthos_large" sourcetype="kube:*" NOT "kube-audit"
| search "error" OR "warning" OR "fatal" OR "exception" OR "traceback" OR "stacktrace"
| where request_uri="/deposit" or request_uri="/payment"
| stats count by sourcetype
```

Observation

We observe there are two different types of containers, kube:container:front and kube:container:ledgerwriter, with 544 and 12 counts respectively. This indicates that the kube:container:front is the more popular container type. It appears that there is an issue with the connection to the ledgerwriter server. The first log indicates that the connection was refused, while the second log indicates that the connection timed out. This could be due to a network issue, or it could be due to an issue with the ledgerwriter server itself.

Observation

We observe there are two different types of containers, kube:container:front and kube:container:ledgerwriter, with 544 and 12 counts respectively. This indicates that the kube:container:front is the more popular container type. It appears that there is an issue with the connection to the ledgerwriter server. The first log indicates that the connection was refused, while the second log indicates that the connection timed out. This could be due to a network issue, or it could be due to an issue with the ledgerwriter server itself.

Error Localization Query

```
index="bank_of_anthos_large" sourcetype="kube:container:ledgerwriter" | search "error" OR "warning" OR "fatal" OR "exception" OR "traceback" OR "stacktrace"
```

Observation

"kube:container:ledgerwriter" doesn't seem to have any logs that indicate the cause of HTTPConnectionPool errors. We need to look into container host metrics and system logs.

The application logs for ledgerwriter service has an unrelated issue with retrieving account balances and submitting transactions due to insufficient balances. This could be caused by a number of issues, such as a misconfigured account, a lack of funds in the account, or a problem with the system's connection to the account. It is also possible that there is a bug in the code that is causing the issue. Connect your codebase to Flip for code analysis based intepretation.

Error Localization Query

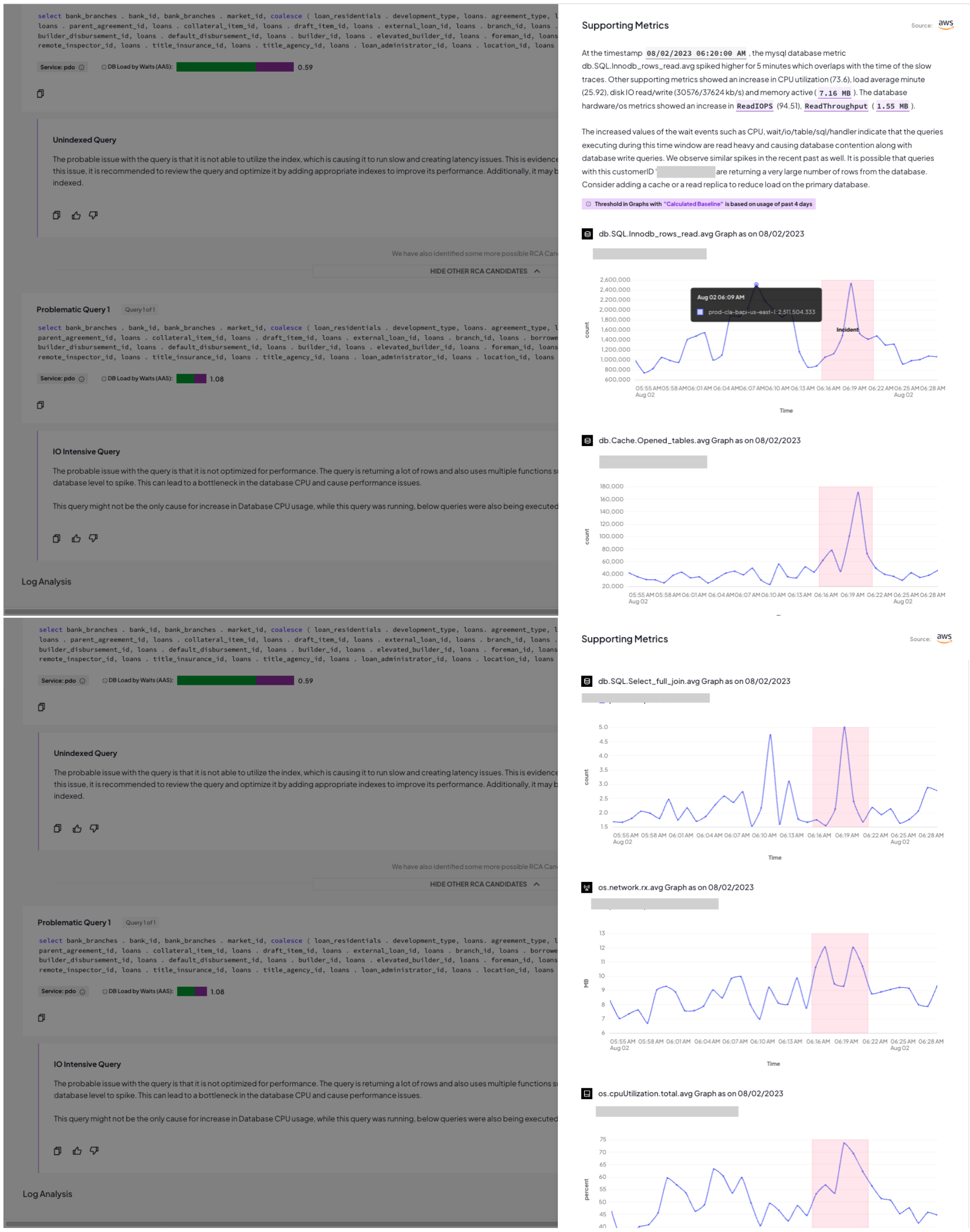
```
FIELDS @timestamp, @message | FILTER @logStream not like /kube-apiserver-audit.*/
```

Observation

The log lines show the process of dialing into different IP addresses from different locations. The network latencies range from 344.048μs to 3.232576ms, with an average latency of 1.1ms. The jitter in the dialing calls confirms the network issues seen in the metrics. The root cause is a network issue impacting connections to services.

A.4 Database Debugging

Incident: Database is experiencing performance issues



