

SYSTEM IMPLEMENTATION AND TESTING

The System Implementation and Testing chapter outlines the process of transforming the system design into a functional application, followed by rigorous testing to ensure its correctness and reliability. This chapter covers the models' implementation, screenshots of the outcomes of each model in different scenarios, and test cases used to validate each of the system's components and its functionality, performance, and adherence to requirements.

4.1 SYSTEM MODELS

The system is implemented as a modular architecture, with each module designed to handle a specific functionality. The implementation details of the five models are as follows:

4.1.1 Upload Documents Model

This model enables users to upload, process, and store documents into a vector database or cloud service he specifies for efficient retrieval.

- Documents are uploaded through a user interface and pre-processed.
- The user can specify the context store in which the documents are to be stored, either GCS or Amazon s3 Bucket or Chroma vector database.
- If the context store selected is chroma, then the next three steps are taken. Else the document is stored in the cloud service storage.
- Preprocessing includes splitting documents into smaller chunks using a splitter to handle large inputs effectively.
- Each chunk is embedded using an embedding model to create embeddings.
- These embeddings are stored in the Chroma vector database for future retrieval.

4.1.2 Retriever Model

This model retrieves top-k most relevant contexts from the Chroma vector database or GCS or Amazon s3 Bucket based on user specification and the prompt. 'k' represents the number of contexts to be retrieved.

- If the chosen context store is Chroma, then the following three steps are taken.
- User prompt is embedded using the same embedding model.
- The embedded prompt is compared with stored vectors using similarity metrics, and the top-k most relevant contexts are retrieved.
- Retrieved contexts are returned with their respective scores, indicating relevance.
- Else if the context store selected is Google or AWS, then the user prompt is sent to the search service of the cloud storage service providers to retrieve the top-k most relevant contexts, then the response is processed and returned.

4.1.3 RAG Model

This model combines contexts retrieval and response generation to create meaningful response to the prompt.

- Relevant contexts are retrieved using retriever models (Chroma or Google or AWS).
- Retrieved contexts are passed to the response generation component.
- Then a Large Language Model (LLM) is prompted to answer the prompt using the retrieved context to produce response.
- Response is finally formatted and returned to the user.

4.1.4 Robust RAG Model using Keywords

This enhanced model ensures robust responses by most frequent keywords among potential contexts and utilizing them for response generation.

- Retrieved contexts are passed one after the other in an isolated manner to a LLM response generator to extract initial responses.
- Keywords are extracted from responses which are not placeholder response 'I don't know' using spaCy.
- Keyword frequencies are analyzed, and filtering is performed using constants such as ALPHA and BETA or user-specified percentile.
- Filtered keywords are incorporated into a prompt and sent to an LLM for robust response generation.

4.1.5 Robust RAG Model using Contexts

This model uses context scoring and filtering in addition to the process of extracting most frequent keywords to improve the accuracy of generated responses.

- Retrieved contexts are passed one after the other in an isolated manner to a LLM response generator to extract initial responses.
- Keywords are extracted from responses which are not placeholder response 'I don't know' using spaCy.
- Keyword frequencies are analysed, and filtering is performed using constants such as ALPHA and BETA or user-specified percentile.
- Retrieved contexts are scored based on the presence and frequency of most frequent keywords.
- A scoring threshold is calculated using user-specified percentile.
- Contexts with scores above the threshold are selected as the most relevant and benign contexts.
- Filtered contexts are included in the prompt for generating the final response using an LLM.

4.2 RESULT SCREENSHOTS

This section includes screenshots of the results of various models developed in this project.

4.2.1 Upload Documents Model

This model enables users to upload documents, preprocess them for efficient handling, and store them in a specified context store such as Google, AWS, or Chroma vector database for seamless retrieval.

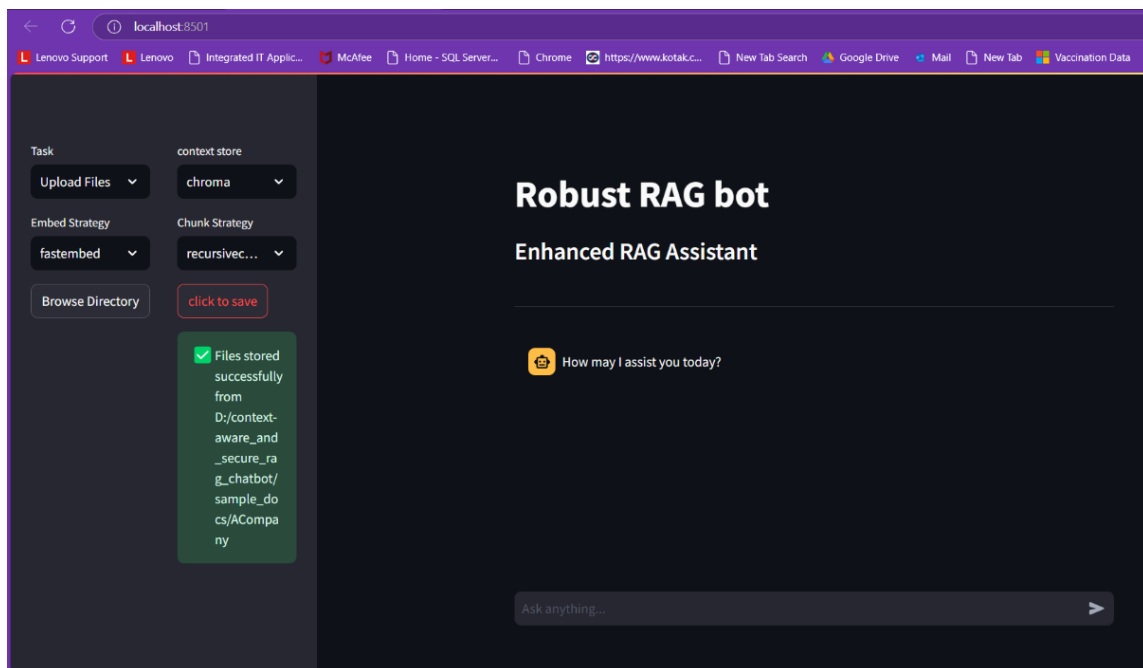


Figure 4.1 Successful Message of Upload Documents Model

Figure 4.1 shows the successful message with the path of the directory that is being chosen by the user. All the documents from this directory are stored in the specified context store. This message appears only when all the documents are stored successfully.

