SYSTEM DESIGN

3.1 SYSTEM ARCHITECTURE

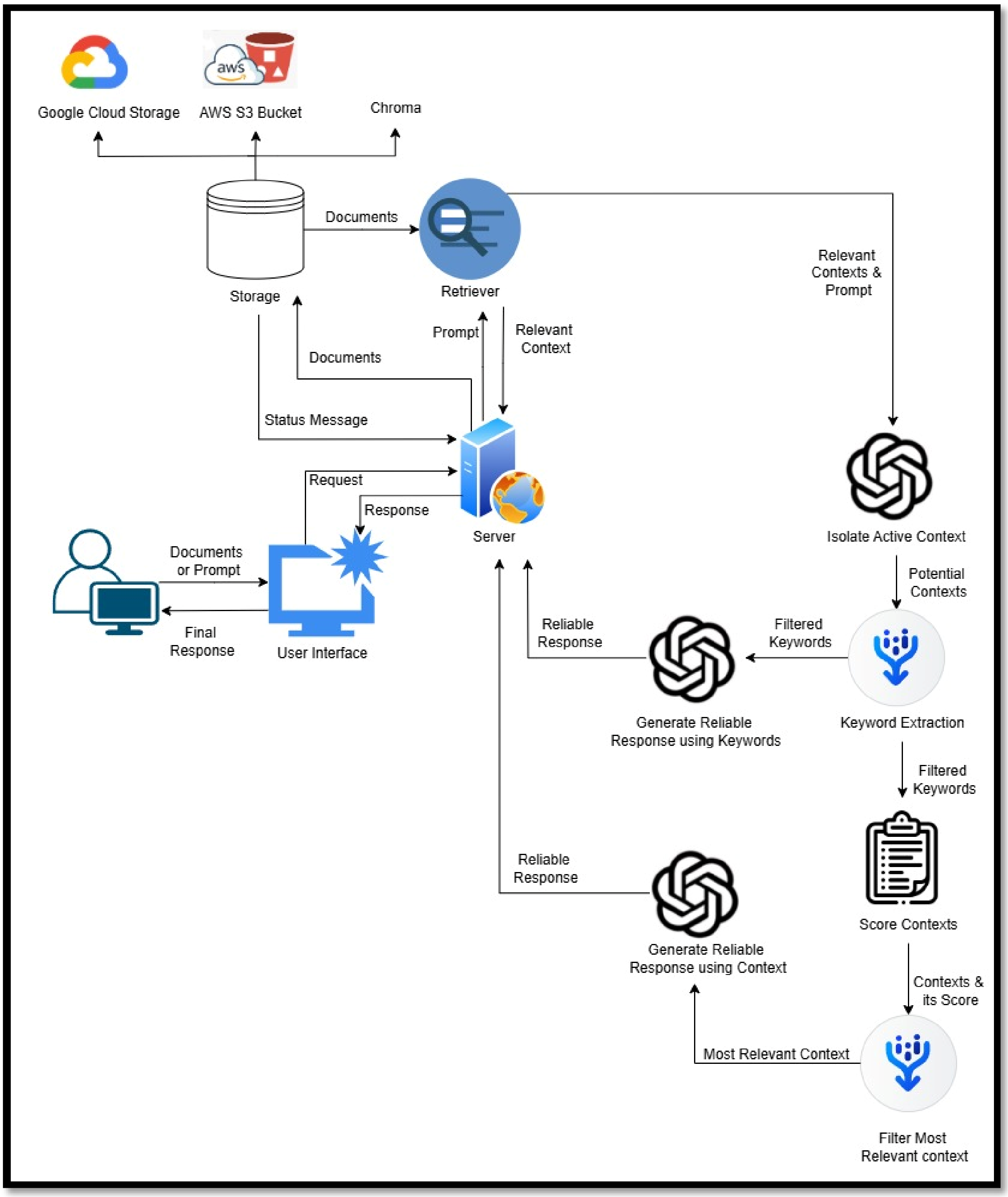


Figure 3.1 System Architecture

The architecture diagram in Figure 3.1 provides a comprehensive overview of the system for a retrieval-augmented chatbot that integrates context filtering, reliable response generation, and seamless document storage and retrieval. Here's a detailed description of the components:

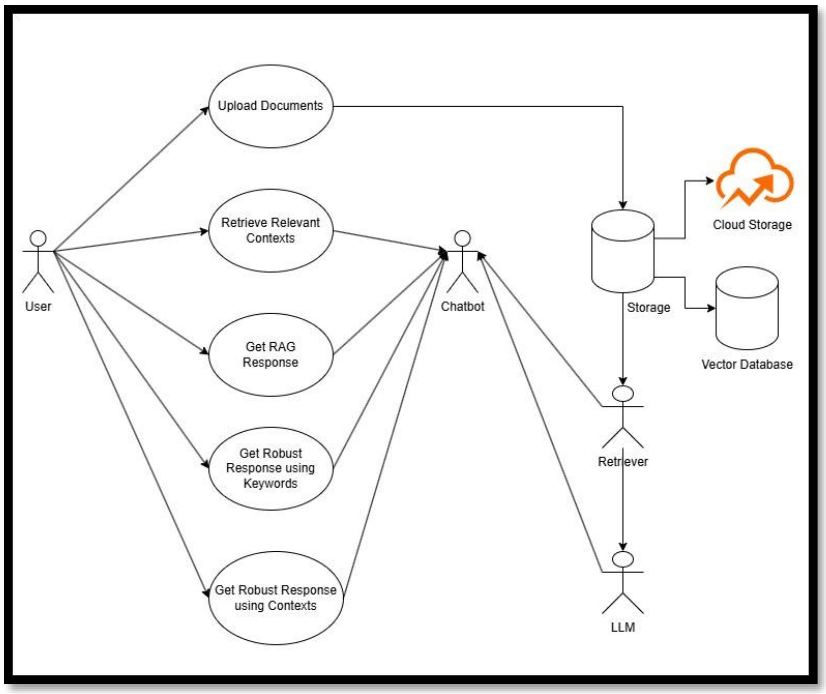
- Storage and Document Management: The system stores documents in external storage services such as Google Cloud Storage, Amazon S3 Bucket, or Chroma.
- Retriever Component: This component acts as a bridge between the storage system and the response generator and receives a user prompt and retrieves relevant contexts from the stored documents.
- Server: The server orchestrates the entire workflow by managing requests from the user interface and coordinating with other components. It handles incoming documents or prompts from the user, forwards them to the retriever, and manages the response generation pipeline.
- Keyword Extraction and Context Scoring: Extracts potential keywords from the retrieved contexts and the most relevant context is selected based on context scores for generating precise and accurate responses.
- Response Generation: It uses filtered keywords or most relevant contexts to ensure promising response.
- User Interface: Accepts user inputs, such as documents or prompts, and displays the final response after processing.

3.2 UML DIAGRAMS

Unified Modelling Language (UML) diagrams are visual representations used to model and document the design of software systems. They provide a standardized way to visualize, design, and document complex systems, enhancing communication and clarity among stakeholders. For this project, the following UML diagrams have been created to represent various aspects of the system:

- Use Case Diagram: Highlights the interactions between users and the system.
- Sequence Diagram: Depicts the flow of messages and interactions between system components.
- Activity Diagram: Illustrates the workflow or processes within the system.
- Class Diagram: Describes the system's structure, including its classes and relationships.
- State Machine Diagram: Represents the states and transitions of a specific system component.

3.2.1 Use Case Diagram



Upload Documents
Retrieve Relevant Contexts
Get RAG Response
Get Robust Response using Keywords
Get Robust Response using Contexts
User
Chatbot
Storage
Cloud Storage
Vector Database
Retriever
LLM

Figure 3.2 Use case Diagram

This use case diagram in Figure 3.2 illustrates the interaction between a user, a chatbot, a retriever, and backend storage systems in an AI-based system. The user can upload documents or provide a prompt, which the chatbot processes.

The chatbot retrieves relevant contexts from a vector database or cloud storage using the retriever.

Based on the retrieved contexts, the system supports multiple functionalities: generating a regular response, refining it using keyword-based filtering, or generating response by most relevant context.