```
select bank_branches . bank_id, bank_branches . market_id, coalesce ( loan_residentials . development_type, loans. agreement_type, l
loans . parent_agreement_id, loans . collateral_item_id, loans . draft_item_id, loans . external_loan_id, loans . branch_id, loans .
builder_disbursement_id, loans . default_disbursement_id, loans . builder_id, loans . elevated_builder_id, loans . foreman_id, loans
remote_inspector_id, loans . title_insurance_id, loans . title_agency_id, loans . loan_administrator_id, loans . location_id, loans
```

Service: pdo ⓘ DB Load by Waits (AAS): 0.59

Unindexed Query

The probable issue with the query is that it is not able to utilize the index, which is causing it to run slow and creating latency issues. This is evidence
this issue, it is recommended to review the query and optimize it by adding appropriate indexes to improve its performance. Additionally, it may b
indexed.

We have also identified some more possible RCA Can

Problematic Query 1 Query 1 of 1

```
select bank_branches . bank_id, bank_branches . market_id, coalesce ( loan_residentials . development_type, loans. agreement_type, l
parent_agreement_id, loans . collateral_item_id, loans . draft_item_id, loans . external_loan_id, loans . branch_id, loans . borrowe
builder_disbursement_id, loans . default_disbursement_id, loans . builder_id, loans . elevated_builder_id, loans . foreman_id, loans
remote_inspector_id, loans . title_insurance_id, loans . title_agency_id, loans . loan_administrator_id, loans . location_id, loans
```

Service: pdo ⓘ DB Load by Waits (AAS): 1.08

IO Intensive Query

The probable issue with the query is that it is not optimized for performance. The query is returning a lot of rows and also uses multiple functions s
database level to spike. This can lead to a bottleneck in the database CPU and cause performance issues.

This query might not be the only cause for increase in Database CPU usage, while this query was running, below queries were also being executed

Log Analysis

```
select bank_branches . bank_id, bank_branches . market_id, coalesce ( loan_residentials . development_type, loans. agreement_type, l
loans . parent_agreement_id, loans . collateral_item_id, loans . draft_item_id, loans . external_loan_id, loans . branch_id, loans .
builder_disbursement_id, loans . default_disbursement_id, loans . builder_id, loans . elevated_builder_id, loans . foreman_id, loans
remote_inspector_id, loans . title_insurance_id, loans . title_agency_id, loans . loan_administrator_id, loans . location_id, loans
```

Service: pdo ⓘ DB Load by Waits (AAS): 0.59

Unindexed Query

The probable issue with the query is that it is not able to utilize the index, which is causing it to run slow and creating latency issues. This is evidence
this issue, it is recommended to review the query and optimize it by adding appropriate indexes to improve its performance. Additionally, it may b
indexed.

We have also identified some more possible RCA Can

HIDE OTHER RCA CANDIDATES ^

Problematic Query 1 Query 1 of 1