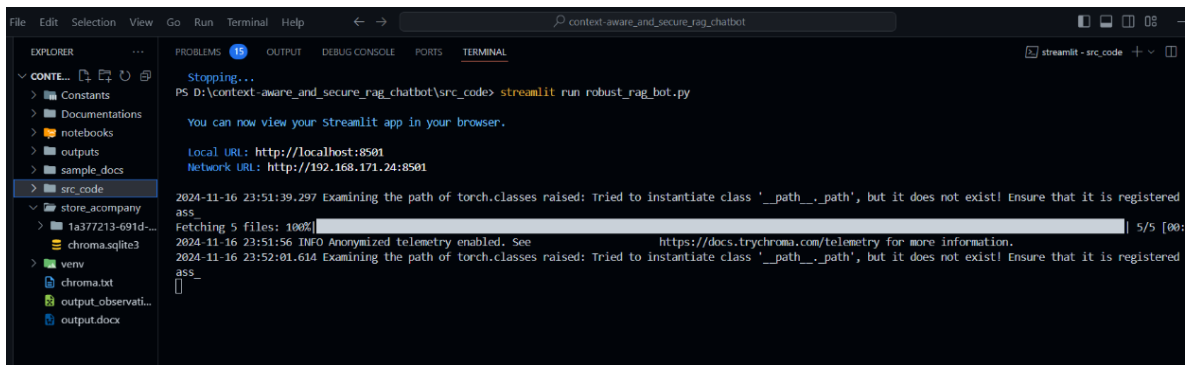


Figure 4.2 Successful Upload Documents in Chroma Vector Database

Figure 4.2 shows that the Chroma Vector Database named ‘store_acompany’ is created, where the uploaded documents is stored when the user uploads documents for the first time in the Chroma Vector Database.

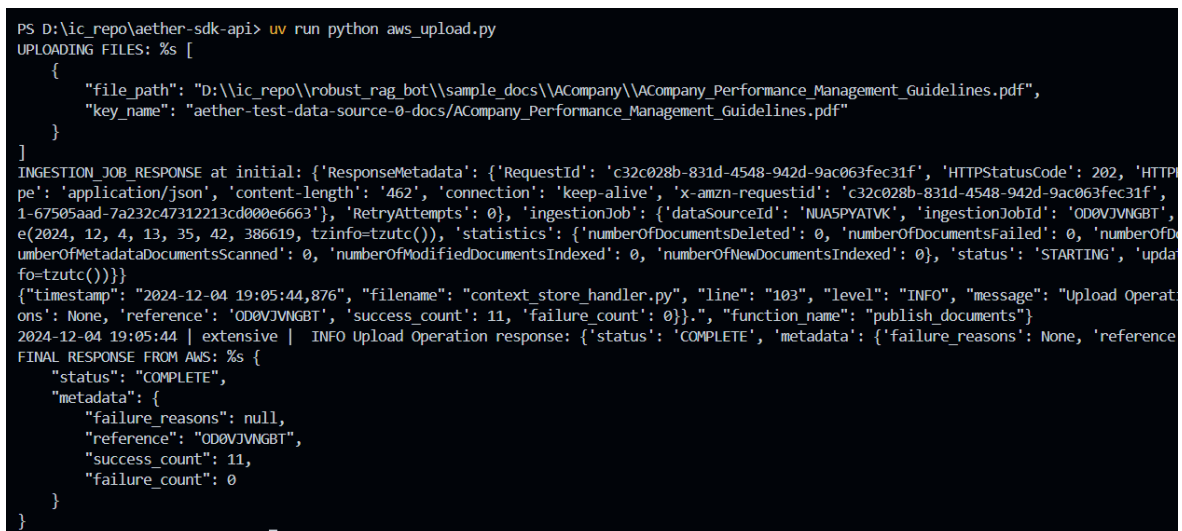
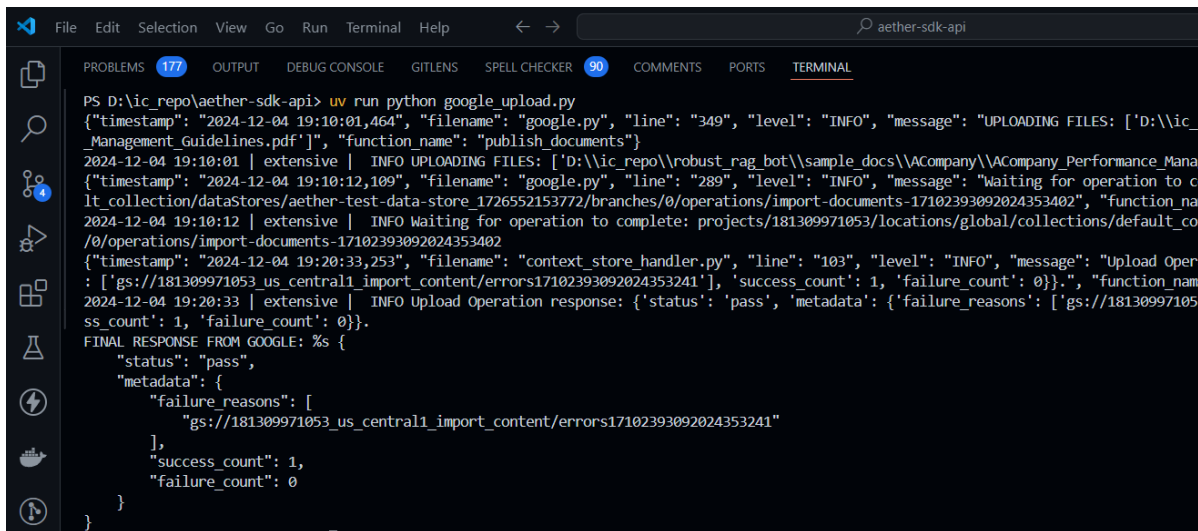