3.2.2 Activity Diagram

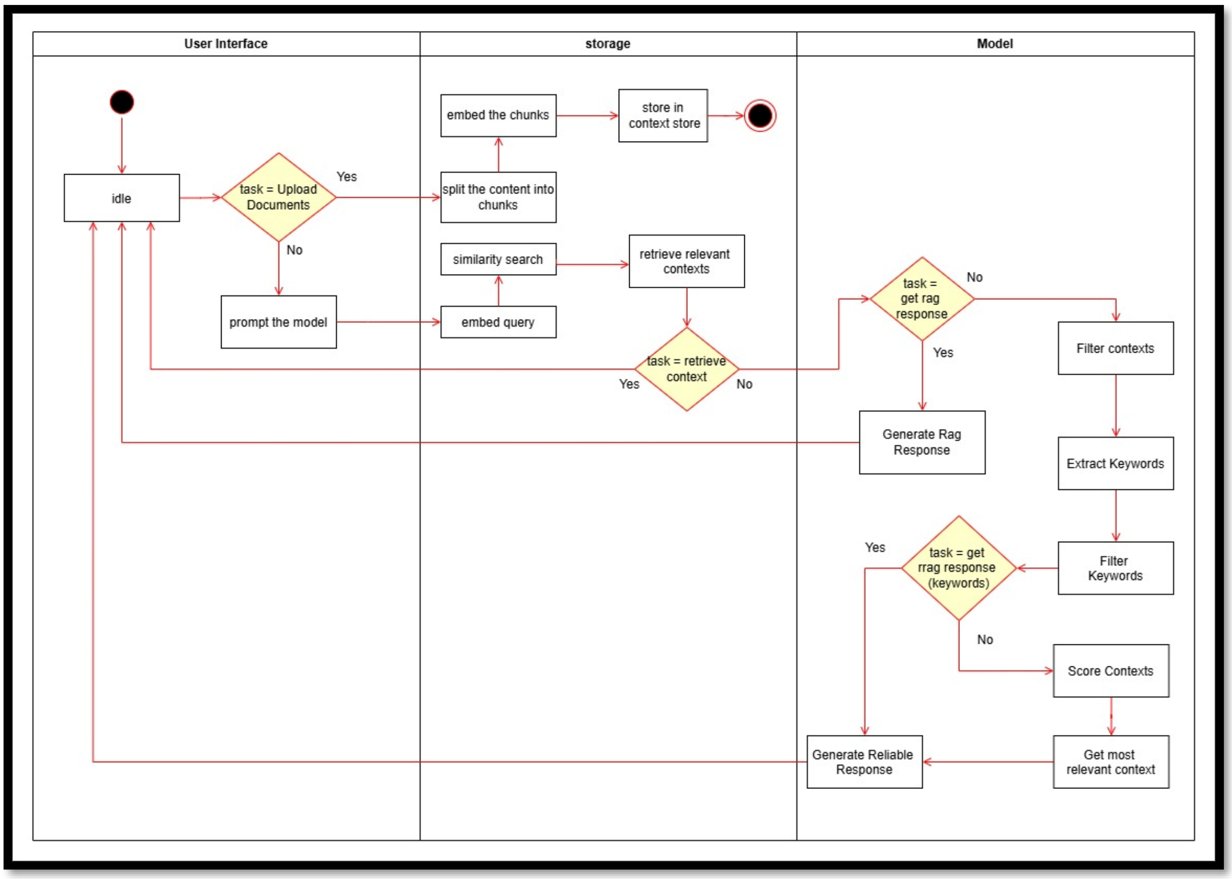


Figure 3.3 Activity Diagram

An activity diagram is a type of UML diagram that represents the flow of activities or processes in a system. This activity diagram, which is in Figure 3.3 shows how the system processes user-uploaded documents or queries to provide accurate responses. Users can upload documents or ask questions directly through the interface. Uploaded documents are divided into smaller parts, embedded, and stored in a database. When a query is made, the system searches for similar contexts in the database, according to the user's desired task, it either sends back the contexts or generate regular RAG response for the query or generate reliable response using high frequent keywords or filters and scores them using keywords, and selects the most relevant context to generate a more reliable response and finally the generated response is sent back to the user.

3.2.3 Sequence Diagram

A sequence diagram represents the flow of interactions between different components in a system over time.

The sequence diagram in Figure 3.4 shows the flow of interactions in upload documents, retrieve, retrieve and generate and reliable retrieve and generate models. Here is the explanation of the sequence diagram shown in Figure 3.4:

- User initiates the process by uploading documents or providing a prompt.
- If documents are uploaded, then they stored in the context store.
- When the user provides a prompt, the prompt is sent to the retriever to get relevant context.
- If the user only needs the context, the contexts are sent back else proceds further.
- The relevant contexts are filtered, and keywords are extracted and then keywords are filtered.
- If the user wants to use only keywords, then a LLM generates a response based on the keywords and returns back to the user.

