UNIVERSIDADE DE SÃO PAULO
Instituto de Ciências Matemáticas e de Computação

Automated Generation of Troubleshooting Guides from Incident Data Using Large Language Models

Bruna Magrini da Cruz

Monograph - MBA in Artificial Intelligence and Big Data

ICMC USP SÃO CARLOS

Bruna Magrini da Cruz

Automated Generation of Troubleshooting Guides from Incident Data Using Large Language Models

Monograph presented to the Departamento de Ciências de Computação do Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo - ICMC/USP, as part of the requirements for obtaining the title of Specialist in Artificial Intelligence and Big Data.

Concentration area: Artificial Intelligence and Big Data

Advisor: Profa. Dra. Solange Oliveira Rezende

Original version

São Carlos

2025

Ficha catalográfica elaborada pela Biblioteca Prof. Achille Bassi
e Seção Técnica de Informática, ICMC/USP,
com os dados inseridos pelo(a) autor(a)

C957a

Cruz, Bruna Magrini da
Automated Generation of Troubleshooting Guides from Incident Data Using Large Language Models / Bruna Magrini da Cruz; orientadora Solange Oliveira Rezende. -- São Carlos, 2025.
56 p.

Trabalho de conclusão de curso (MBA em Inteligência Artificial e Big Data) -- Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, 2025.

1. Incident clustering. 2. Creation of troubleshooting guides. 3. Incident cause detection. 4. Incident mitigation detection. I. Rezende, Solange Oliveira, orient. II. Título.

Bibliotecários responsáveis pela estrutura de catalogação da publicação de acordo com a AACR2:
Gláucia Maria Saia Cristianini - CRB - 8/4938
Juliana de Souza Moraes - CRB - 8/6176

Bruna Magrini da Cruz

Automated Generation of Troubleshooting Guides from Incident Data Using Large Language Models

Monografia apresentada ao Departamento de Ciências de Computação do Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo - ICMC/USP, como parte dos requisitos para obtenção do título de Especialista em Inteligência Artificial e Big Data.

Área de concentração: Inteligência Artificial e Big Data

Orientadora: Profa. Dra. Solange Oliveira Rezende

Versão original

São Carlos

2025

ACKNOWLEDGEMENTS

First and foremost, I would like to thank my advisor Profa. Dra. Solange Oliveira Rezende whose expertise and constant support have been fundamental to the completion of this thesis.

I would like to express my sincere gratitude to my colleagues from the IV class of the specialization in Artificial Intelligence and Big Data, whose unwavering support, generosity in sharing knowledge, and collaborative spirit greatly enriched my learning experience even within the limitations of a remote setting.

I am also grateful to my wife Heloísa Silva de Lima Araujo who has been my pillar of strength throughout this journey and my most unwavering source of support, offering encouragement and insight even beyond the boundaries of her own field.

Finally, I thank the artificial intelligence tools, particularly ChatGPT, which served as an invaluable assistant in refining my technical writing under my supervision.

ABSTRACT

CRUZ, B.M. **Automated Generation of Troubleshooting Guides from Incident Data Using Large Language Models**. 2025. 56 p. Monograph (MBA in Artificial Intelligence and Big Data) - Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, São Carlos, 2025.

The maintenance phase in large-scale cloud environments is critical as it ensures that incidents reported by users or engineers are effectively mitigated. Troubleshooting Guides (TSGs) play a key role in documenting past incidents, enabling faster resolution by future On-Call Engineers (OCEs); however, generating high-quality TSGs is time-consuming, requiring technical expertise and synthesis of diverse information sources. This study proposes an automated pipeline to generate TSGs by leveraging clustering techniques to group incidents representing the same underlying issue, followed by the application of Large Language Models (LLMs) guided through prompt engineering to extract the cause, symptoms, and mitigation steps. A dataset combining synthetic and confidential cloud incidents was created, totaling 26 incidents across 10 classes. The effectiveness of K-Means clustering was assessed using the Adjusted Rand Index (ARI), yielding a score of 0.71, which indicates a high level of alignment with the ground truth incident classes. Generated TSGs were manually evaluated for correctness and quality, achieving a 67% success rate for correctness and 78% for quality. Incorporating chat histories as additional context improved the completeness of the guides. These results demonstrate the feasibility of automatically generating high-quality TSGs, reducing manual effort and enhancing incident resolution efficiency, while highlighting the potential of combining human expertise with LLM-based automation.

Keywords: incident clustering. creation of troubleshooting guides. incident cause detection. incident mitigation detection.

RESUMO

CRUZ, B.M. **Automated Generation of Troubleshooting Guides from Incident Data Using Large Language Models**. 2025. 56 p. Monografia (MBA em Inteligência Artificial e Big Data) - Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, São Carlos, 2025.

A fase de manutenção em ambientes de nuvem em larga escala é essencial para garantir a mitigação eficaz de incidentes reportados por usuários ou engenheiros. Os *Troubleshooting Guides* (TSGs) são fundamentais para documentar incidentes passados, permitindo que *On-Call Engineers* (OCEs) resolvam novos olproblemas de forma mais rápida. Entretanto, a criação de TSGs de alta qualidade é demorada, exigindo expertise técnica e síntese de múltiplas fontes de informação. Este estudo propõe um pipeline automatizado para gerar TSGs, combinando técnicas de agrupamento para identificar incidentes relacionados ao mesmo problema subjacente e o uso de grandes modelos de linguagem (*Large Language Models* LLMs) com engenharia de prompts para extrair causa, sintomas e etapas de mitigação. Um conjunto de dados, composto por incidentes sintéticos e confidenciais, totalizando 26 incidentes distribuídos em 10 classes, foi criado para validação. A clusterização foi realizada com *K-Means* e avaliada pelo *Adjusted Rand Index* (ARI), que resultou em 0,71, indicando alta concordância com os rótulos de referência. Os TSGs geradas foram avaliadas manualmente, obtendo 67% de sucesso em corretude e 78% em qualidade. A inclusão de históricos de chat como contexto adicional aprimorou a completude dos guias. Os resultados demonstram a viabilidade da geração automática de TSGs de alta qualidade, reduzindo o esforço manual e aumentando a eficiência na resolução de incidentes, além de evidenciar o potencial da combinação entre expertise humana e automação baseada em LLMs.

Palavras-chave: agrupamento de incidentes; criação de documentação; detecção de causa de incidentes; detecção de mitigação de incidentes.

LIST OF FIGURES

LIST OF TABLES

LIST OF PROMPTS

CONTENTS

1 INTRODUCTION

1.1 Context

The engineering life cycle encompasses all stages of a system, including planning, design, implementation, testing, deployment, and maintenance. In a large-scale cloud environment, the maintenance phase is crucial as it ensures that errors reported by users or engineers are handled.

Technology companies, such as Google, Meta, and Microsoft, include the maintenance of existing systems among the responsibilities of an engineer (Newson, 2017; Gasperetti; Egebo, 2022; Staff, 2023). In this sense, incident management is an essential task, and engineers work in rotations for an on-call shift where they are the main point of contact for solving issues. During their shift, they can be referred to as On-Call Engineers (OCEs) whose goal is to investigate and solve incidents.

When the same issue occurs multiple times, it is important that an engineer documents it. As a result, future OCEs can rely on existing instructions, reducing Time To Mitigate (TTM) incidents. According to An *et al.* (2024), incidents with documentation show a 60% shorter TTM compared to those without it. The documentation of incidents is known as a Troubleshooting Guide (TSG), it can include background knowledge, possible root causes, and steps for diagnosis and mitigation. Ideally, a TSG is comprehensive, concise, precise, and reproducible. Therefore, when a new incident arrives, the OCE can search for the corresponding TSG and follow the guidelines.

1.2 Motivation

The TSG is a valuable source of information, as it outlines the steps to resolve an issue. Not only does it assist OCEs during their shifts, but with the advancement of reasoning capabilities in large language models (LLMs), it also facilitates the automated resolution of incidents (Las-Casas *et al.*, 2024).

However, creating a TSG is a time-consuming task, as it requires technical expertise and the collection of various elements such as explanation, code, data, logs, screenshots, and more.

Furthermore, because engineers work on multiple projects, writing a TSG is not always a high priority, and when written, the TSG can suffer from insufficient and inadequate content, outdated and ambiguous information, as well as incorrect and unexplained examples (Aghajani *et al.*, 2020).

From that, the study seeks to answer the question: How can the generation of

TSGs be automated to save engineers' time and produce high-quality instructions?

1.3 Objective

The goal is to propose an automated pipeline to create high-quality TSGs. Therefore, given at least two different incidents related to the same issue, the pipeline should output a TSG containing instructions on the root cause, symptoms, and/or mitigation steps.

Considering this problem, the following specific objectives have been established:

1. Create a dataset of incidents, each with a title, description, and a set of comments that simulates the discussions made during its resolution. Also, a corresponding high-quality TSG, with information that could be derived from some or all of the incident's fields.
2. Group incidents that are related to the same issue using clustering techniques.
3. Extract the most relevant information from the incident and generate a high-quality TSG using LLM.

As a result, the workflow can be applied to any real incident database and generate TSGs. Finally, TSGs can be used by engineers or automated incident resolution systems, with minimal or no corrections.

2 THEORETICAL FRAMEWORK

2.1 Related work

In the field of generating suggestions to support incident resolution, Ahmed *et al.* (2023) study the effectiveness of GPT-3.x models to assist engineers in identifying root causes and mitigating incidents. Likewise, An *et al.* (2024) propose Nissist, a system which leverages TSGs to create incident mitigation suggestions. Related, Goel *et al.* (2024) explore using data from different stages of the software development life cycle to generate root cause recommendations for the incident.