Figure 4.3 Successful Upload Documents in Amazon S3 Bucket

Figure 4.3 shows that the documents are ingested into Amazon S3 bucket successfully. The failure_count denotes number of documents that are failed to store and if it is not 0, its reasons will be listed in failure_reasons.



```
PS D:\ic_repo\ather-sdk-api> uv run python google_upload.py
{"timestamp": "2024-12-04 19:10:01,464", "filename": "google.py", "line": "349", "level": "INFO", "message": "UPLOADING FILES: ['D:\\ic_repo\\Management_Guidelines.pdf']", "function_name": "publish_documents"}
2024-12-04 19:10:01 | extensive | INFO UPLOADING FILES: ['D:\\ic_repo\\robust_rag_bot\\sample_docs\\ACompany\\ACompany_Performance_Management_Guidelines.pdf']
{"timestamp": "2024-12-04 19:10:12,109", "filename": "google.py", "line": "289", "level": "INFO", "message": "Waiting for operation to complete: projects/181309971053/locations/global/collections/default_collection/dataStores/aether-test-data-store-1726552153772/branches/0/operations/import-documents-17102393092024353402", "function_name": "publish_documents"}
2024-12-04 19:10:12 | extensive | INFO Waiting for operation to complete: projects/181309971053/locations/global/collections/default_collection/dataStores/aether-test-data-store-1726552153772/branches/0/operations/import-documents-17102393092024353402
{"timestamp": "2024-12-04 19:20:33,253", "filename": "context_store_handler.py", "line": "103", "level": "INFO", "message": "Upload Operation response: {'status': 'pass', 'metadata': {'failure_reasons': ['gs://181309971053_us_central1_import_content/errors17102393092024353241'], 'success_count': 1, 'failure_count': 0}}.", "function_name": "publish_documents"}
2024-12-04 19:20:33 | extensive | INFO Upload Operation response: {'status': 'pass', 'metadata': {'failure_reasons': ['gs://181309971053_us_central1_import_content/errors17102393092024353241'], 'success_count': 1, 'failure_count': 0}}.
ss_count': 1, 'failure_count': 0}}.
FINAL RESPONSE FROM GOOGLE: %s {
  "status": "pass",
  "metadata": {
    "failure_reasons": [
      "gs://181309971053_us_central1_import_content/errors17102393092024353241"
    ],
    "success_count": 1,
    "failure_count": 0
  }
}
```

Figure 4.4 Successful Upload Documents in GCS Bucket

Figure 4.4 shows that the documents are stored successfully in the GCS. The `failure_count` denotes the number of documents that are failed to store and if it is not 0, then the Google Storage location shown in `failure_reasons` contains the reasons for it.

4.2.2 Retrieve Model

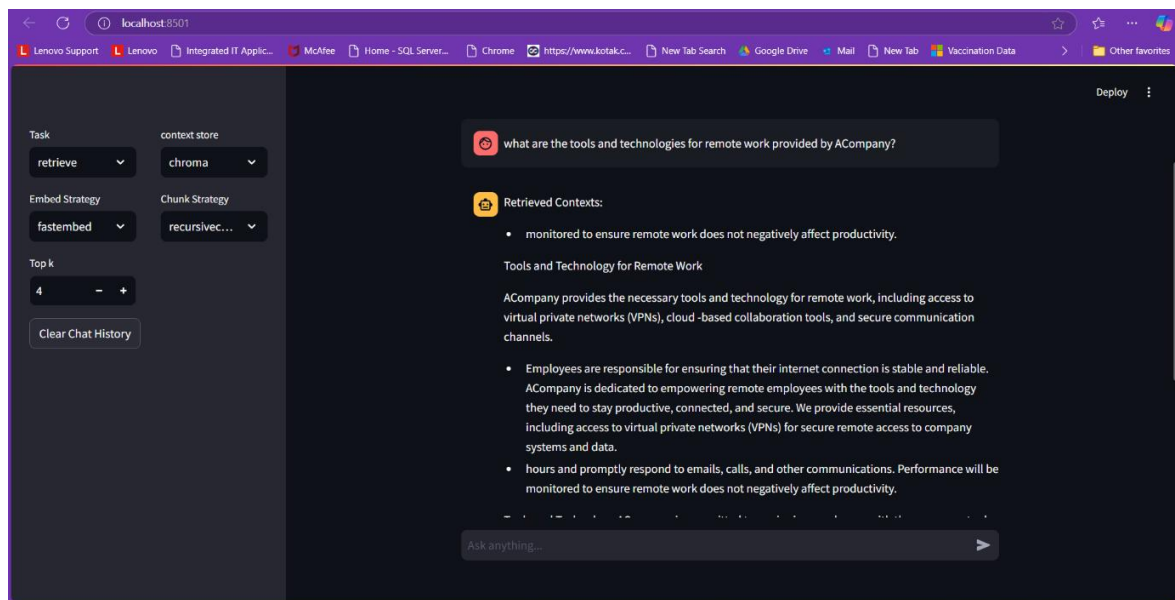


Figure 4.5 Retrieve Model response from Chroma Vector Database

This model would retrieve the relevant context from the specified context store with number of contexts to be retrieved is mentioned in the field 'Top k'.

Figure 4.5 shows the retrieved contexts from the Chroma vector database which is uploaded earlier shown in figure 4.2 and here the Embed Strategy field and Chunk Strategy field must be same as the one which is used during uploading. These two fields represent which embedding model should be used and which model should be used for splitting chunks.