- Else contexts are scored and the most relevant one is selected. The LLM then generates a reliable response based on the selected context and the response is sent back.

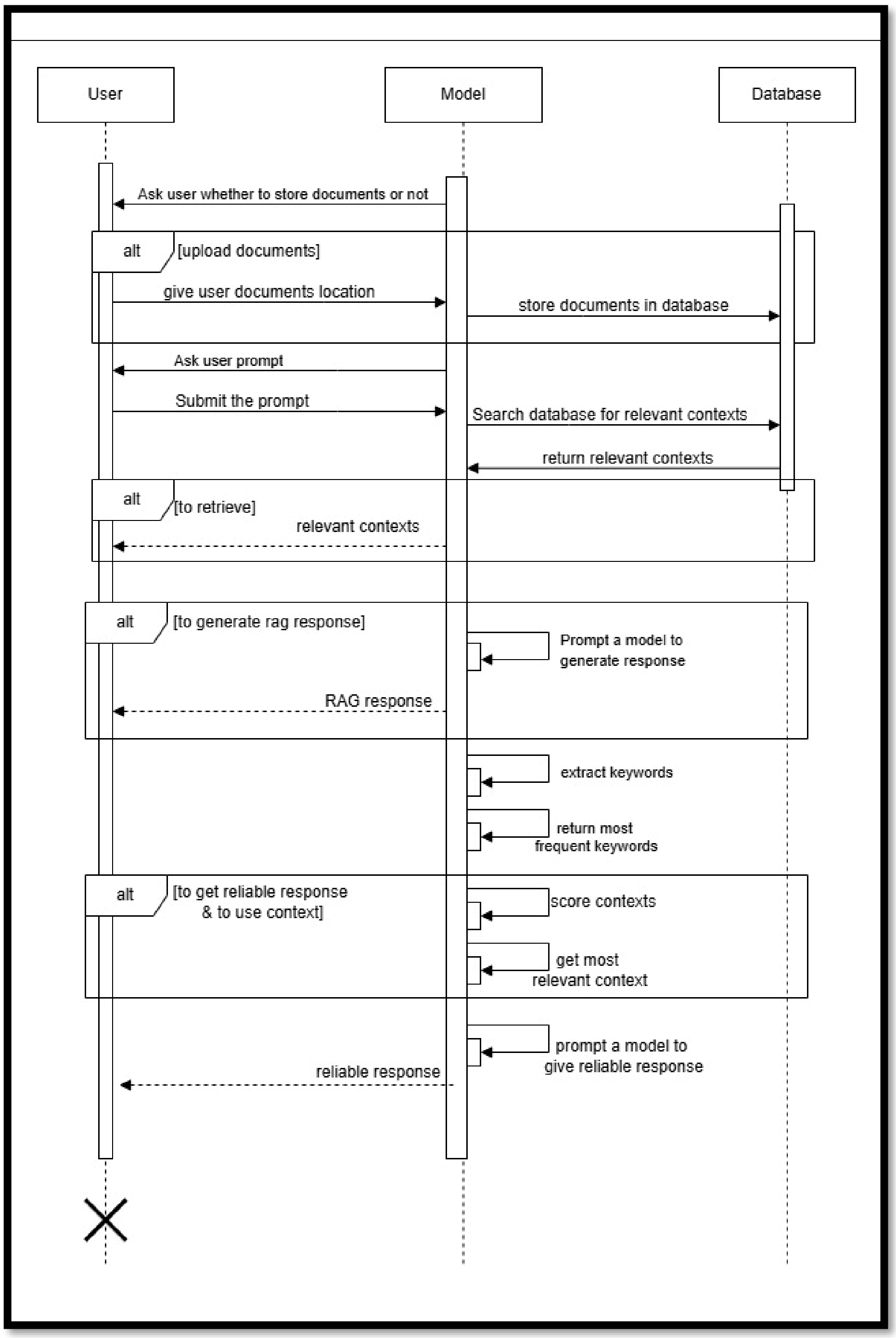

User
Model
Database
Ask user whether to store documents or not
alt
[upload documents]
give user documents location
store documents in database
Ask user prompt
Submit the prompt
Search database for relevant contexts
return relevant contexts
alt
[to retrieve]
relevant contexts
alt
[to generate rag response]
Prompt a model to generate response
RAG response
extract keywords
return most frequent keywords
alt
[to get reliable response & to use context]
score contexts
get most relevant context
prompt a model to give reliable response
reliable response

Figure 3.5 Sequence Diagram

3.2.4 Class Diagram

A class diagram is a type of Unified Modelling Language (UML) diagram that visually represents the structure of a system by showing its classes, attributes, methods, and the relationships between those classes. It's widely used in software design and modelling to understand and communicate the architecture of object-oriented systems.