Concerning automated execution of TSGs, Shetty *et al.* (2022) analyze the usage of over 4,000 TSGs, propose a taxonomy of quality-related issues found in these guides, and introduce AutoTSG, a framework designed to execute TSGs automatically by extracting relevant statements and translating them into actionable commands. Equally relevant, Las-Casas *et al.* (2024) introduce LLexus, a system that transforms TSGs into executable plans during an offline phase, which can later be executed by an executor system when an incident occurs.

Additionally, Jiang *et al.* (2020) propose DeepRmd to match an incident with the correct TSG by using textual similarity between the incident description and its corresponding TSG. Similarly, Chen *et al.* (2019) explore incident matching, but focuses on identifying the correct team to own the incident.

Finally, within the scope of automated documentation generation, Khan and Uddin (2022) explore the use of a GPT-3-based model to generate code documentation based on source code.

A thorough search of the relevant literature found no studies exploring the automated creation of TSGs based on previously resolved incidents.

2.2 Concepts

The main concepts that directly impact this study are LLMs, clustering, and prompt engineering, which are presented in the following sections.

2.2.1 Large language models

LLMs are models that can comprehend and generate text. They are trained on vast amounts of texts using Transformer architecture (Lin *et al.*, 2022) to learn patterns and relationships within language. Then, the models can be used in a variety of tasks such as translation, question answering, and code generation.

For enhancing even more the knowledge of LLM, fine-tuning can be applied, where the model is adapted to a specific domain by further training it on a task-specific dataset. Another approach is to use prompt engineering, a practice for creating prompts to guide the LLM.

A well-known LLM platform is ChatGPT, created by OpenAI, capable of generating responses to text input in a conversation format (Deng; Lin, 2023). Currently, ChatGPT can be based on different deep learning models, such as GPT-4, GPT-4o, or GPT-5. They differ on cost, speed, token limits, knowledge cutoff, and output quality (OpenAI, 2025a). As the most recent release, GPT-5 unifies advanced reasoning, multimodal input, and task execution into a single system, besides reducing hallucinations. Also, it contains variants with different purposes: GPT-5 is the best for deep reasoning and complex workflows, GPT-5-Mini is faster, lower-cost option with solid reasoning, and GPT-5-Nano is ultra-fast for real-time and embedded uses.

Similarly, Gemini is an alternative created by Google with the same capabilities, but independent deep learning models, such as Gemini 2.0 and Gemini 2.5. The Gemini 2 builds on the original Gemini framework with enhanced reasoning and multimodal understanding, offering improved language comprehension, contextual awareness, and support for complex tasks. It includes variants such as Gemini 2 Pro, which is optimized for advanced coding and agentic tasks; Gemini 2 Lite, designed for faster performance and lower computational cost; and Gemini Flash, focused on ultra-high-speed processing and efficiency for applications where latency is critical (Google, 2025).

Finally, other options are currently available in the market, including Microsoft Copilot, Anthropic's Claude, Meta AI, and DeepSeek.

2.2.2 Clustering

Clustering is a technique used to group data points based on their similarity. It falls under unsupervised learning which obtains information from unlabeled examples. With this technique, it is possible to assign data points to groups or estimate how likely each data point is to belong to a given group. The algorithms operate on data represented as vectors where each vector encodes an item through its corresponding feature values.

In the case where data are formatted as text, they may first go through preprocessing, such as the removal of stopwords, removal of irrelevant tokens, lemmatization, lowercasing, and other normalization steps to reduce noise.

After preprocessing, the text needs to be transformed into a numerical format that can be used for computation. A common method is Term Frequency–Inverse Document Frequency (TF-IDF) which assigns weights to words based on how often they appear in a document compared to their frequency across the whole dataset. TF-IDF is simple, fast,

and works well in many classical natural language processing tasks, but it does not capture the meaning or relationships between words.

To address this, more advanced vectorization techniques are used to capture semantics. Word2Vec, for example, learns word representations where similar words are placed closer together in a vector space. Sentence-BERT (SBERT) extends this idea to entire sentences, producing embeddings that keep their overall meaning. More recently, LLMs like GPT have been used to create contextual embeddings, where the meaning of a word depends on the text around it. (Birunda; Devi, 2021).

Once texts are represented as vectors, clustering algorithms can be applied to group items that share similarities. K-Means is one of the most widely used techniques, as it creates k groups by assigning each item to the nearest center. Its main drawback is that the number of groups must be set in advance. However, k can be empirically found with techniques such as silhouette method, which measures how similar a point is to its own cluster compared to other clusters. Other clustering alternatives include hierarchical clustering, which builds a tree of groups; Density-Based Spatial Clustering of Applications with Noise (DBSCAN), which can find clusters of different shapes and handle noise; and neural network–based methods, that learn clusters from the data itself (Lam; Wunsch, 2014).

To measure how well clustering results align with ground truth labels, the Rand Index (RI) is commonly used. RI evaluates the similarity between two partitions by considering all pairs of elements and checking whether they are assigned to the same or different clusters in both partitions. However, since RI can be biased by chance, the Adjusted Rand Index (ARI) was introduced. ARI corrects this bias by adjusting the score according to the expected similarity of random labelings. As a result, ARI provides a more reliable evaluation of clustering quality, with values ranging from -1 (worse than random), through 0 (random), up to 1 (perfect agreement), while RI ranges from 0 to 1 without correction for chance (Hubert; Arabie, 1985).

In many cases, the vector space is very large, which makes visualization and analysis difficult. To solve this, dimensionality reduction techniques are used. Principal Component Analysis (PCA) is one of the most popular, as it reduces the data to fewer dimensions while keeping most of the important information. Another option is t-distributed Stochastic Neighbor Embedding (t-SNE) which is useful for showing small-scale patterns, although it is more computationally expensive and does not scale as well as PCA.

With that, it is possible to automatically identify incidents that are similar and, thus, refer to the same issue.

2.2.3 Prompt engineering

Prompt engineering is a technique for designing and optimizing prompts that instructs LLMs. Fine-tuning a model is not always an option, as it requires training the model again in a specific dataset. Therefore, prompt engineering is an alternative where the engineer writes instructions and/or adds specific knowledge for the model to perform a task.

The prompt provided to the model impacts its final answers. In this sense, to achieve the best result possible, LLM companies offer guidelines on how to write an effective prompt.

Accordingly, OpenAI suggestions are, among others, as follows (OpenAI, 2025b):

- Use delimiters to clearly indicate distinct parts of the input.
- Specify the steps required to complete a task.
- Use inner monologue or a sequence of queries to hide the model's reasoning process.
- Ask the model if it missed anything on previous passes.
- Include input/output examples in the prompt to showcase the expected patterns.

Additionally, Google suggestions are, among others, as follows (Cloud, 2025b):

- Use action verbs to specify the desired action.
- Define the desired length and format of the output.
- Specify the target audience.
- Reference specific sources or documents.
- Show the desired level of detail.
- Quantify your requests whenever possible.

These best practices will be leveraged to write the prompts that will guide the LLMs in generating the TSGs.

2.3 Research gap

As of today, LLM is employed to find the root cause, the mitigation steps, and the appropriate existing TSG to an incident. Nevertheless, the generation of TSGs remains largely underexplored. If left unaddressed, this gap may lead to documentation that is

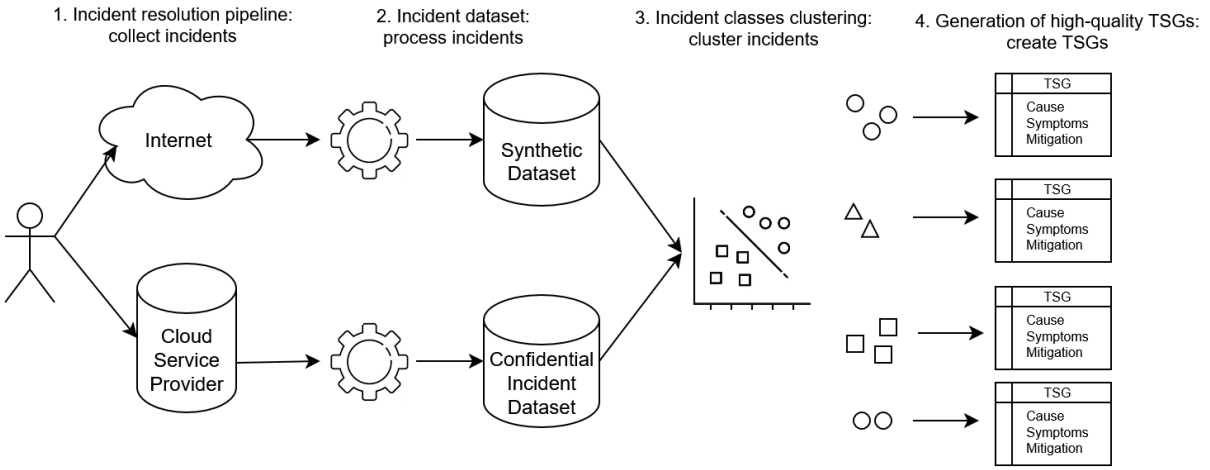
incomplete, of poor quality, and costly in terms of time and effort. Therefore, this project seeks to advance the automated creation of TSG by leveraging clustering techniques to group incidents that represent the same underlying issue, and applying prompt engineering strategies to extract and structure the essential information.