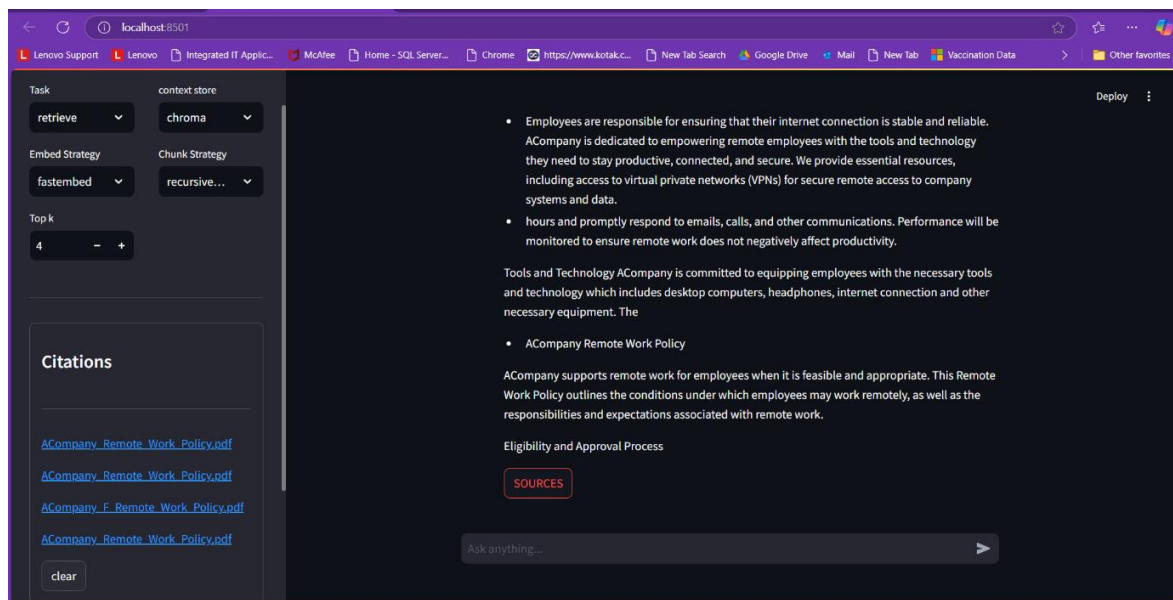


Figure 4.6 Sources of retrieve model response from Chroma Vector Database

Figure 4.6 shows the sources of the retrieved contexts from the Chroma vector database in the sidebar. Here sources mean that the name of the file from which the contexts are retrieved which is searched to be relevant for the prompt.

Figure 4.8 shows the retrieved context from Amazon S3 Bucket with its sources and the Uniform Resource Locator (URL) to navigate to the document from which the corresponding contexts are retrieved.

4.2.3 RAG Model

The RAG model are to allow the user not only to retrieve the relevant context from the context store but also to get meaning answer for the user prompt. This answer might be a summary or an answer for a query based on the user prompt.

This model searches the specified context store, then retrieves the relevant contexts and sends these contexts and the prompt to the LLM to generate a response.

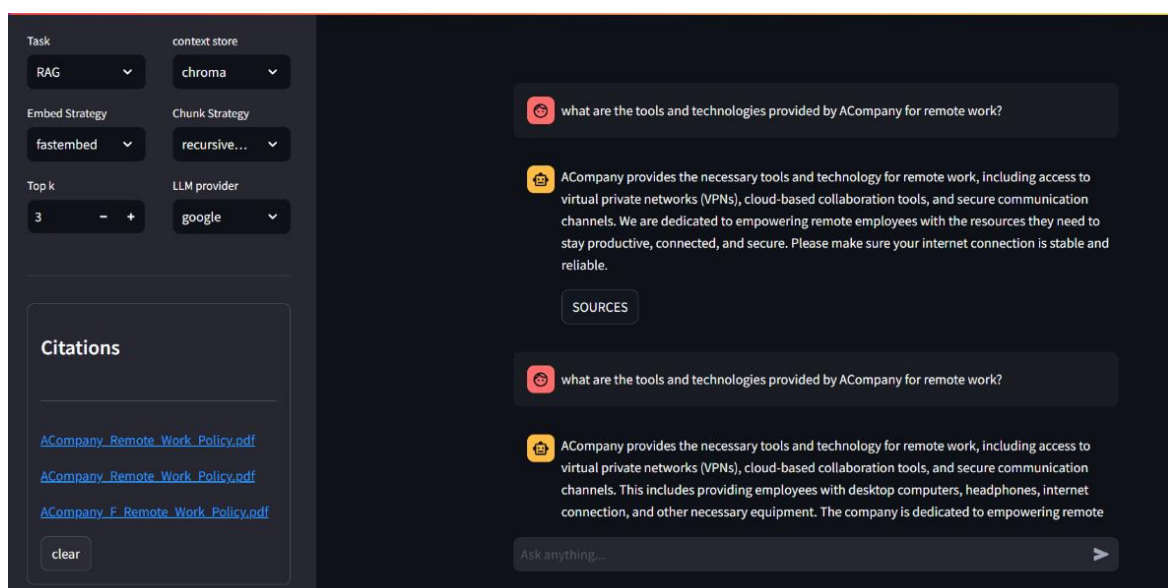


Figure 4.9 RAG Model Response and its Sources using Chroma Vector Database

Figure 4.9 shows the response and its sources generated using the Google's Gemini model and the contexts retrieved from Chroma vector database for the prompt given by the user. This Figure 4.9 demonstrates that if the database is corrupted, then there is a chance of getting corrupted response. Here One response is pure but the other one is corrupted for the same prompt given.

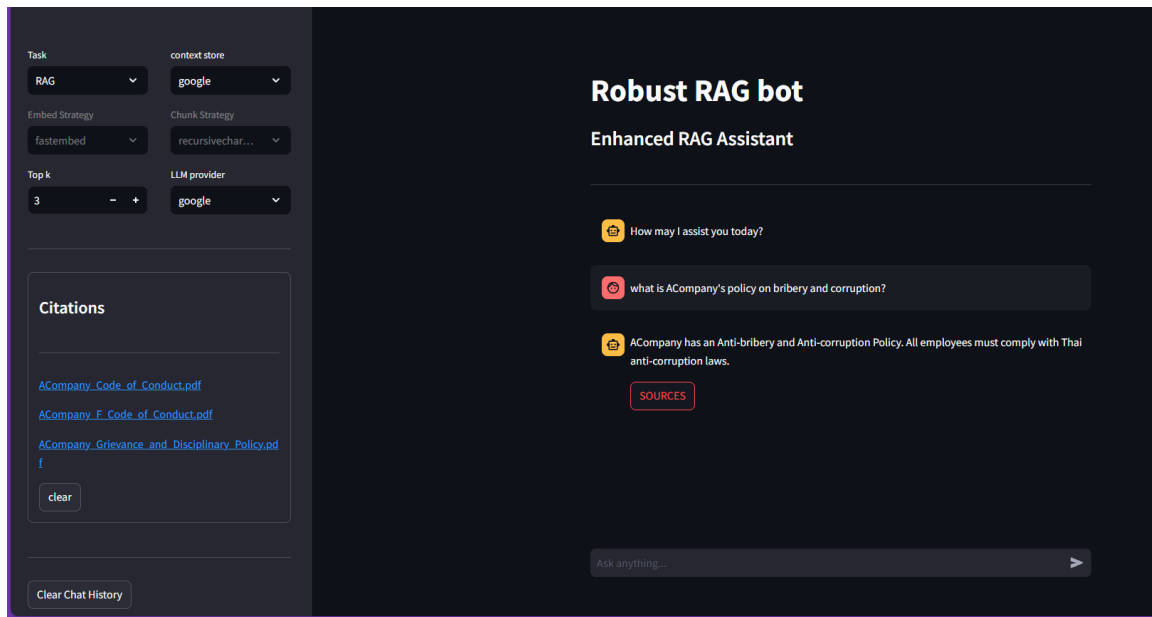


Figure 4.10 RAG Model Response and its Sources using GCS Bucket

Figure 4.10 shows the response from the Google's Gemini model based on the contexts retrieved from GCS for the prompt given by the user. Its sources are given in the sidebar and its accessible.