```
select bank_branches . bank_id, bank_branches . market_id, coalesce ( loan_residentials . development_type, loans. agreement_type, l
parent_agreement_id, loans . collateral_item_id, loans . draft_item_id, loans . external_loan_id, loans . branch_id, loans . borrowe
builder_disbursement_id, loans . default_disbursement_id, loans . builder_id, loans . elevated_builder_id, loans . foreman_id, loans
remote_inspector_id, loans . title_insurance_id, loans . title_agency_id, loans . loan_administrator_id, loans . location_id, loans
```

Service: pdo ⓘ DB Load by Waits (AAS): 1.08

IO Intensive Query

The probable issue with the query is that it is not optimized for performance. The query is returning a lot of rows and also uses multiple functions s
database level to spike. This can lead to a bottleneck in the database CPU and cause performance issues.

This query might not be the only cause for increase in Database CPU usage, while this query was running, below queries were also being executed

Log Analysis

Supporting Metrics Source: aws

At the timestamp `08/02/2023 06:20:00 AM`, the mysql database metric db.SQL.Innodb_rows_read.avg spiked higher for 5 minutes which overlaps with the time of the slow traces. Other supporting metrics showed an increase in CPU utilization (73.6), load average minute (25.92), disk IO read/write (30576/37624 kb/s) and memory active (`7.16 MB`). The database hardware/os metrics showed an increase in `ReadIOPS` (94.51), `ReadThroughput` (`1.55 MB`).

The increased values of the wait events such as CPU, wait/io/table/sql/handler indicate that the queries executing during this time window are read heavy and causing database contention along with database write queries. We observe similar spikes in the recent past as well. It is possible that queries with this customerID '[redacted] are returning a very large number of rows from the database. Consider adding a cache or a read replica to reduce load on the primary database.

ⓘ Threshold in Graphs with "Calculated Baseline" is based on usage of past 4 days

db.SQL.Innodb_rows_read.avg Graph as on 08/02/2023

db.Cache.Opened_tables.avg Graph as on 08/02/2023

Supporting Metrics Source: aws

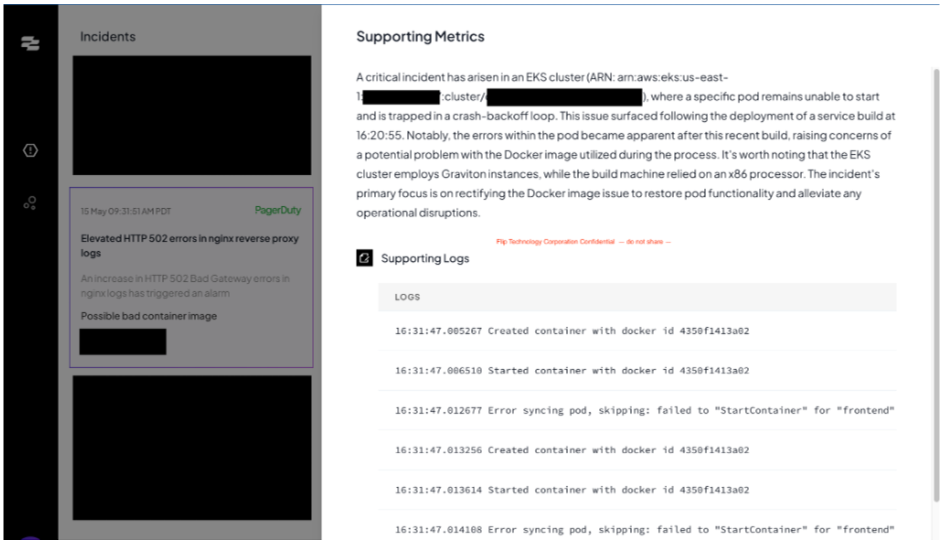
db.SQL.Select_full_join.avg Graph as on 08/02/2023

os.network.rx.avg Graph as on 08/02/2023

os.cpuUtilization.total.avg Graph as on 08/02/2023

A.5 Configuration Issue

Incident: Increase in HTTP 502 errors from load balancer for a service deployed to Kubernetes



Incidents

[redacted]

15 May 09:31:51 AM PDT

PagerDuty

Elevated HTTP 502 errors in nginx reverse proxy logs

An increase in HTTP 502 Bad Gateway errors in nginx logs has triggered an alarm

Possible bad container image

[redacted]

[redacted]

Supporting Metrics

A critical incident has arisen in an EKS cluster (ARN: arn:aws:eks:us-east-1[redacted]:cluster/[redacted]), where a specific pod remains unable to start and is trapped in a crash-backoff loop. This issue surfaced following the deployment of a service build at 16:20:55. Notably, the errors within the pod became apparent after this recent build, raising concerns of a potential problem with the Docker image utilized during the process. It's worth noting that the EKS cluster employs Graviton instances, while the build machine relied on an x86 processor. The incident's primary focus is on rectifying the Docker image issue to restore pod functionality and alleviate any operational disruptions.

Flip Technology Corporation Confidential — do not share —

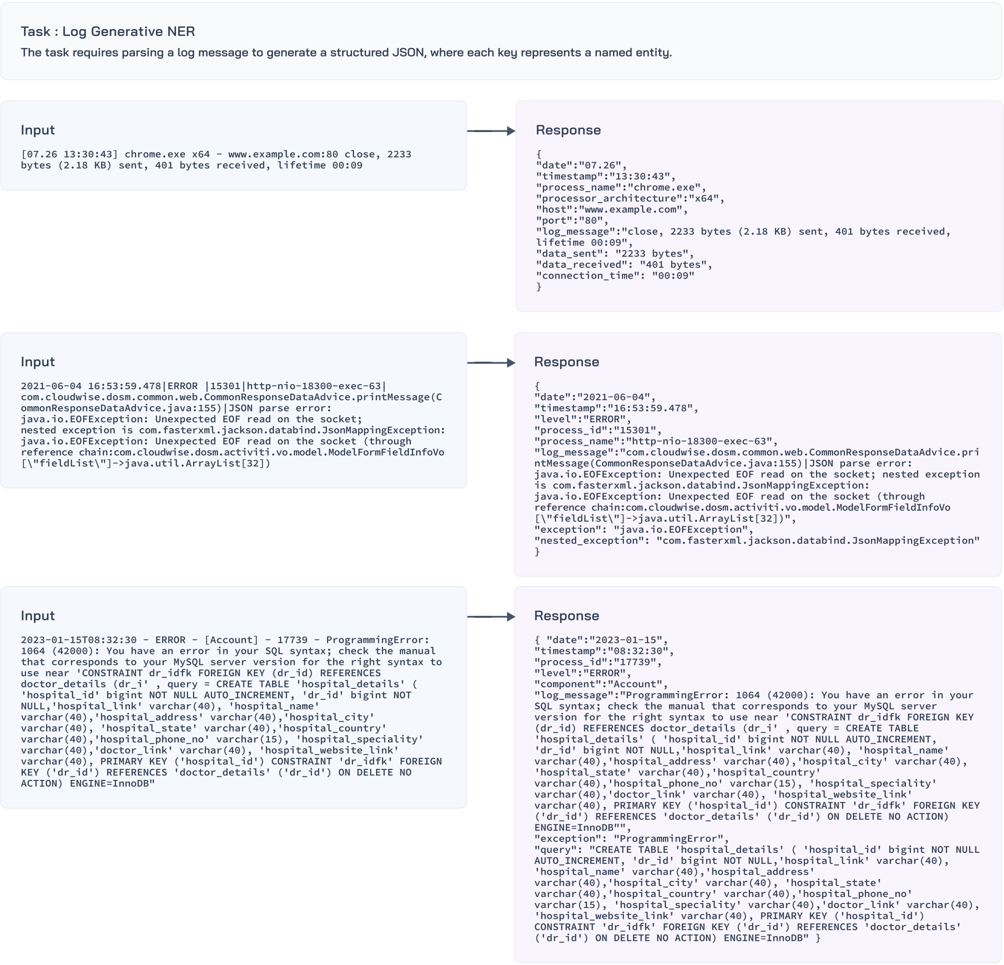
Supporting Logs

LOGS