3 METHODOLOGY

3.1 Project

Briefly, a dataset of incidents was constructed using mocked data collected from the Internet and real incidents obtained from the internal data of a cloud service provider. Subsequently, incidents representing the same underlying issue were clustered. Finally, the cause, symptoms, and mitigation were extracted by an LLM from each cluster to generate a high-quality TSG. Figure 1 presents the step-by-step workflow employed.

Figure 1 – Overview of step-by-step workflow



Source: Author

3.2 Incident resolution pipeline

When an incident is created in a cloud environment, it contains the title, description, Severity (SEV), and the responsible team as required information. Conventionally, the SEV levels range from 5 to 1: SEV 5 is a cosmetic issue without customer impact; SEV 4 is a minor issue also without customer impact; SEV 3 represents a minor incident with low impact; SEV 2 is a critical incident with significant impact; and SEV 1 corresponds to a critical incident with high impact. Incidents may be triggered either manually, reported by an engineer or customer, or automatically, through monitoring systems that detect metrics breaching predefined thresholds.

As soon as the incident is assigned to a team, a respective OCE is designed. The first goal of the OCE is mitigation of the incident, where this process may or may not include the finding of the root cause of the issue. Usually, the OCE follows one or more steps from the described below:

- Search for existing TSGs based on key words.

- Search for similar incidents based on key words.
- Search for existing chats conversations or email threads based on key words.
- Obtain additional data about the issue using identifiers provided in the incident in databases or debugging tools.
- Notify stakeholders about potential impacts.
- Seek input from stakeholders on mitigation strategies.
- Understand how and mitigate the incident.
- Discover the correct team that should own the incident and reassign it.

This process can take a few hours to several months, depending on the SEV. After the incident is mitigated and the customer is no longer impacted, it is up to the OCE and the team to decide whether or not the root cause should be identified and added to the incident, or if a TSG will be created. The Figure 2 shows an illustrative example of a fictitious incident.

It is important to emphasize that relevant information may be only shared or discussed on external platforms (such as chats, emails, or synchronous meetings). Therefore, not all critical details relevant to the incident resolution will necessarily be recorded in the comments. Additionally, incorrect hypotheses can also be commented, and the incident can undergo multiple rounds of ownership changes, during which actions could be executed by intermediate teams.

This project focus exclusively on SEV 3 type. Also, the dataset used only contain information shared in the incidents. Finally, the mitigation may take the form of resolving the incident or transferring its ownership.

3.3 Incident dataset

Existing public datasets lacked one or more required fields necessary for the analysis: description, comments, cause, symptoms, mitigation, and/or TSGs. Consequently, two datasets were developed for this project. The first is a synthetic incident dataset simulating the typical behavior of cloud incidents. The second is a confidential incident dataset derived from internal data of a cloud service provider.

Both datasets were normalized to the same structure, consisting of incident classes that group together incidents related to the same issue. The fields for each class are defined in Table 1.

Furthermore, Table 2 describes the fields for each incident.

Figure 2 – Illustrative example of a fictitious incident

Title: My kubernetes pod keep restarting
Description: I deploy a elasticsearch to minikube with below configure file:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: elasticsearch
spec:
  replicas: 1
  [...]
    spec:
      containers:
        - name: elasticsearch
          image: elasticsearch:7.10.1
          ports:
            - containerPort: 9200
            - containerPort: 9300
```

However, my pod keeps restarting:

```
$ kubectl get pod
NAME                            READY   STATUS  RESTARTS   AGE
elasticsearch-fb9b44948-bchh2   1/1     Running 5          6m23s
```

SEV: 3
Team: Kubernetes Pod Team
OCE: John

Comments:
- John: Can you check the error message using `kubectl describe pod elasticsearch-fb9b44948-bchh2` ?

- User: Yes, here is the result:

```
Events:
  Type     Reason     Age                    From               Message
  ----     ------     ----                   ----               -------
  Normal   Scheduled  7m11s                  default-scheduler  Successfully assigned
  Normal   Created    3m18s (x5 over 7m11s)  kubelet            Created container elasticsearch
  Normal   Started    3m18s (x5 over 7m10s)  kubelet            Started container elasticsearch
  Warning  BackOff    103s (x11 over 5m56s)  kubelet            Back-off restarting failed container
```

- John: The error message indicates that there is a problem with your application, and therefore, kubernetes can't initialize the pod. Please, double check your configuration file. I will transfer your case to ElasticSearch Team, since this is application related.

- Transfer from Kubernetes Pod Team to ElasticSearch Team [...]

Source: Author

For the creation of the synthetic incidents, a manual approach was taken. Firstly, by looking for existing TSGs of well established clouds like Azure (2025), Cloud (2025a) or AWS (2025). Then, searching on the Internet within question-and-answers forums, such as Stackoverflow or Reddit, for inquiries related to resolving the issue addressed by some of the TSGs. Finally, the inquiries were processed through transformation into markdown format, substitution of images with text descriptions generated by a LLM, and assignment of numeric identifiers to each author. Moreover, the TSGs were also processed by converting to markdown format and generating structured fields to represent the cause, symptoms, and mitigation. Subsequently, each pair was later interpreted as (incident, TSG) within the dataset.

The synthetic dataset was constructed with 11 incidents in total, distributed across

Table 1 – Description of the fields defined for each incident class in the dataset

Name	Description	Type
Class	A unique identifier representing a group of incidents related to the same underlying issue.	String
Incidents	A list comprising all incidents associated with the same issue.	List

Source: Author

Table 2 – Description of the fields defined for each incident in the dataset

Name	Description	Type
Title	A brief sentence summarizing the issue.	String
Description	A detailed explanation about the issue. It may contain error messages, logs or other relevant data.	String
Comments	A list of comments provided by various authors. Each comment contains the author, with author "0" indicating the individual who created the incident, and subsequent represent other commenters. Also, the comment text.	List
Link	The link for the question that originated the synthetic incident. This field is empty for the confidential dataset.	String
Troubleshooting	A high-quality TSG example, including its link if available, as well as the cause, symptoms, and mitigation of the incident.	Object

Source: Author

5 classes.

For the creation of the confidential incident dataset, four incident classes were selected—three resolved through mitigation steps and one through ownership transfer. At least three incidents per class were retrieved from the internal database. The description and comments fields were then processed by removing HTML tags (if applicable), replacing images with text descriptions generated by an LLM (where relevant), and assigning numeric identifiers to each author. Since no existing TSG matched the selected classes, internal experts were consulted to establish the criteria for an acceptable solution for each class. Finally, it is important to emphasize that, due to the confidential nature of the data, this dataset will not be shared. Only its particular characteristics were referenced during the tuning phase, and the corresponding results were reported during the validation phase.

The confidential dataset was constructed with 14 incidents in total, distributed across 4 classes.

At last, the incidents from both datasets were consolidated, yielding the final dataset used for prompt tuning and validation. The Table 3 presents the number of incidents per incident class.

Exceptionally, one additional confidential incident was collected, named "extra", accompanied by two chat histories documenting its resolution. Initially, the issue was

Table 3 – Description of number of incidents per incident class

Dataset	**Class**	**Number of incidents**
Synthetic	locate_statistics_sql_server	3
Synthetic	request_body_too_large_azure_storage	3
Synthetic	container_in_transitioning_state	2
Synthetic	full_text_search	2
Synthetic	load_balancer_with_link_cannot_be_deleted	1
Confidential	class1	3
Confidential	class2	3
Confidential	class3	4
Confidential	class4	4

Source: Author

contextualized in an exchange between the responsible OCE and a subject-matter expert. Subsequently, a group chat involving the OCE and multiple subject-matter experts was created, where additional details were addressed. Finally, the incident comments were also used for incident resolution discussion. The goal is to identify whether additional knowledge sources related to an incident can enhance the TSG creation process. This incident underwent the same processing as the synthetic and confidential datasets, with the additional inclusion of comment and message timestamps, enabling the LLM to establish temporal context. The author numeric identifiers was also standardized between the incident and the two chat histories. Finally, due to the confidential nature of the data, this incident will also remain undisclosed; however, its content and behavior were analyzed.

3.4 Incident classes clustering

In real cloud environments, the incident class is unknown because classification is not part of the incident resolution pipeline. As such, a clustering phrase is essential to automatically determine which incidents are related to the same underlying issue.

Based on the synthetic and confidential datasets, together with the extra incident, a total of 26 incidents were available for clustering. Here, the dataset was manually classified to evaluate the clustering algorithms. Although the small sample size may bias the results, the approach should be regarded as an exploration of the pipeline's feasibility, which can be further expanded and validated with larger datasets. Moreover, it is important to highlight that the majority of the incident classes are associated with distinct technologies. A summary of the scope of each incident class is provided below:

- locate_statistic_sql_server: Issues related to Microsoft SQL Server statistics and replicas.
- request_body_too_large_azure_storage: Errors caused by Azure Blob Storage

request size limitations.

- container_group_in_transitioning_state: Failures in Azure Container Instances stuck in transitioning state.
- full_text_search: Problems related to Microsoft SQL Server full-text indexing.
- load_balancer_with_link_cannot_be_deleted: Deletion conflicts involving Azure Private Link load balancers.
- class1: Incidents associated with disk-related failures.
- class2: Incidents associated with GPU.
- class3: Incidents associated with GPU.
- class4: Incidents associated with network connectivity issues.
- extra: Incidents associated with CPU memory resource constraints.

Therefore, satisfactory clustering results are expected, given the limited dataset size and the fact that the incident topics are nearly exclusive.

Accordingly, a classical pipeline was adopted. TF-IDF was employed for text-to-vector representation due to its interpretability, efficiency with small datasets, and low computational cost. K-Means was then selected as the clustering algorithm, given its efficiency and ease of implementation, despite its sensitivity to outliers. Finally, t-SNE was applied for dimensionality reduction and visualization, as it produces high-quality visual representations of high-dimensional data in small datasets.

The dataset, entirely composed in English, underwent an initial preprocessing stage that included stopword removal and text normalization through lowercasing. Subsequently, TF-IDF was applied to obtain the vector representation. Finally, the performance of K-Means was assessed under the assumption of the correct number of clusters, $k = 10$, and its effectiveness was quantitatively evaluated using the ARI.

An additional experiment was conducted to evaluate the capability of inferring the optimal k. K-Means was executed with k varying as integer from two to 20, and for each configuration the silhouette score was computed. This metric quantifies intra-cluster cohesion and inter-cluster separation without reliance on ground truth. The value of k corresponding to the maximum silhouette score was interpreted as the optimal.

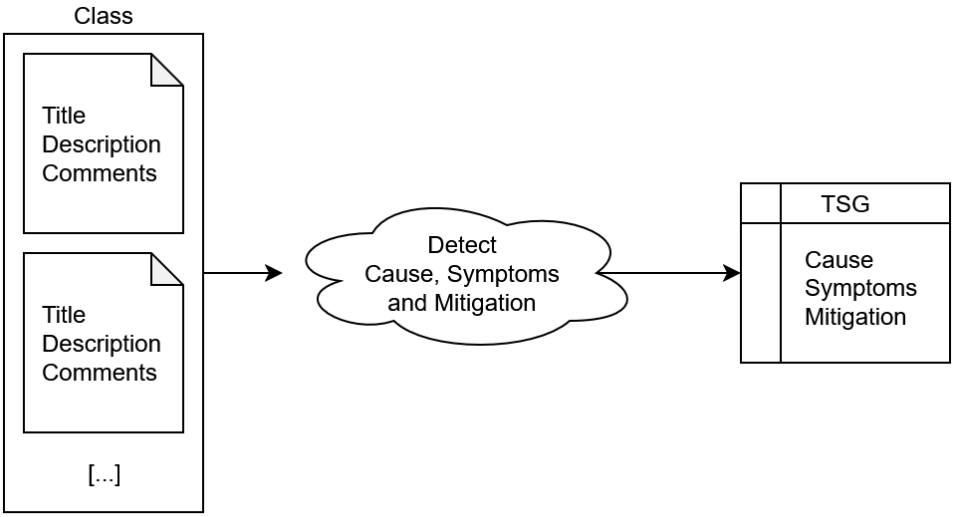
Ultimately, two aspects of evaluation were considered: (i) the feasibility of estimating the optimal number of clusters, and (ii) the feasibility of correctly identifying the clusters themselves. If both conditions are met, the clustering pipeline can be applied to incident datasets to identify entries that refer to the same underlying issue.

3.5 Generation of high-quality TSGs

Given at least two incidents from the same class, a TSG was generated through the application of prompt engineering techniques using Gemini 2.0 Flash from Google.

The first workflow attempt used an unified prompt responsible for finding the three main elements of the TSG: cause, symptoms, and mitigation. Its design is shown in Figure 3. However, this approach mostly caused confusion for the LLM which had difficulties understanding the difference between cause and symptoms. Additionally, because each element has its own structure — for example, symptoms are typically expressed as error messages, whereas mitigation may involve commands — the LLM was unable to adhere to all the instructions within a single prompt. Finally, most prompt engineering guidelines recommend framing specific and well-defined questions, thereby enabling the LLM to produce more precise responses.

Figure 3 – Unified prompt design



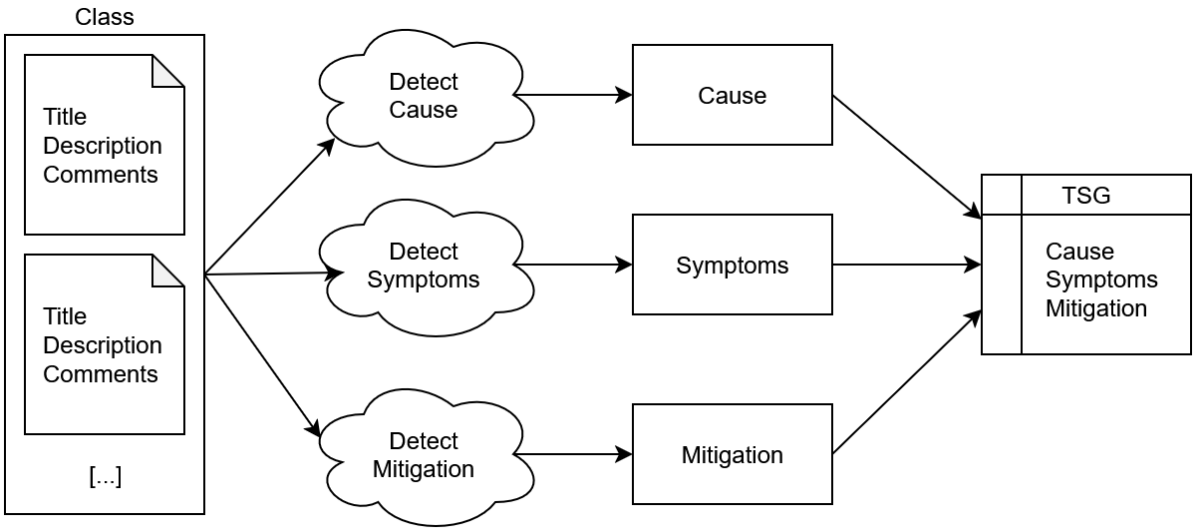
Source: Author

Therefore, the second workflow, which was ultimately adopted as the design, consisted of a multi-prompt pipeline, in which the cause, symptoms, and mitigation were individually generated through distinct prompts. Later, they were consolidated into a TSG. The diagram is shown in Figure 4.

Two metrics were defined to evaluate the TSGs: TSG correctness and TSG quality.

Because the LLM is not intended to independently determine the cause, symptoms, or mitigation, but rather to utilize the existing available knowledge to generate the TSG, "TSG correctness" is defined as the mandatory inclusion of, at a minimum, the correct mitigation steps, given that all incidents in the dataset contain this information within the comments. Furthermore, the inclusion of the cause and symptoms depends on the incident

Figure 4 – Distinct prompt design



Source: Author

class and whether this information is available within the respective incidents. Table 4 presents the requirements for cause, symptoms, and mitigation for each incident class.