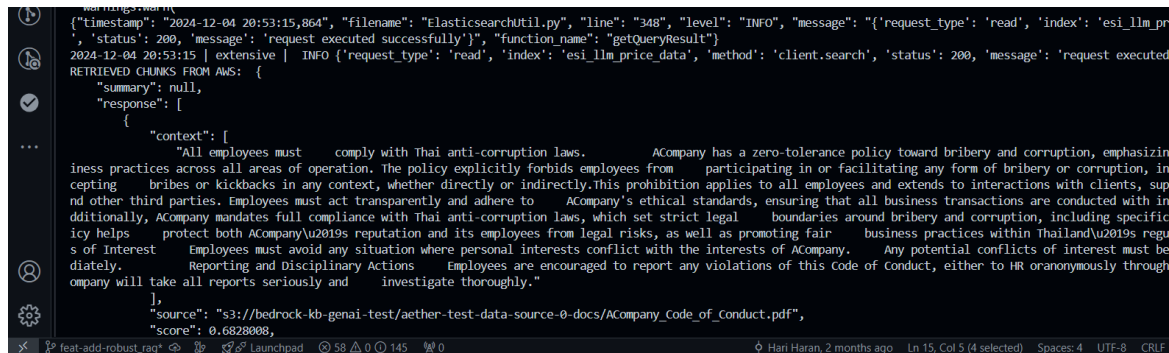


Figure 4.11 RAG Model Response's Sources using Amazon S3 Bucket

Figure 4.12 RAG Model Response using Amazon S3 Bucket

Figure 4.11 and 4.12 shows the response from the AWS' Claude Sonnet model based on the contexts retrieved from Amazon S3 Bucket for the prompt given by the user. Its sources are given in the sidebar and its accessible.

4.2.4 Robust RAG Model using Keywords

The robust RAG model enhance response generation by extracting keyword frequencies from potential contexts, filtering them based on the percentile threshold or constant threshold the user fixes, and incorporating the filtered keywords into prompts for robust and robust LLM-generated responses.

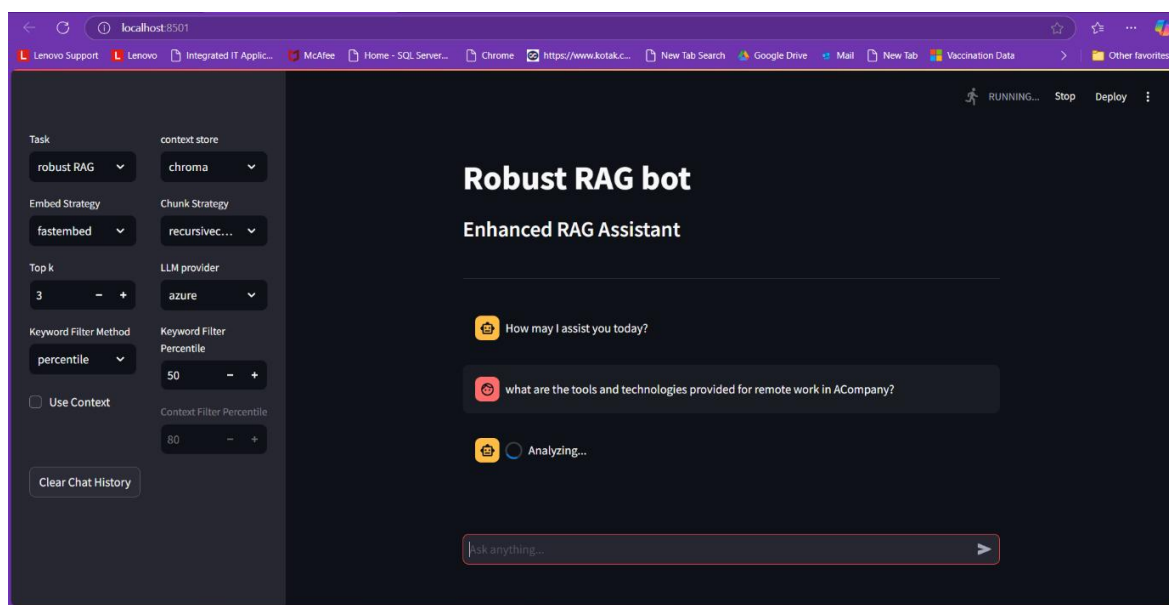


Figure 4.13 Robust RAG Model using Keywords' prompt and configuration using Chroma Vector Database.

Figure 4.13 shows the configuration for generating a robust response using keywords. The same configuration for retrieve model is followed here as the chosen context store is Chroma vector database.