```
16:31:47.005267 Created container with docker id 4350f1413a02

16:31:47.006510 Started container with docker id 4350f1413a02

16:31:47.012677 Error syncing pod, skipping: failed to "StartContainer" for "frontend"

16:31:47.013256 Created container with docker id 4350f1413a02

16:31:47.013614 Started container with docker id 4350f1413a02

16:31:47.014108 Error syncing pod, skipping: failed to "StartContainer" for "frontend"
```

B Appendix: Task Definition and Examples

B.1 Log Understanding



Task : Log Generative NER

The task requires parsing a log message to generate a structured JSON, where each key represents a named entity.

Input

```
[07.26 13:30:43] chrome.exe x64 - www.example.com:80 close, 2233
bytes (2.18 KB) sent, 401 bytes received, lifetime 00:09
```

Response

```
{
"date":"07.26",
"timestamp":"13:30:43",
"process_name":"chrome.exe",
"processor_architecture":"x64",
"host":"www.example.com",
"port":"80",
"log_message":"close, 2233 bytes (2.18 KB) sent, 401 bytes received,
lifetime 00:09",
"data_sent": "2233 bytes",
"data_received": "401 bytes",
"connection_time": "00:09"
}
```

Input

```
2021-06-04 16:53:59.478|ERROR |15301|http-nio-18300-exec-63|
com.cloudwise.dosm.common.web.CommonResponseDataAdvice.printMessage(C
ommonResponseDataAdvice.java:155)|JSON parse error:
java.io.EOFException: Unexpected EOF read on the socket;
nested exception is com.fasterxml.jackson.databind.JsonMappingException:
java.io.EOFException: Unexpected EOF read on the socket (through
reference chain:com.cloudwise.dosm.activiti.vo.model.ModelFormFieldInfoVo
[\"fieldList\"]->java.util.ArrayList[32])
```

Response