Table 4 – Requirements for cause, symptoms, and mitigation per incident class

Dataset	**Class**	**Mitigation**	**Symptoms**	**Cause**
Synthetic	locate_statistics_sql_server	Yes	Yes	No
Synthetic	request_body_too_large_-azure_storage	Yes	Yes	Yes
Synthetic	container_in_transitioning_-state	Yes	Yes	No
Synthetic	full_text_search	Yes	Yes	Yes
Synthetic	load_balancer_with_link_can-not_be_deleted	Yes	Yes	Yes
Confidential	class1	Yes	Yes	Yes
Confidential	class2	Yes	Yes	No
Confidential	class3	Yes	No	No
Confidential	class4	Yes	Yes	No

Source: Author

After each TSG was generated, it was evaluated against the predefined requirements of its incident class, with each requirement assigned a value of one point. The Equation 3.1 presents the formula used to calculate the final score.

$$S_c = \frac{R}{M} \tag{3.1}$$

Where:

- S_c = The correctness score of the TSG, ranging from 0 to 1.

- R = The amount of requirements obtained by the TSG.
- M = The maximum number of requirements for the incident class.

In addition to including the appropriate cause, symptoms, and mitigation, the TSG is expected to adhere to best practices, thereby ensuring organization, reproducibility, and completeness. For "TSG quality", each TSG was evaluated and assigned a score based on the presence of the following criteria:

1. No overlapping or repeated information between the different sections.
2. Steps must be ordered and numbered, where each step has no more than one action, and if a step needs multiple actions, an existing sub list (and sub numbered, e.g. 2.1, 2.1.1, etc.).
3. Whenever an action or condition depends on some external data, split it explicitly into two distinct steps.
4. Explicitly specification on how to perform an action, using markdown like blocks and the corresponding language for actions to be taken.
5. Explicitly specification on data flow for each step where the input required for that step must be obtained from a previous step or from the incident metadata.
6. Generalized steps to ensure reproducibility.
7. Appropriate placeholders for parameters when specifying commands/APIs (e.g. <DNS IP> instead of <XYZ>).
8. Focused on cause, symptoms and mitigation, without "teaching" in the TSG.
9. Usage of simple, concise, correct and formatted grammar.
10. Consistent terminology and abbreviations throughout the TSG, ensuring that the same terms are used for the same concepts to avoid confusion.

After each TSG was generated, it was also evaluated against the criteria, with each criterion assigned a value of one point. The Equation 3.2 presents the formula used to calculate the final score.

$$S_q = \frac{T}{10} \tag{3.2}$$

Where:

- S_q = The quality score of the TSG, ranging from 0 to 1.

- T = The amount of criterion obtained by the TSG.

A different approach was taken to the incident extra in order to evaluate the potential benefits of incorporating additional sources of information. Unlike the other incident classes, which were processed solely based on incident data, this case was explored in two distinct ways: one using only the original incident information and another enriched with the corresponding chat histories exchanged among experts during the troubleshooting process. By applying both versions to the prompt engineering pipeline, it was possible to compare how the presence of contextual discussions could influence the quality and completeness of the generated troubleshooting guide.

Finally, the main prompt engineering techniques explored were inner monologue to hide the model's reasoning, Markdown delimiters to clearly indicate distinct parts of the input and output, definition of the desired length and format of the output, and examples. Additionally, to reduce variance and avoid sporadic hits, each prompt was always executed three times to ensure that potential LLM inconsistencies did not compromise the results. The acceptance criterion is defined as achieving success in at least two out of three attempts.

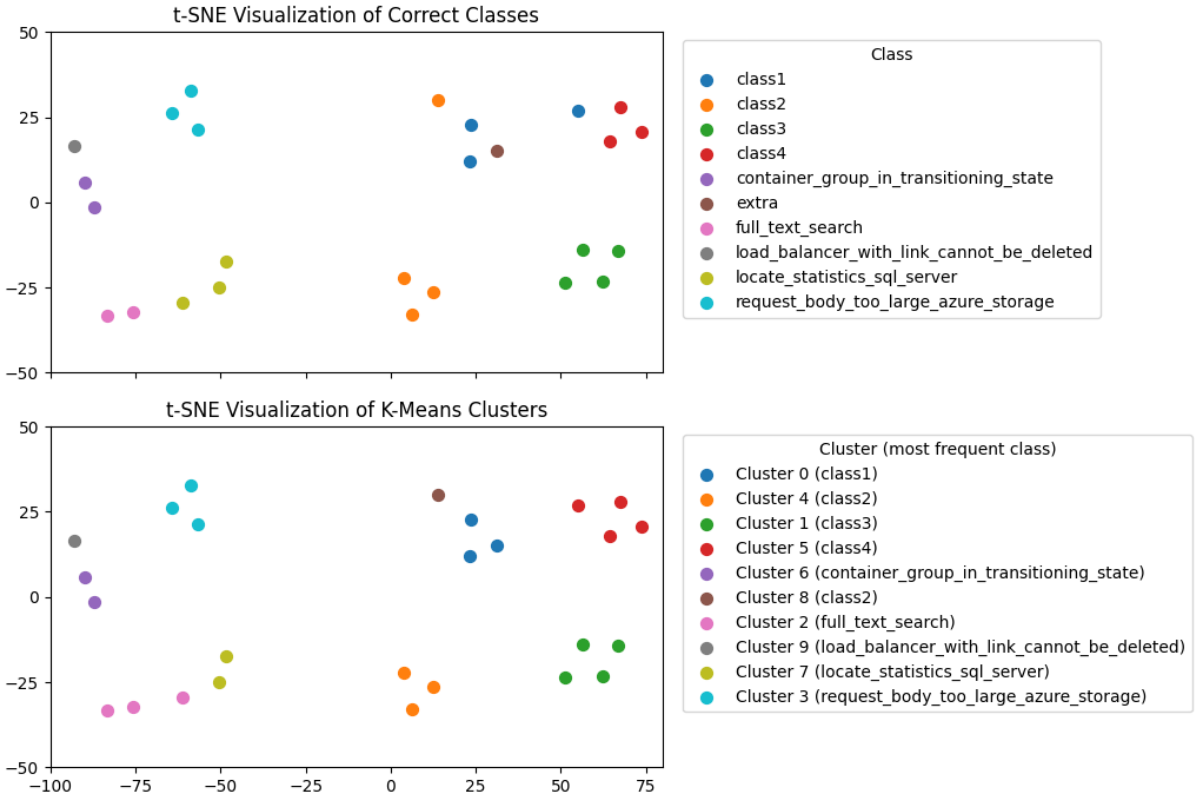
4 EVALUATION

4.1 Evaluation of incident classes

Within the incident dataset, including the extra incident, a total of 10 incident classes were defined. K-Means was first executed with the correct number of clusters, $k = 10$. The clustering quality was assessed using the ARI, which yielded a score of 0.71, demonstrating a high degree of agreement with the ground truth labels.

For interpretability and inspection of the results, t-SNE was applied to project the high-dimensional vectors into a two-dimensional space. Figure 5 first presents the ground-truth classes and then illustrates the cluster distribution produced by K-Means. In the first visualization, each color represents a distinct class, whereas in the second visualization, each color represents a distinct cluster, with the most frequent correct class shown in parentheses.

Figure 5 – Visualizations of correct classes and k-Means clusters with t-SNE



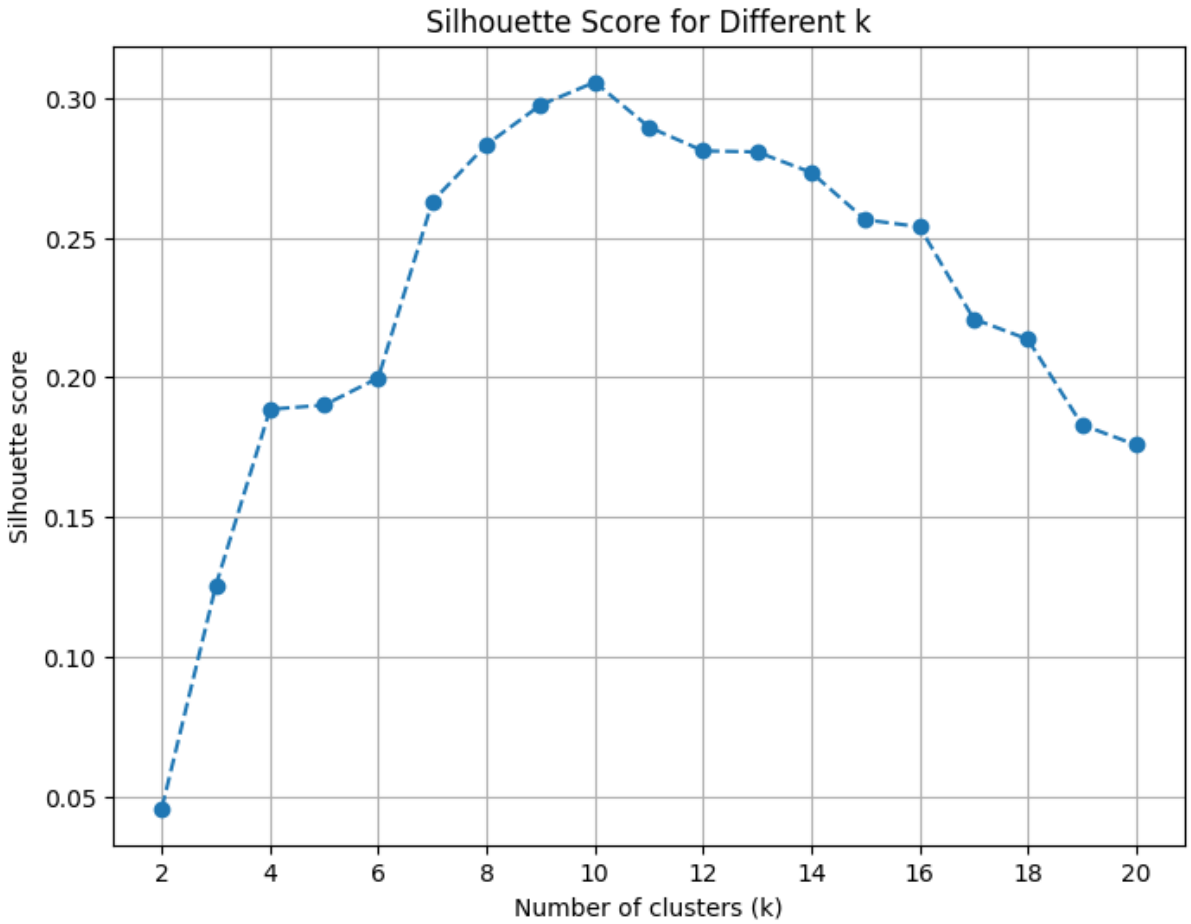
Source: Author

The results confirm that, when provided with the correct number of clusters, K-Means successfully groups the majority of incidents according to their underlying issues. Misclassifications occurred in specific cases: within class1, one incident was incorrectly

assigned to class4, likely due to both incidents being generated by the same automation pipeline and thus sharing a highly similar structure. Moreover, one incident from locate_statistics_sql_server was incorrectly assigned to the full_text_search class, probably because both incidents are related to the same technology: Microsoft SQL Server. Furthermore, one incident from class2 was treated as an outlier instead of the incident extra. Such misclassifications suggest that more advanced representation techniques than TF-IDF could improve the clustering performance, by more accurately representing the distinctions between incidents.

In practical scenarios, the number of clusters k is typically unknown, raising the question of whether the silhouette score can effectively estimate the optimal value. To evaluate this, silhouette scores were computed for k values ranging from two to 20, as shown in Figure 6.

Figure 6 – Silhouette score with k ranging from two to 20



Source: Author

The result suggests that the silhouette score can serve as a reliable criterion for estimating k. As observed, the maximum score indicates the correct $k = 10$ as the appropriate cluster value. Consequently, in contexts where the true number of clusters is not

predefined, this approach enables the estimation of k and, by extension, the identification of coherent clusters.

Ultimately, clustering techniques can be leveraged to estimate the number of failures in a dataset and to group incidents that correspond to the same underlying issue. This pipeline can be further extended as the dataset scales, paving the way for a fully automated framework for incident clustering.

4.2 Evaluation of high-quality TSGs

After several iterations of testing, the Appendix A, Appendix B, and Appendix C contain the final versions of the prompts for detecting the cause, symptoms, and mitigation, respectively.

Specifically for class4, where the correct solution was to transfer the incident to another team, the LLM only generated the correct mitigation when an explicit example was provided in the prompt, indicating that this could be an acceptable solution. It is important to recognize that prompt engineering is not a static process but rather an iterative methodology that adapts to the characteristics of each dataset and the behavior of the LLMs employed.

Moreover, the LLM was tested to consolidate the sections into a TSG. However, since the cause, symptoms, and mitigation were already well-defined individually, they were simply appended. This ensured full correctness while avoiding unpredictability or unintended text alterations by an LLM.

Due to the limited dataset size, the TSG was manually assessed based on the two evaluation criteria: TSG correctness and TSG quality. The Table 5 presents the fulfillment of requirements across incident classes for TSG correctness: 1 denotes that the requirement was satisfied, 0 that it was not, and N/a that the section was not applicable to that incident class.

It is important to recall that all incidents involved are categorized as SEV 3, meaning that the mitigations are generally less complex. However, SEV 3 incidents represent the most frequent type in a real cloud operational environment. As such, the ability to generate effective TSGs for these incidents can have the highest impact on operational efficiency and incident resolution at scale.

In summary, each TSG was assigned two scores: the TSG correctness score S_c and TSG quality score S_q, both ranging from 0 to 1. The Table 6 summarizes the correctness score and the quality score achieved by the TSGs per incident class.

Out of the nine TSGs, six were correctly generated, yielding a success rate of 67%. For the three instances where the TSGs were not accurately created, the first issue occurred with request_body_too_large_azure_storage and the load_balancer_-

Table 5 – Result of the correctness score for cause, symptoms, and mitigation of the TSG per incident class

Dataset	**Class**	**Mitigation**	**Symptoms**	**Cause**
Synthetic	locate_statistics_sql_server	1	1	N/a
Synthetic	request_body_too_large_-azure_storage	1	1	0
Synthetic	container_in_transitioning_-state	1	1	N/a
Synthetic	full_text_search	1	1	1
Synthetic	load_balancer_with_link_can-not_be_deleted	1	1	0
Confidential	class1	1	1	1
Confidential	class2	1	1	N/a
Confidential	class3	0	N/a	N/a
Confidential	class4	1	1	N/a

Source: Author

Table 6 – Result of the correctness score and the quality score of the TSG per incident class

Dataset	**Class**	S_c	S_q
Synthetic	locate_statistics_sql_server	1	1
Synthetic	request_body_too_large_azure_storage	0.67	1
Synthetic	container_in_transitioning_state	1	1
Synthetic	full_text_search	1	1
Synthetic	load_balancer_with_link_cannot_be_deleted	0.67	0.9
Confidential	class1	1	1
Confidential	class2	1	0.8
Confidential	class3	0	1
Confidential	class4	1	1

Source: Author

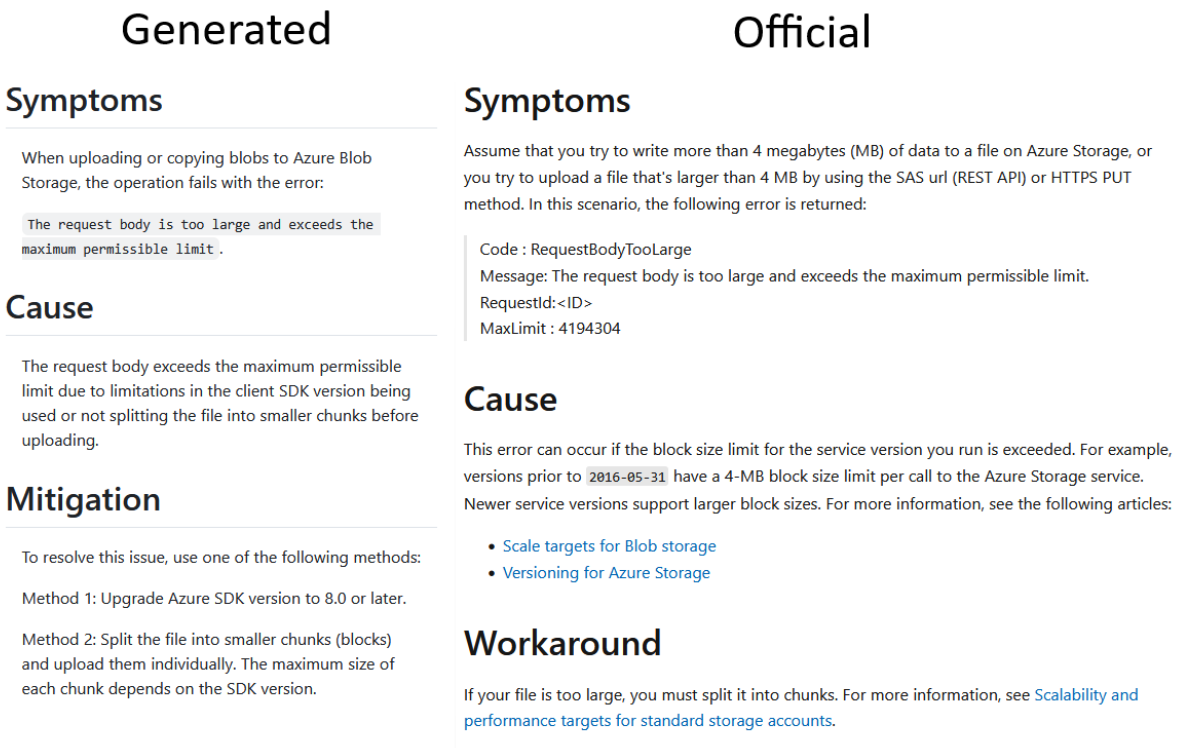
with_link_cannot_be_deleted classes, where the LLM incorrectly insisted on adding the mitigation within the cause. In the second case, concerning class3, the incident comments presented conflicting mitigation suggestions: one was generated by an automation system, while the other was the action performed by the OCEs. The LLM encoutered confusion in determing which recommendation to follow. It is noteworthy that this situation could also confuse an entry-level engineer, as both suggestions had validity in different contexts.

Out of the nine TSGs, seven were considered to be of high quality, yielding a success rate of 78%. The criteria with the least adherence were 2 and 10. For criterion 2, the LLM struggled with using lists and sub lists effectively; it should be noted that no attempts were made to iteratively refine or optimize the prompt to address this limitation, as the goal was to evaluate the LLM's performance under typical conditions. For criterion 10, each section was generated independently, which affected the consistency of terminology and abbreviations throughout the entire TSG.