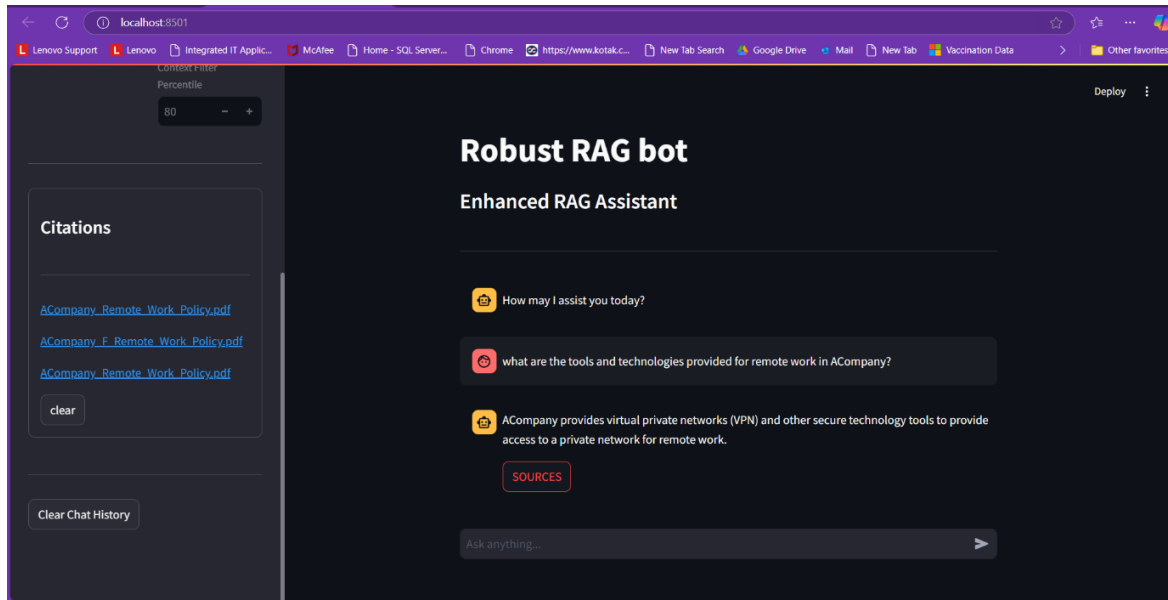


Figure 4.14 Robust RAG Model using Keywords' valid response and sources using Chroma Vector Database

The response is generate using Azure's OpenAI model. The keyword filter method says whether to use the default constant value to filter keywords or go with the percentile method and give a percentile value to filter the keywords. Its response and sources are shown in the Figure 4.14.

Figure 4.15 shows a sample response where the robust RAG model using keywords works inefficiently for limited keywords. Here the response is invalid since its generated one will be either placeholder response or not the one user expects.

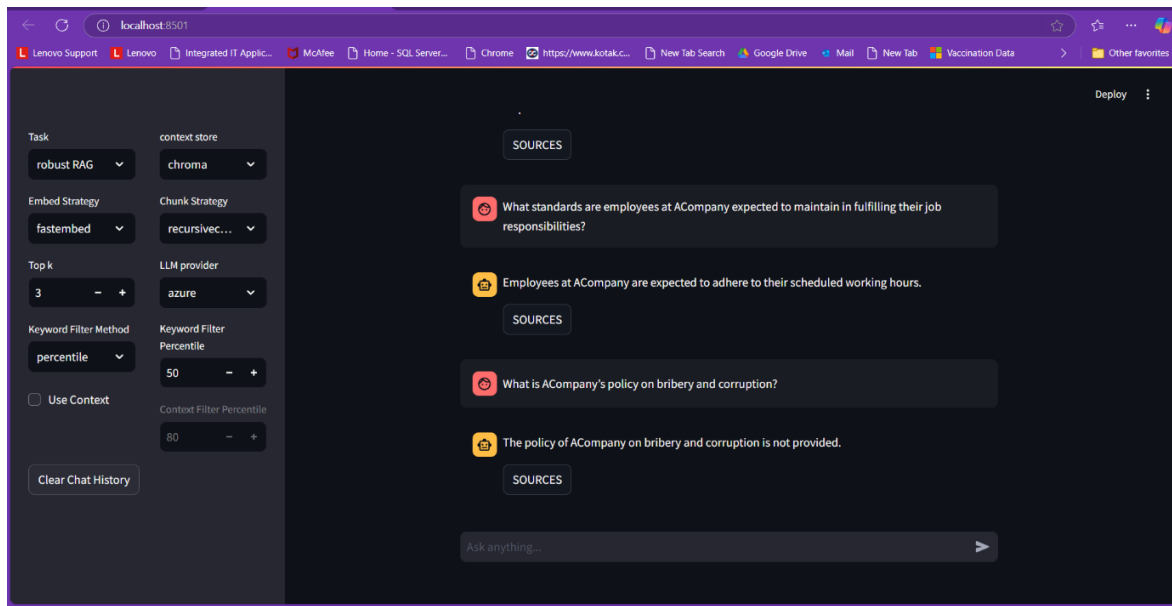


Figure 4.15 Robust RAG Model using Keywords' Invalid response using Chroma Vector Database

4.2.5 Robust RAG Model using Contexts

The robust RAG model using contexts enhances the response generated by the same model using keywords by scoring the documents based on the percentile to filter the contexts given by the user and then filtering them to get the most relevant and richer one, and finally incorporating the filtered contexts with the prompts for quality and promising responses.

Figure 4.16 shows a quality and a valid response generated by model which uses filtered contexts. Here, to note that for the same prompt, the robust RAG model using keywords gave just a invalid and a poor response.