```
{
"date":"2021-06-04",
"timestamp":"16:53:59.478",
"level":"ERROR",
"process_id":"15301",
"process_name":"http-nio-18300-exec-63",
"log_message":"com.cloudwise.dosm.common.web.CommonResponseDataAdvice.pri
ntMessage(CommonResponseDataAdvice.java:155)|JSON parse error:
java.io.EOFException: Unexpected EOF read on the socket; nested exception
is com.fasterxml.jackson.databind.JsonMappingException:
java.io.EOFException: Unexpected EOF read on the socket (through
reference chain:com.cloudwise.dosm.activiti.vo.model.ModelFormFieldInfoVo
[\"fieldList\"]->java.util.ArrayList[32])",
"exception": "java.io.EOFException",
"nested_exception": "com.fasterxml.jackson.databind.JsonMappingException"
}
```

Input

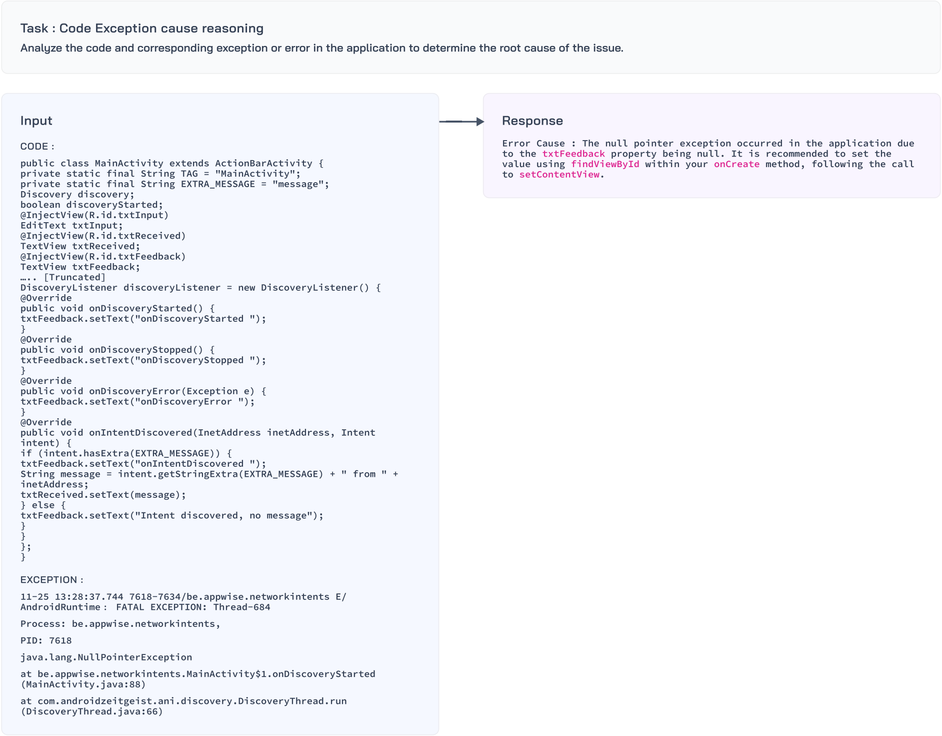
```
2023-01-15T08:32:30 - ERROR - [Account] - 17739 - ProgrammingError:
1064 (42000): You have an error in your SQL syntax; check the manual
that corresponds to your MySQL server version for the right syntax to
use near 'CONSTRAINT dr_idfk FOREIGN KEY (dr_id) REFERENCES
doctor_details (dr_i' , query = CREATE TABLE 'hospital_details' (
'hospital_id' bigint NOT NULL AUTO_INCREMENT, 'dr_id' bigint NOT
NULL,'hospital_link' varchar(40), 'hospital_name'
varchar(40),'hospital_address' varchar(40),'hospital_city'
varchar(40), 'hospital_state' varchar(40),'hospital_country'
varchar(40),'hospital_phone_no' varchar(15), 'hospital_speciality'
varchar(40),'doctor_link' varchar(40), 'hospital_website_link'
varchar(40), PRIMARY KEY ('hospital_id') CONSTRAINT 'dr_idfk' FOREIGN
KEY ('dr_id') REFERENCES 'doctor_details' ('dr_id') ON DELETE NO
ACTION) ENGINE=InnoDB"
```

Response

```
{ "date":"2023-01-15",
"timestamp":"08:32:30",
"process_id":"17739",
"level":"ERROR",
"component":"Account",
"log_message":"ProgrammingError: 1064 (42000): You have an error in your
SQL syntax; check the manual that corresponds to your MySQL server
version for the right syntax to use near 'CONSTRAINT dr_idfk FOREIGN KEY
(dr_id) REFERENCES doctor_details (dr_i' , query = CREATE TABLE
'hospital_details' ( 'hospital_id' bigint NOT NULL AUTO_INCREMENT,
'dr_id' bigint NOT NULL,'hospital_link' varchar(40), 'hospital_name'
varchar(40),'hospital_address' varchar(40),'hospital_city' varchar(40),
'hospital_state' varchar(40),'hospital_country'
varchar(40),'hospital_phone_no' varchar(15), 'hospital_speciality'
varchar(40),'doctor_link' varchar(40), 'hospital_website_link'
varchar(40), PRIMARY KEY ('hospital_id') CONSTRAINT 'dr_idfk' FOREIGN KEY
('dr_id') REFERENCES 'doctor_details' ('dr_id') ON DELETE NO ACTION)
ENGINE=InnoDB"",
"exception": "ProgrammingError",
"query": "CREATE TABLE 'hospital_details' ( 'hospital_id' bigint NOT NULL
AUTO_INCREMENT, 'dr_id' bigint NOT NULL,'hospital_link' varchar(40),
'hospital_name' varchar(40),'hospital_address'
varchar(40),'hospital_city' varchar(40), 'hospital_state'
varchar(40),'hospital_country' varchar(40),'hospital_phone_no'
varchar(15), 'hospital_speciality' varchar(40),'doctor_link' varchar(40),
'hospital_website_link' varchar(40), PRIMARY KEY ('hospital_id')
CONSTRAINT 'dr_idfk' FOREIGN KEY ('dr_id') REFERENCES 'doctor_details'
('dr_id') ON DELETE NO ACTION) ENGINE=InnoDB" }
```

Figure 8: Log Understanding

B.2 Code, Bugs and Exceptions



Task : Code Exception cause reasoning

Analyze the code and corresponding exception or error in the application to determine the root cause of the issue.

Input

CODE :