The class diagram in Figure 3.5 shows the class diagram of upload documents, retrieve, retrieve and generate and reliable retrieve and generate models. The purpose of each class in this class diagram is given below:

- GenerateResponse: Handles the core retrieve and generate process, including retrieving contexts, generating regular response using retrieved contexts, and generating reliable responses tailored to user prompt.
- BaseContextStore: An abstract class providing a blueprint for context retrieval, implemented by specific context providers like Chroma, AWS and Google.
- AwsBedrock, Google, and ChromaContextStore: These classes store or fetch contexts from Amazon S3 Bucket, Google Cloud Storage, and vector database like Chroma, respectively.
- RobustRag: Enhances response reliability through keyword extraction, filtering keywords, or further scoring and filtering contexts to ensure relevance and accuracy.
- LlmResponseGenerator: Integrates with various LLM providers (Google, Azure, AWS) to generate responses based on retrieved contexts.
- Relationships**:** Classes like AWSBedrock and Google inherit from BaseContextStore which is generalization relationship, while GenerateResponse depends on these and other classes like RobustRag and LlmResponseGenerator to fetch, process, and generate outputs which is dependency relationship.

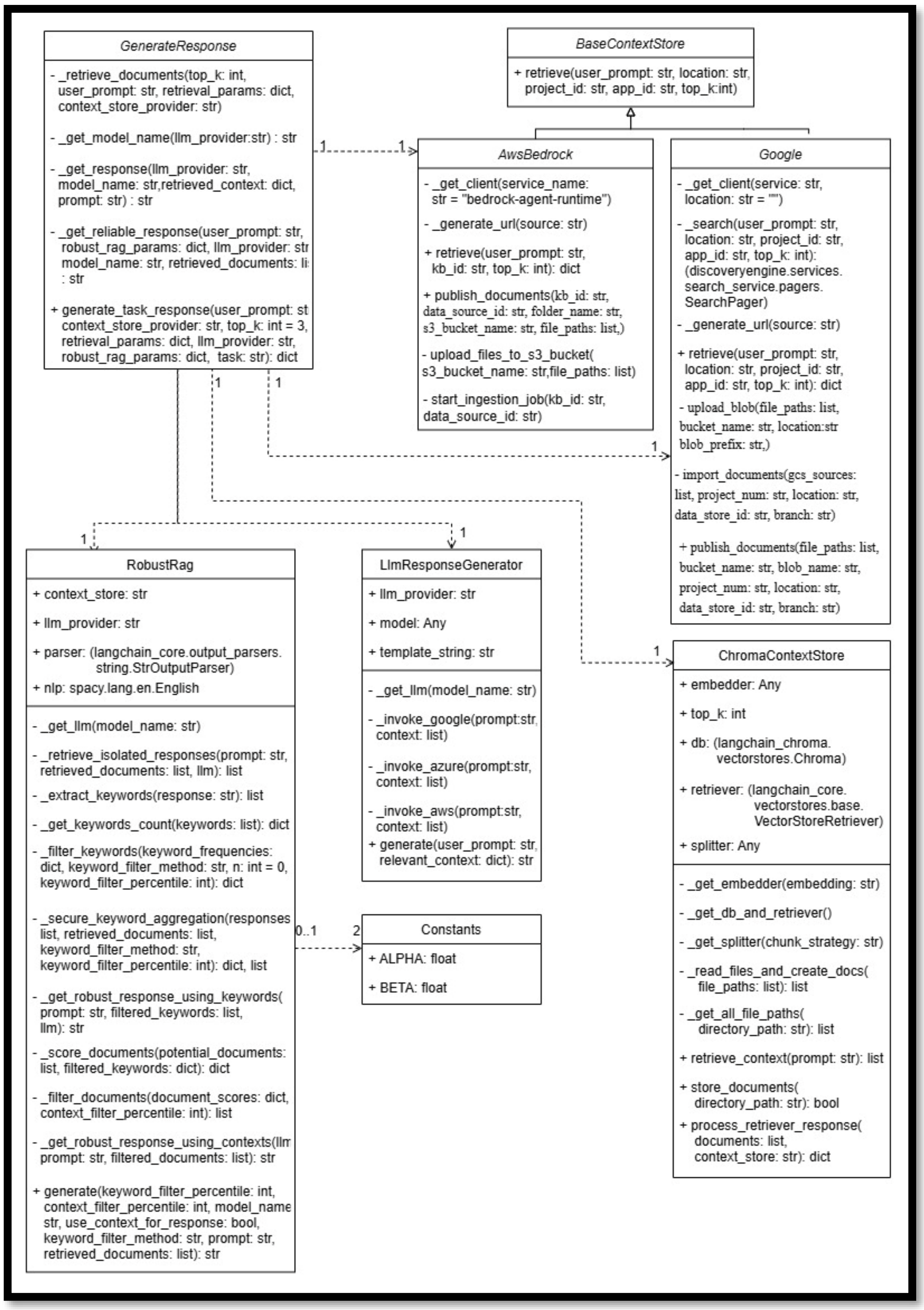

GenerateResponse
- _retrieve_documents(top_k: int, user_prompt: str, retrieval_params: dict, context_store_provider: str)
- _get_model_name(llm_provider:str) : str
- _get_response(llm_provider: str, model_name: str,retrieved_context: dict, prompt: str) : str
- _get_reliable_response(user_prompt: str, robust_rag_params: dict, llm_provider: str model_name: str, retrieved_documents: li : str
+ generate_task_response(user_prompt: st context_store_provider: str, top_k: int = 3, retrieval_params: dict, llm_provider: str, robust_rag_params: dict, task: str): dict
BaseContextStore
+ retrieve(user_prompt: str, location: str, project_id: str, app_id: str, top_k:int)
AwsBedrock
- _get_client(service_name: str = "bedrock-agent-runtime")
- _generate_url(source: str)
+ retrieve(user_prompt: str, kb_id: str, top_k: int): dict