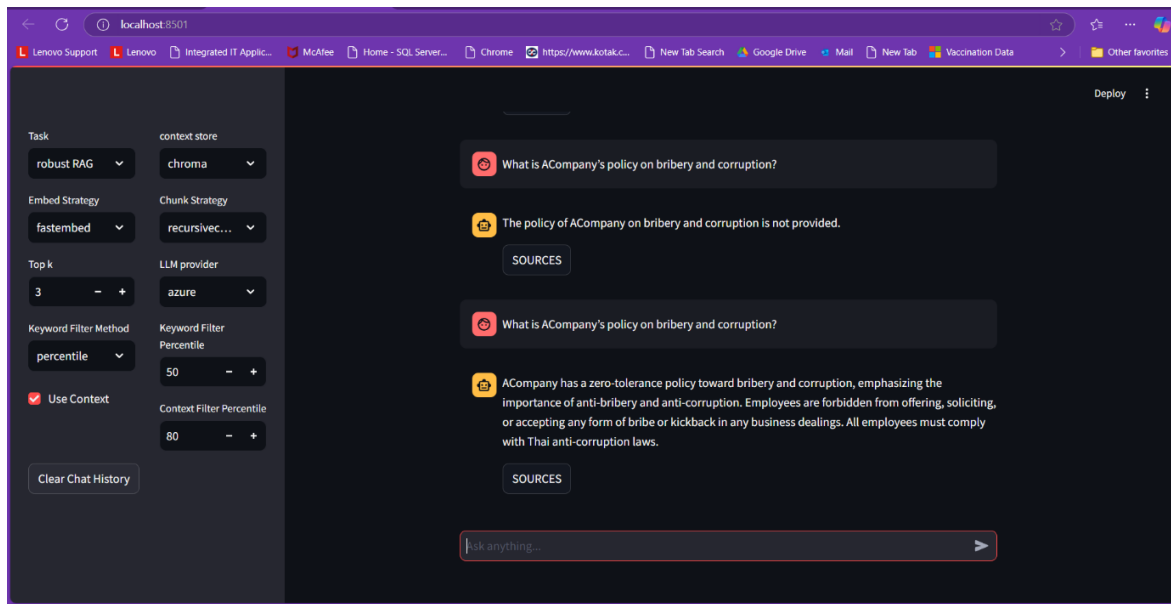


Figure 4.16 Robust RAG Model using Contexts' valid response using Chroma Vector Database

4.3 TESTING

Software testing is the process of evaluating a software application to ensure it works as intended, is free of bugs, and meets specified requirements. To validate the functionality, integration, and performance of the implemented system models, the unit testing, system testing and performance testing are applied.

4.3.1 Unit Testing

Unit Testing ensure the correctness of each individual sub-module by testing their functionalities in isolation. This includes verifying the accuracy of document processing, retrieval mechanisms, response generation, keyword filtering, and decoding strategies. The goal is to identify and fix any issues at the sub-module level before integration.

4.3.1.1 Upload Documents Model

- Test the Upload Documents model to ensure correct preprocessing, splitting, embedding creation, and storage in both cloud services and Chroma vector database.
- The upload documents model is tested with Chroma vector database, GCS using Google's Cloud services and Amazon S3 bucket using AWS's boto3 Software Development Kit (SDK).
- Its results are shown in Figure 4.2, Figure 4.3 and Figure 4.4 respectively. A successful message will be appeared upon successful storage and its result is Figure 4.1.

4.3.1.2 Retrieve Model

- Validate the Retrieve model for accurate and top-k context retrieval.
- The retrieve model is tested with Chroma vector database, GCS and Amazon S3 bucket as context store and to get top-k relevant context for the prompt and the sources of the retrieved contexts.
- Retrieve model response with Chroma is shown in Figure 4.5 and its sources in Figure 4.6. The result of retrieve model with GCS and Amazon S3 bucket is shown in Figure 4.7 and Figure 4.8 respectively with sources.

4.3.1.3 RAG Model

- Ensure RAG generates response from the retrieved contexts which is the result of retrieve model.
- RAG Model is tested with Chroma vector database, GCS and Amazon S3 bucket as context store and to get top-k relevant context for the prompt and the final response for the prompt is generated using the retrieved contexts.

- Figure 4.10 shows RAG Model response with GCS. It shows the sources of the retrieved contexts in the sidebar which is the result of retrieve model and then the contexts are given to the Large Language Model (LLM) to generate the final response for the prompt and it is shown in the same Figure 4.10 in the chat.
- Figure 4.11 and Figure 4.12 shows RAG Model response with Amazon S3 Bucket.
- Figure 4.9 shows RAG Model response using Chroma Vector database where once the response was pure and the other time, for the same prompt and configuration, the response was corrupted which is the nature of regular RAG model.

4.3.1.4 Robust RAG Model using Keywords

- Test Robust RAG using Keywords for accurate keyword extraction, frequency analysis, and robust response generation.
- The robust RAG model is tested with Chroma Vector database only since it has the corrupted context to test the working of robust RAG model and its results are shown in the Figure 4.13 and Figure 4.14.
- Its keyword extraction, frequency counting and filtering keywords are analysed as intermediary results and found to be correct and accurate in the terminal.
- This model has proven to produce pure response by the Figure 4.14. For which prompt, the regular RAG model failed to produce pure response, for the same prompt this model has produced pure response.
- However, this model using keywords is found to be working inefficient for which a sample prompt and response is shown in Figure 4.15.

4.3.1.5 Robust RAG Model using Contexts

- Check Robust RAG using Contexts for proper context scoring, filtering contexts and response quality.

- The same prompt for which the robust RAG model using keywords failed to produce a quality response, for the same prompt, this model using contexts has given a pure and quality response and its result is shown in Figure 4.16.
- Its context scoring, and then filtering them to get the most relevant context are analysed as intermediary results and identified to be correct and accurate in the terminal.

4.3.2 System Testing

System testing is a type of software testing where the entire system is tested as a whole to ensure it meets the specified requirements. It evaluates the system's functionality, performance, security, and compatibility in an end-to-end environment.

Here, system testing evaluates the complete system to ensure all models integrate seamlessly and deliver context-aware, high-quality responses. This phase validates end-to-end workflows, including document upload, context retrieval, response generation, and scoring mechanisms. It ensures that all modules collectively fulfil the system's requirements and deliver robust outputs under realistic use cases.

The system is built initially with each component as smaller units and then a user interface is built to provide the system as an agent using streamlit leaving AWS to be an independent module since it is not a high priority component and parked to integrate with the system for later use when needed.

The results of each component in the system testing are attached as screenshots which is labelled as figure from 4.1 to 4.17 in the Section 4.2 Result screenshot and proven that all the modules and components work well. Each of those results are explained in the Section 4.3.1 Unit Testing.

4.3.3 Performance Testing

Performance testing is a type of software testing that evaluates how a system performs under specific conditions, focusing on speed, scalability, responsiveness, and stability. It ensures the application can handle expected workloads effectively.

Here performance testing assesses the system's efficiency and scalability under varying workloads to ensure optimal performance in real-world scenarios. This includes evaluating latency, throughput, and the system's ability to scale with larger inputs and simultaneous user requests.

Table 4.1 Performance of Robust RAG Model

Robust Retrieve & Generate Model	Percentile	Valid and Pure	Invalid and Pure	Valid and Corrupted	Invalid and Corrupted
Keywords	85	7	8	0	0
Contexts	85	11	1	1	2
Keywords	80	8	7	0	0
Contexts	80	11	2	0	2

The robust RAG model using keywords and using contexts to assess which one works better and find its performance and attain optimal solutions. The table 4.1 has first column to be the model using keywords or contexts. The retrieve model should output only top 3 most relevant contexts since it is default count of retrieval in most cases and the experiment is intended to find the optimal values under this condition. ‘Percentile’ is chosen as the method to filter the keywords to fix the threshold based on the distribution of the keyword frequencies.