```
public class MainActivity extends ActionBarActivity {
private static final String TAG = "MainActivity";
private static final String EXTRA_MESSAGE = "message";
Discovery discovery;
boolean discoveryStarted;
@InjectView(R.id.txtInput)
EditText txtInput;
@InjectView(R.id.txtReceived)
TextView txtReceived;
@InjectView(R.id.txtFeedback)
TextView txtFeedback;
….. [Truncated]
DiscoveryListener discoveryListener = new DiscoveryListener() {
@Override
public void onDiscoveryStarted() {
txtFeedback.setText("onDiscoveryStarted ");
}
@Override
public void onDiscoveryStopped() {
txtFeedback.setText("onDiscoveryStopped ");
}
@Override
public void onDiscoveryError(Exception e) {
txtFeedback.setText("onDiscoveryError ");
}
@Override
public void onIntentDiscovered(InetAddress inetAddress, Intent
intent) {
if (intent.hasExtra(EXTRA_MESSAGE)) {
txtFeedback.setText("onIntentDiscovered ");
String message = intent.getStringExtra(EXTRA_MESSAGE) + " from " +
inetAddress;
txtReceived.setText(message);
} else {
txtFeedback.setText("Intent discovered, no message");
}
}
};
}
```

EXCEPTION :

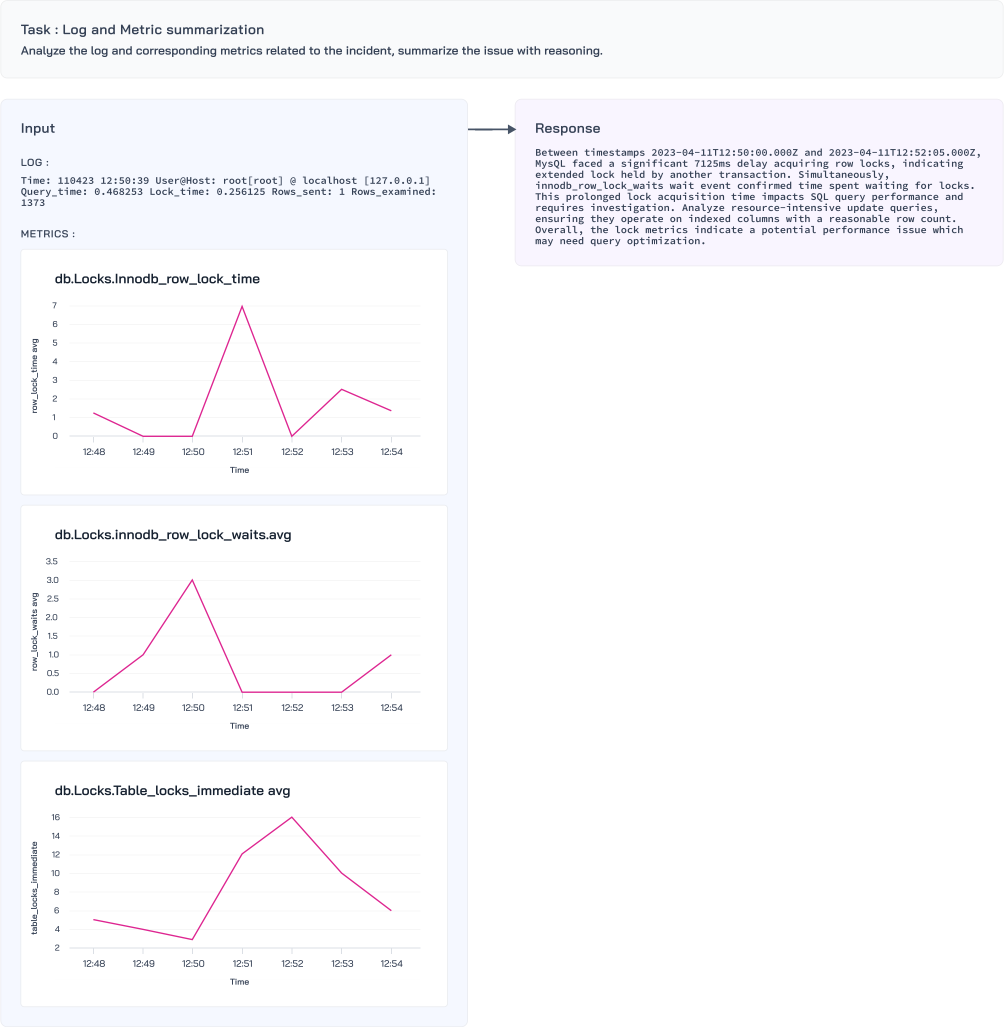
```
11-25 13:28:37.744 7618-7634/be.appwise.networkintents E/
AndroidRuntime :  FATAL EXCEPTION: Thread-684