Lastly, the results were assessed by an expert, who confirmed the correctness and quality scores. The expert also noted that, although the TSGs did not capture all possible knowledge about the issue, they did include all relevant information that could be extracted from the incident data.

Finally, Figure 7 presents a comparison between the TSG generated by the workflow proposed in this study for the class request_body_too_large_azure_storage and its official version available on the troubleshooting section of Microsoft documentation (Microsoft, 2025). Similarly, Figure 8 also presents a comparison, but for full_text_search class (Microsoft, 2023).

Figure 7 – Comparison between a TSG generated by this study for request_body_too_large_azure_storage class and its official version written by Microsoft

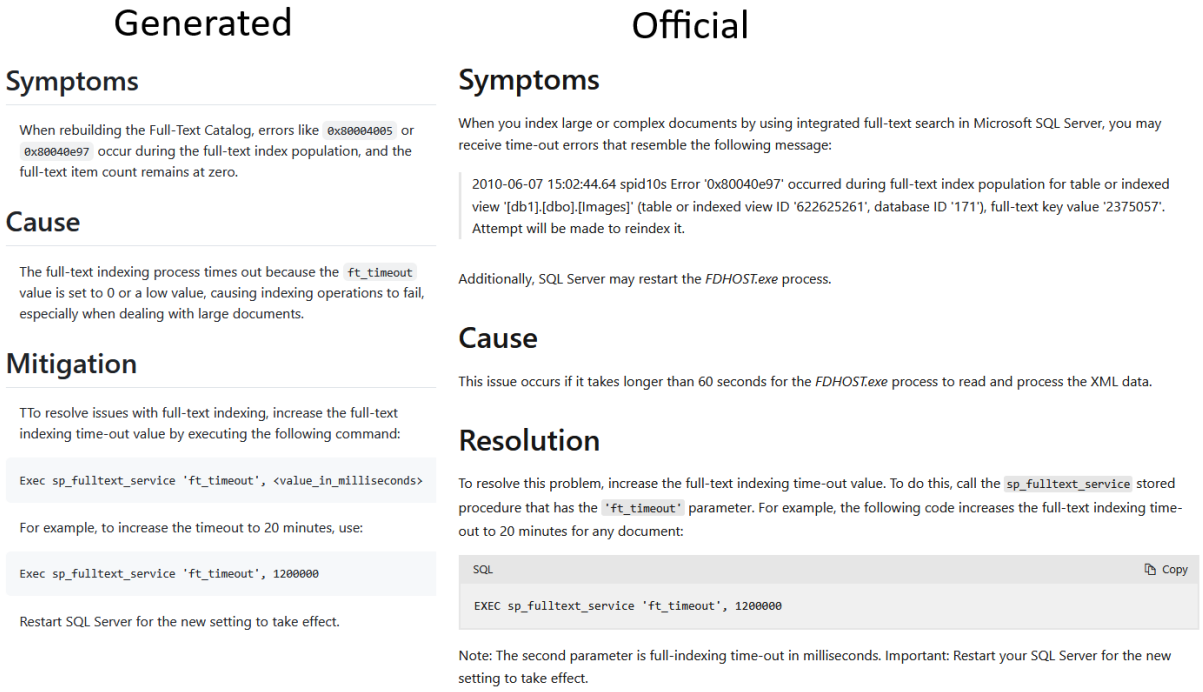


Source: Author

For incident extra, the prompts were executed using two variants: one including the chat histories and another without them. In both cases, the symptoms and root cause were successfully identified. Nevertheless, the mitigation generated when chat histories were incorporated proved to be more comprehensive and detailed, offering a deeper explanation of the underlying cause due to the multiple rounds of expert discussion. Moreover, this version provided additional step-by-step commands, despite the LLM exhibiting limitations in handling highly complex queries, which it preferred to simplify.

Ultimately, the LLM was able to detect the symptoms, cause, and mitigation from a set of incidents related to the same underlying issue, and thereby generate a high-quality

Figure 8 – Comparison between a TSG generated by this study for full_text_search class and its official version written by Microsoft



Source: Author

TSG. Such a TSG can be leveraged by OCEs when a similar incident is triggered, or by LLMs as part of automated incident management pipelines.

Therefore, if the OCEs responsible for an incident provide relevant comments with sufficiently detailed information, the automatic generation of TSGs with enough examples can be accomplished. This highlights the potential of combining human expertise with LLM-based automation to improve incident response efficiency.

5 CONCLUSION

This study presented an automated approach for generating TSGs from previously resolved incidents in cloud environments. By applying clustering techniques, incidents related to the same underlying issue were effectively grouped while LLMs guided by prompt engineering enabled structured extraction of the cause, symptoms, and mitigation steps.

Evaluation demonstrated the feasibility of the proposed pipeline: clustering achieved an ARI of 0.71, indicating alignment with ground truth labels, and TSG generation achieved a 67% success rate for correctness and 78% for quality. Additionally, incorporating chat discussions as supplementary context enhanced the completeness and detail of the generated guides.

In conclusion, the studied approach can be realistically applied in large cloud environments to automate the generation of TSGs. Its adoption has the potential to significantly increase the availability of high-quality documentation, directly supporting OCEs during their shifts with reliable troubleshooting material. This ensures that engineers can focus more on problem-solving and less on repetitive documentation, besides ensuring that the knowledge is systematically captured.

For companies, this translates into tangible benefits: higher engineer productivity by outsourcing documentation tasks, faster incident mitigation through reduced TTM, and ultimately improved service reliability and customer satisfaction. Furthermore, once integrated with additional automation pipelines, the generated TSGs could enable partial or even full incident resolution without human intervention, representing a step toward self-healing cloud infrastructures. In the long term, such advances can reduce operational costs, and optimize resource allocation.

Despite these positive outcomes, certain limitations were identified. The dataset size was limited, occasional confusion between cause and mitigation occurred, and inconsistencies in terminology were observed across TSGs generated with multiple prompts. Furthermore, the study focused exclusively on SEV 3 incidents, leaving other severity levels unexplored. These constraints highlight the importance of scaling experiments and testing under more diverse real-world conditions.

Future work may explore several directions to enhance the pipeline. First, different workflows for prompt engineering could be tested, including providing the LLM responsible for generating the mitigation steps with both the cause and symptoms, allowing it to produce more accurate and context-aware solutions. It is also important to improve the LLM's understanding that mitigation instructions should be separated from the cause, reducing confusion and enhancing clarity. Additionally, integrating external information

sources through multi-cloud platform (MCP) servers or other external systems could further improve TSG generation by providing richer contextual data. Moreover, more advanced representation and clustering techniques could be employed to enhance performance, or alternatively, different LLMs could be utilized. Finally, expanding the dataset to include incidents of varying severity and technologies, and integrating automatic TSG generation into real operational workflows would further reduce mitigation time and increase OCEs efficiency.

Overall, the results demonstrate the feasibility and potential impact of combining human expertise with LLM-based automation to produce high-quality TSGs, paving the way for more efficient incident management in cloud environments.

REFERENCES

AGHAJANI, E. *et al.* Software documentation: the practitioners' perspective. *In*: **2020 IEEE/ACM 42nd International Conference on Software Engineering (ICSE)**. [*S.l.: s.n.*], 2020. p. 590–601.

AHMED, T. *et al.* Recommending root-cause and mitigation steps for cloud incidents using large language models. *In*: **2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)**. [*S.l.: s.n.*], 2023. p. 1737–1749.

AN, K. *et al.* Nissist: An incident mitigation copilot based on troubleshooting guides. **arXiv preprint arXiv:2402.17531**, 2024.

AWS. **AWS TROUBLESHOOTING**. 2025. Available at: https://docs.aws.amazon.com/awssupport/latest/user/troubleshooting.html. Access at: 04 aug. 2025.

AZURE. **MICROSOFT TROUBLESHOOTING DOCUMENTATION**. 2025. Available at: https://learn.microsoft.com/en-us/troubleshoot/. Access at: 04 aug. 2025.

BIRUNDA, S. S.; DEVI, R. K. A review on word embedding techniques for text classification. **Innovative Data Communication Technologies and Application: Proceedings of International Conference on Innovative Data Communication Technologies and Application (ICIDCA 2020)**, Springer, p. 267–281, 2021.

CHEN, J. *et al.* An empirical investigation of incident triage for online service systems. *In*: **2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)**. [*S.l.: s.n.*], 2019. p. 111–120.

CLOUD, G. **GOOGLE CLOUD TROUBLESHOOTING**. 2025. Available at: https://cloud.google.com/storage/docs/troubleshooting. Access at: 04 aug. 2025.

CLOUD, G. **PROMPT ENGINEERING: OVERVIEW AND GUIDE**. 2025. Available at: https://cloud.google.com/discover/what-is-prompt-engineering?hl=en. Access at: 27 aug. 2025.

DENG, J.; LIN, Y. The benefits and challenges of chatgpt: An overview. **Benefits**, v. 2, n. 2, p. 2022, 2023.

GASPERETTI, J.; EGEBO, N. **META'S SEV CULTURE: HOW TODAY'S SEVS CREATE TOMORROW'S RELIABILITY**. 2022. Available at: https://atscaleconference.com/videos/metas-sev-culture-how-todays-sevs-create-tomorrows-reliability/. Access at: 19 apr. 2025.

GOEL, D. *et al.* X-lifecycle learning for cloud incident management using llms. *In*: **2024 ACM 32nd ACM International Conference on the Foundations of Software Engineering (FSE)**. [*S.l.: s.n.*], 2024. p. 417–428.

GOOGLE. **MODELS**. 2025. Available at: https://ai.google.dev/gemini-api/docs/models. Access at: 27 aug. 2025.

HUBERT, L.; ARABIE, P. Comparing partitions. **Journal of classification**, Springer, v. 2, n. 1, p. 193–218, 1985.

JIANG, J. *et al.* How to mitigate the incident? an effective troubleshooting guide recommendation technique for online service systems. *In*: **Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering**. [*S.l.: s.n.*], 2020. p. 1410–1420.

KHAN, J. Y.; UDDIN, G. Automatic code documentation generation using gpt-3. *In*: **2022 IEEE/ACM 37th International Conference on Software Engineering: Automated Software Engineering (ICSE-ASE) Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering**. [*S.l.: s.n.*], 2022. p. 1–6.

LAM, D.; WUNSCH, D. C. Clustering. **Academic Press Library in Signal Processing**, Elsevier, v. 1, p. 1115–1149, 2014.

LAS-CASAS, P. *et al.* Llexus: an ai agent system for incident management. **ACM Special Interest Group on Operating Systems (SIGOPS): Operating Systems Review (OSR)**, Association for Computing Machinery, New York, NY, USA, v. 58, n. 1, p. 23–36, ago. 2024. ISSN 0163-5980. Available at: https://doi.org/10.1145/3689051.3689056.

LIN, T. *et al.* A survey of transformers. **AI open**, Elsevier, v. 3, p. 111–132, 2022.

MICROSOFT. **Error 0x80040e97 occurs when you use integrated full-text search in SQL Server**. 2023. Available at: https://learn.microsoft.com/en-us/troubleshoot/sql/database-engine/search/error-0x80040e97-using-full-text-search. Access at: 03 sep. 2025.

MICROSOFT. **Error when you write more than 4 MB of data to Azure Storage: Request body is too large**. 2025. Available at: https://learn.microsoft.com/en-us/troubleshoot/azure/azure-storage/blobs/connectivity/request-body-large. Access at: 03 sep. 2025.

NEWSON, P. **ADVENTURES IN SRE-LAND: INCIDENT MANAGEMENT AT GOOGLE**. 2017. Available at: https://cloud.google.com/blog/products/gcp/incident-management-at-google-adventures-in-sre-land. Access at: 19 apr. 2025.

OPENAI. **MODELS**. 2025. Available at: https://platform.openai.com/docs/models/. Access at: 19 apr. 2025.

OPENAI. **PROMPT ENGINEERING**. 2025. Available at: https://platform.openai.com/docs/guides/prompt-engineering/prompt-engineering. Access at: 19 apr. 2025.

SHETTY, M. *et al.* 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**. [*S.l.: s.n.*], 2022. p. 1477–1488.

STAFF, I. T. **ROTATING DEVOPS ROLE IMPROVES ENGINEERING SERVICE QUALITY**. 2023. Available at: https://www.microsoft.com/insidetrack/blog/rotating-devops-role-improves-engineering-service-quality/. Access at: 19 apr. 2025.

APPENDIX

APPENDIX A – PROMPT USED TO DETECT THE CAUSE

The Prompt A.1 was used to detect the cause of the incident. It should be noted that the TSGs "Error 41131 when creating availability group"[1] and "Error 41131 when creating availability group"[2] were incorporated into the prompt as high-quality examples.