Here totally 15 different prompts were given to test both the model. The valid and pure column says that the count of the responses that are pure and quality. The

invalid and pure column says that the count of the responses that are pure but has placeholder response. The valid and corrupted column says that the count of the responses that are corrupted but responses are to be acceptable. The invalid and corrupted column says that the count of the responses that are corrupted and unacceptable.

Observation of Table 4.1:

- When there is equivalent number of benign contexts and corrupted contexts are provided for the model to respond, it produces response those are acceptable but still corrupted.
- The robust RAG model using contexts produces corrupted response only when the corrupted contexts are richer than the benign contexts.
- The model using keywords produces more invalid responses where the user might be annoyed in using the model, so the robust RAG model which produces high number of valid and pure responses compared to the one produced by the model using keywords and a smaller number of invalid responses is preferred and proven to be improvised better.

4.4 TEST CASES

Test cases are detailed, step-by-step instructions created to verify that a specific functionality of a software application works as expected. They define the test inputs, conditions, execution steps, and expected outcomes.

Reasons to Write Test Cases:

- **Ensure Coverage:** Helps cover all functional and non-functional requirements systematically.
- **Detect Bugs Early:** Identifies issues early in the development cycle, saving time and cost.

- **Standardized Testing:** Provides a clear and consistent guide for testers, especially in teams.
- **Facilitate Automation:** Serves as a base for creating automated test scripts.
- **Traceability:** Ensures each test maps to a specific requirement, helping in requirement validation.
- **Reusability:** Can be reused for regression testing and future versions of the software.
- **Improved Quality:** Enhances the overall quality of the software by ensuring rigorous testing.

As shown in table 4.2, this test plan covers the following models:

- **Upload Documents Model:** The test cases for this model validates the ability to upload, process, and store documents in the system's vector database.
- **Retrieve Model:** The test cases for this model ensures relevant contexts are accurately retrieved from the Chroma vector database. The test cases also ensure accurate retrieval of contexts and their sources from GCS and AWS S3 bucket.
- **RAG Model:** The test cases assess the integration of retrieval and generation mechanisms to produce meaningful responses.
- **Robust RAG Model Using Keywords:** The test case validates the improved generation mechanism that filters keywords for greater response accuracy.
- **Robust RAG Model Using Contexts:** This test case validates that the enhanced retrieval and generation mechanism that filters contexts based on most frequent keywords to produce optimal and valid responses.

Table 4.2 Test Cases

MODELS	TEST CASE SUMMARY	EXPECTED RESULT	ACTUAL RESULT	STATUS (PASS/ FAIL)
Upload Documents Model	Select a valid directory and upload files from the directory into Chroma Vector Database	The model processes files correctly, splits them into chunks, embeds them, and stores them in Chroma. The database is created at the specified path.	Files are processed as expected, chunks are embedded, stored in Chroma. Database is created and verified.	PASS
	Select a list of files and upload files into GCS	The model processes files correctly and store them in specified GCS bucket.	Files are stored as expected, and response returned with number of files uploaded as success count.	PASS
	Select a list of files and upload files into Amazon S3 Bucket.	The model processes files correctly and store them in specified Amazon S3 Bucket.	Files are stored as expected, and response returned with number of files uploaded and existing files count as success count.	PASS
	Select no directory and upload files	A warning message is displayed, and the system remains functional without disrupting the workflow.	Warning message displayed as expected. The system did not crash or interrupt the workflow.	PASS
Retrieve Model	Retrieve top k relevant contexts for a valid prompt from the Chroma Database	The model retrieves the top k relevant contexts from the Chroma database based on the prompt.	The top k relevant contexts were retrieved and displayed on the UI with their sources in the sidebar.	PASS

Table 4.2 Test Cases (Continued)

MODELS	TEST CASE SUMMARY	EXPECTED RESULT	ACTUAL RESULT	STATUS (PASS/FAIL)
Retrieve Model	Retrieve top k relevant contexts for a valid prompt from GCS	The model retrieves the top k relevant contexts and return their sources.	The top k relevant contexts were retrieved, and sources displayed in the sidebar.	PASS
	Retrieve top k relevant contexts for a valid prompt from Amazon S3 Bucket	The model retrieves the top k relevant contexts and return their sources.	The top k relevant contexts were retrieved, and sources were displayed in the sidebar.	PASS
RAG Model	Generate a response with relevant contexts using Chroma as the context store	The model retrieves relevant contexts from Chroma and generates a valid response using the specified LLM.	Contexts were retrieved, and a valid response was generated as expected.	PASS
	Generate a corrupted response from corrupted contexts using Chroma as the context store	The model generates a corrupted response due to retrieval of corrupted context.	The response was corrupted as expected, highlighting the limitations of regular RAG models.	PASS
	Generate a response with relevant contexts using GCS as the context store	The model retrieves relevant contexts from GCS bucket and generates a valid response using the LLM.	Contexts were retrieved, and a valid response was generated as expected.	PASS

Table 4.2 Test Cases (Continued)

MODELS	TEST CASE SUMMARY	EXPECTED RESULT	ACTUAL RESULT	STATUS (PASS/FAIL)
RAG Model	Generate a response using contexts from Amazon S3 Bucket as the context store	The model retrieves contexts from Amazon S3 bucket and returns a valid response using the LLM.	Contexts were retrieved, and a valid response was generated as expected.	PASS
Robust RAG Model using keywords	Generate a valid response using the constant method	The model filters contexts using keywords and generates a precise and valid response.	The response was valid and precise, as expected.	PASS
	Generate a valid response using the percentile method	The model filters contexts using keywords and generates a precise and valid response.	The response was valid and precise, as expected.	PASS
	Generate an invalid response with insufficient keywords	The model generates a placeholder response indicating insufficient information.	An invalid response such as "No specific information provided" or useless information was generated.	PASS
Robust RAG Model using Contexts	Generate a valid response with context percentile	The model uses the specified context percentile (default: 80) and generates a sensible response.	The response was meaningful and accurate, exceeding the performance of the keyword-based model.	PASS