Process: be.appwise.networkintents,

PID: 7618

java.lang.NullPointerException

at be.appwise.networkintents.MainActivity$1.onDiscoveryStarted
(MainActivity.java:88)

at com.androidzeitgeist.ani.discovery.DiscoveryThread.run
(DiscoveryThread.java:66)
```

Response

Error Cause : The null pointer exception occurred in the application due to the `txtFeedback` property being null. It is recommended to set the value using `findViewById` within your `onCreate` method, following the call to `setContentView`.

Figure 9: Code Exception Reasoning

B.3 Log and Metric Summarization



Task : Log and Metric summarization

Analyze the log and corresponding metrics related to the incident, summarize the issue with reasoning.

Input

LOG :

```
Time: 110423 12:50:39 User@Host: root[root] @ localhost [127.0.0.1]
Query_time: 0.468253 Lock_time: 0.256125 Rows_sent: 1 Rows_examined:
1373
```

METRICS :

Response

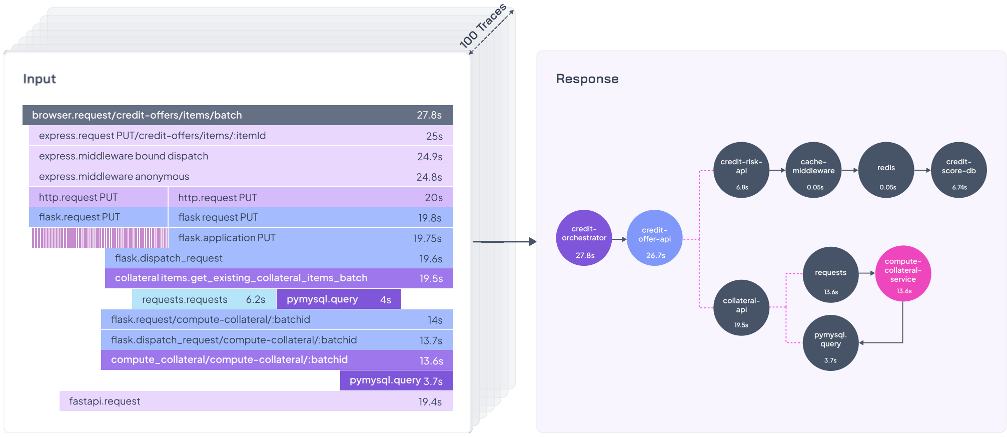
```
Between timestamps 2023-04-11T12:50:00.000Z and 2023-04-11T12:52:05.000Z,
MysQL faced a significant 7125ms delay acquiring row locks, indicating
extended lock held by another transaction. Simultaneously,
innodb_row_lock_waits wait event confirmed time spent waiting for locks.
This prolonged lock acquisition time impacts SQL query performance and
requires investigation. Analyze resource-intensive update queries,
ensuring they operate on indexed columns with a reasonable row count.
Overall, the lock metrics indicate a potential performance issue which
may need query optimization.
```

Figure 10: Multi-Modal Reasoning with Logs and Metric Timeseries

B.4 Trace Summary

Task : Trace summary

The task is to analyze series of flame graphs then summarize the interaction among different services. The goal is to identify and summarize the relationships between these services, pinpoint pathways with high latency that adversely impact overall performance.



100 Traces
Input
browser.request/credit-offers/items/batch 27.8s
express.request PUT/credit-offers/items/:itemId 25s
express.middleware bound dispatch 24.9s
express.middleware anonymous 24.8s
http.request PUT
http.request PUT 20s
flask.request PUT
flask request PUT 19.8s
flask.application PUT 19.75s
flask.dispatch_request 19.6s
collateral items.get_existing_collateral_items_batch 19.5s
requests.requests 6.2s
pymysql.query 4s
flask.request/compute-collateral/:batchid 14s
flask.dispatch_request/compute-collateral/:batchid 13.7s
compute_collateral/compute-collateral/:batchid 13.6s
pymysql.query 3.7s
fastapi.request 19.4s
Response
credit-orchestrator 27.8s
credit-offer-api 26.7s
credit-risk-api 6.8s
cache-middleware 0.05s
redis 0.05s
credit-score-db 6.74s
collateral-api 19.5s
requests 13.6s
compute-collateral-service 13.6s
pymysql.query 3.7s

Figure 11: Trace Summary

B.5 Software Engineering Reasoning

Task : Software Engineering Reasoning

Provide a response to the question pertaining to a software engineering task, considering both best and bad practices in areas like programming, operating systems, and networking, etc.

Input

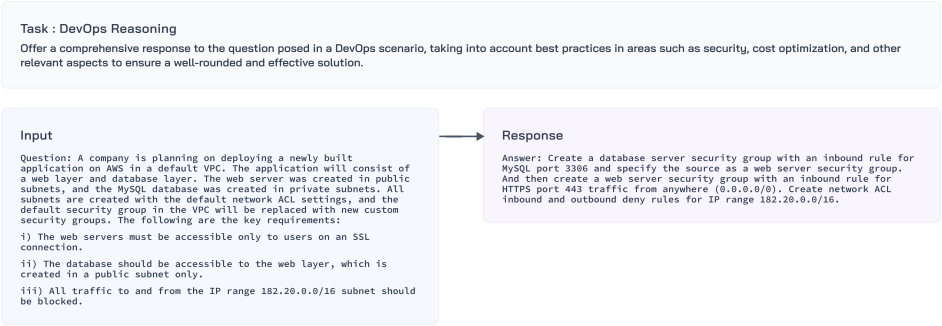
```
Question: Why is it dangerous to use MutableInteger?
```

Response