```
# Goal
You are an assistant responsible for identifying the cause for a ticket.

# Input
You will receive between 1 to 5 tickets about the same issue. Each of them will contain:
- Title: A brief sentence summarizing the error.
- Description: A detailed explanation about the issue. It may contain error messages, logs or
↪  other relevant data.
- Comments: A list of comments made by different authors.
  - The author 0 represents the person who created the ticket. All the other authors represent
  ↪  different people who interacted in the ticket.
  - Each comment may or may not contain relevant information. It can occur that someone is
  ↪  following a wrong direction towards the right cause.

# Instructions
1. Your answer will be used in a technical guide. Therefore, you should always focus on repeted
↪  patterns and reproducible instructions.
2. You should identify the root cause shared by all tickets.
  2.1 The "cause" is what creates the error.
  2.2 Note that "cause" is different than "symptoms" (how the issue will be perceived by the
  ↪  user with error logs/code/message) and than "mitigation" (how to solve the issue).
3. You have to detect the root cause in the tickets fields. DO NOT create or suppose the cause
↪  by yourself. The answer should rely entirely on affirmations made on the tickets.
4. You should identify and answer with the cause, and only the cause. DO NOT include suggestions
↪  or hypothesis of mitigation.
5. Here are two examples of high-quality causes:
First example:
"This issue occurs if the `[NT AUTHORITY\\SYSTEM]` account is missing from the SQL Server login
↪  or if the account lacks the necessary permissions to create the high-availability group."
Second example:
"You configured your AKS cluster to access the cluster API server by using authorized IP address
↪  ranges that your computer can't access."

# Output
You should answer following strictly the template below:

# Cause
## Reasoning
<The reasoning behind you believe this is the cause. You should leverage the content available
↪  in the tickets and your own knowledge.>
## Answer
<The root cause containing less than 500 caracteres. Be detailed and specific, but always with
↪  information that you have confidence.>
```

Prompt A.1 – Prompt used to detect the cause
Source: Author

[1] https://learn.microsoft.com/en-us/troubleshoot/sql/database-engine/availability-groups/error-41131-create-availability-group

[2] https://learn.microsoft.com/en-us/troubleshoot/azure/azure-kubernetes/connectivity/cannot-access-cluster-api-server-using-authorized-ip-ranges

APPENDIX B – PROMPT USED TO DETECT THE SYMPTOMS

The Prompt B.2 was used to detect the symptoms of the incident. It should be noted that the TSGs "Error 41131 when creating availability group"[1] and "Error 41131 when creating availability group"[2] were incorporated into the prompt as high-quality examples.

[1] https://learn.microsoft.com/en-us/troubleshoot/sql/database-engine/availability-groups/error-41131-create-availability-group

[2] https://learn.microsoft.com/en-us/troubleshoot/azure/azure-kubernetes/connectivity/cannot-access-cluster-api-server-using-authorized-ip-ranges

```
# Goal
You are an assistant responsible for identifying the symptoms of a ticket.

# Input
You will receive between 1 to 5 tickets about the same issue. Each of them will contain:
- Title: A brief sentence summarizing the error.
- Description: A detailed explanation about the issue. It may contain error messages, logs or
↪  other relevant data.
- Comments: A list of comments made by different authors.
  - The author 0 represents the person who created the ticket. All the other authors represent
  ↪  different people who interacted in the ticket.
  - Each comment may or may not contain relevant information. It can occur that someone is
  ↪  following a wrong direction towards the right cause.

# Instructions
1. Your answer will be used in a technical guide. Therefore, you should always focus on repeted
↪  patterns and reproducible instructions.
2. You should identify the common symptoms across all tickets and return them unified as one
↪  answer.
  2.1 The "symptons" are how the issue manifests itself.
  2.2 Note that "symptoms" is different than "cause" (what creates the issue) and than
  ↪  "mitigation" (how to solve the issue).
3. Usually, the symptoms manifets itself as an error code, error message or error log. If there
↪  is one, it is mandatory to have it in your answer.
4. You should identify and answer with the symptons, and only the symptoms.
5. Here are two examples of high-quality symptoms:
First example:
"When you try to create a high-availability group in Microsoft SQL Server, you receive the
↪  following error message:
```
Msg 41131, Level 16, State 0, Line 2
Failed to bring availability group 'availability_group' online. The operation timed out. Verify
↪ that the local Windows Server Failover Clustering (WSFC) node is online. Then verify that
↪ the availability group resource exists in the WSFC cluster. If the problem persists, you
↪ might need to drop the availability group and create it again.
```"
Second example:
"If you try to create or manage resources in an AKS cluster, you can't access the cluster API
↪  server. When you run kubectl, you receive the following error message: `Unable to connect to
↪  the server: dial tcp x.x.x.x:443: i/o timeout`"
6. In the end, your explanation should strictly use the format:
<what the user is trying to do, i.e., the action the user is performing> followed by <error
↪  message or log, if any>.
Note on the high-quality examples how no additional information is provided, focusing on what
↪  the user tried and the error that happened, without extra explanations.

# Output
You should answer following strictly the template below:

# Symptoms
## Reasoning
<The reasoning behind you believe this is the symptoms. You should leverage the content
↪  available in the tickets and your own knowledge.>
## Answer
<The symptoms containing less than 300 caracteres. Be detailed and specific, but always with
↪  information that you have confidence.>
```

Prompt B.2 – Prompt used to detect the symptoms
Source: Author

APPENDIX C – PROMPT USED TO DETECT THE MITIGATION

The Prompt C.3 was used to detect the mitigation of the incident. It should be noted that the TSGs "Error 41131 when creating availability group"[1] and "Error 41131 when creating availability group"[2] were incorporated into the prompt as high-quality examples.

[1] https://learn.microsoft.com/en-us/troubleshoot/sql/database-engine/availability-groups/error-41131-create-availability-group

[2] https://learn.microsoft.com/en-us/troubleshoot/azure/azure-kubernetes/connectivity/cannot-access-cluster-api-server-using-authorized-ip-ranges

```
# Goal
You are an assistant responsible for detecting the mitigation for a ticket.

# Input
You will receive between 1 to 5 tickets about the same issue. Each of them will contain:
- Title: A brief sentence summarizing the error.
- Description: A detailed explanation about the issue. It may contain error messages, logs or other relevant data.
- Comments: A list of comments made by different authors.
  - The author 0 represents the person who created the ticket. All the other authors represent different people who interacted in the ticket.
  - Each comment may or may not contain relevant information. It can occur that someone is following a wrong direction towards the right cause.

# Instructions
1. Your answer will be used in a technical guide. Therefore, you should always focus on repeted patterns and reproducible instructions.
2. You should identify the common mitigation across all tickets and return them unified as one answer.
  2.1 The "mitigation" is how to solve the error.
  2.1 Note that "mitigation" is different than "symptoms" (how the issue will be perceived by the user). It is also different than "cause" (what creates the issue).
3. You have to detect the mitigation in the tickets fields. DO NOT create or suppose the cause by yourself. The answer should rely entirely on affirmations made on the tickets.
4. You should identify and answer with the nutugatuin, and only the mitigation. DO NOT include the cause or symptoms.
5. Here are three examples of high-quality mitigation:
First example:
"To resolve this issue, use one of the following methods:

Method 1: Use manual steps
1. Create a login in SQL Server for the `[NT AUTHORITY\\SYSTEM]` account on each SQL Server computer that hosts a replica in your availability group.
2. Grant the `[NT AUTHORITY\\SYSTEM]` account the following server-level permissions:
- ALTER any availability group
- Connect SQL
- View server state
Note: Make sure that no other permissions are granted to the account.

Method 2: Use script
To create the `[NT AUTHORITY\\SYSTEM]` account, run the following statement in a query window:
```SQL
USE [master]
GO
CREATE LOGIN [NT AUTHORITY\\SYSTEM] FROM WINDOWS WITH DEFAULT_DATABASE=[master]
GO
```
2. To grant the permissions to the `[NT AUTHORITY\\SYSTEM]` account, run the following statement in a query window:
```
GRANT ALTER ANY AVAILABILITY GROUP TO [NT AUTHORITY\\SYSTEM]
GO
GRANT CONNECT SQL TO [NT AUTHORITY\\SYSTEM]
GO
GRANT VIEW SERVER STATE TO [NT AUTHORITY\\SYSTEM]
GO
```"
Second example:
"Make sure that when you run the az aks create or az aks update command in Azure CLI, the `--api-server-authorized-ip-ranges` parameter includes the IP addresses or IP address ranges of the automation, development, or tooling systems that are being used."
Third example:
"Since this issue is not related to CPU memory, it should be transfer to GPU memory team."

Output
You should answer following strictly the template below:

Mitigation
Reasoning
<The reasoning behind you believe this is the mitigation. You should leverage the content available in the tickets and your own knowledge.>
Answer
<The mitigation containing less than 500 caracteres. Be detailed, specific, and follow a step by step approach, but always with information that you have confidence.>
```

Prompt C.3 – Prompt used to detect the mitigation
Source: Author