```
Answer: MutableInteger lacks thread safety and is vulnerable to retaining
stale values. Consequently, when one thread invokes the set operation,
concurrent threads making get calls may or may not see the update. This
can result in significant issues, including unexpected exceptions,
corrupted data structures, inaccurate computations, and infinite loops.
```

Figure 12: Software QA

B.6 Cloud DevOps Understanding



Task : DevOps Reasoning

Offer a comprehensive response to the question posed in a DevOps scenario, taking into account best practices in areas such as security, cost optimization, and other relevant aspects to ensure a well-rounded and effective solution.

Input

Question: A company is planning on deploying a newly built application on AWS in a default VPC. The application will consist of a web layer and database layer. The web server was created in public subnets, and the MySQL database was created in private subnets. All subnets are created with the default network ACL settings, and the default security group in the VPC will be replaced with new custom security groups. The following are the key requirements:

i) The web servers must be accessible only to users on an SSL connection.

ii) The database should be accessible to the web layer, which is created in a public subnet only.

iii) All traffic to and from the IP range 182.20.0.0/16 subnet should be blocked.

Response

Answer: Create a database server security group with an inbound rule for MySQL port 3306 and specify the source as a web server security group. And then create a web server security group with an inbound rule for HTTPS port 443 traffic from anywhere (0.0.0.0/0). Create network ACL inbound and outbound deny rules for IP range 182.20.0.0/16.

Figure 13: DevOps QA

C Appendix: Rouge-2 Metric Normalization for Summarization Tasks

Rouge scores are notoriously hard to quantify since they rely on the comparison of automatically generated summaries with reference summaries, introducing subjectivity and variability in the evaluation process. Additionally, the reliance on lexical overlap and n-gram matching in Rouge metrics may not capture the nuances of semantic equivalence, making it challenging to discern the true informativeness and quality of a generated summary. Furthermore, Rouge scores might not fully account for aspects like fluency, coherence, or the overall meaningfulness of the summary, contributing to the complexity in their interpretation. However, Rouge-2 has been identified as the most effective proxy for the mentioned metrics in tasks related to summarization. Nevertheless, the challenge lies in the non-intuitiveness of this metric. To enhance its interpretability and significance, we adopt the inter-annotator expert agreement score as our baseline. This involves averaging the pairwise Rouge-2 scores provided by three human experts, namely E1, E2, and E3. Our investigation revealed that humans tend not to favor a specific summary if it contains more hallucinations, categorizing such summaries as "undesirable". Consequently, if a particular model prediction exhibits more than two hallucinations, we assign a score of 0.

$$\text{Inter-Annotator ROUGE-2 Score Average} = \frac{\text{ROUGE-2}_{\mathrm{E}_1,E_2} + \text{ROUGE-2}_{\mathrm{E}_2,E_3} + \text{ROUGE-2}_{\mathrm{E}_3,E_1}}{3}$$

$$\text{Predicted ROUGE-2 Score Average} = \frac{\text{ROUGE-2}_{\mathrm{m},E_1} + \text{ROUGE-2}_{\mathrm{m},E_2} + \text{ROUGE-2}_{\mathrm{m},E_3}}{3}$$

$$\text{Normalize Model Prediction Score} = \frac{\text{AVG\_Rouge-2}_{\text{predicted}}}{\text{AVG\_Rouge-2}_{\text{annotator}}}$$