+ publish_documents(kb_id: str, data_source_id: str, folder_name: str, s3_bucket_name: str, file_paths: list,)
- upload_files_to_s3_bucket(s3_bucket_name: str,file_paths: list)
- start_ingestion_job(kb_id: str, data_source_id: str)
Google
- _get_client(service: str, location: str = "")
- _search(user_prompt: str, location: str, project_id: str, app_id: str, top_k: int): (discoveryengine.services. search_service.pagers. SearchPager)
- _generate_url(source: str)
+ retrieve(user_prompt: str, location: str, project_id: str, app_id: str, top_k: int): dict
- upload_blob(file_paths: list, bucket_name: str, location:str blob_prefix: str,)
- import_documents(gcs_sources: list, project_num: str, location: str, data_store_id: str, branch: str)
+ publish_documents(file_paths: list, bucket_name: str, blob_name: str, project_num: str, location: str, data_store_id: str, branch: str)
1
1
1
1
1
1
1
1
RobustRag
+ context_store: str
+ llm_provider: str
+ parser: (langchain_core.output_parsers. string.StrOutputParser)
+ nlp: spacy.lang.en.English
- _get_llm(model_name: str)
- _retrieve_isolated_responses(prompt: str, retrieved_documents: list, llm): list
- _extract_keywords(response: str): list
- _get_keywords_count(keywords: list): dict
- _filter_keywords(keyword_frequencies: dict, keyword_filter_method: str, n: int = 0, keyword_filter_percentile: int): dict
- _secure_keyword_aggregation(responses list, retrieved_documents: list, keyword_filter_method: str, keyword_filter_percentile: int): dict, list
- _get_robust_response_using_keywords(prompt: str, filtered_keywords: list, llm): str
- _score_documents(potential_documents: list, filtered_keywords: dict): dict
- _filter_documents(document_scores: dict, context_filter_percentile: int): list
- _get_robust_response_using_contexts(llm prompt: str, filtered_documents: list): str
+ generate(keyword_filter_percentile: int, context_filter_percentile: int, model_name str, use_context_for_response: bool, keyword_filter_method: str, prompt: str, retrieved_documents: list): str
LlmResponseGenerator
+ llm_provider: str
+ model: Any
+ template_string: str
- _get_llm(model_name: str)
- _invoke_google(prompt:str, context: list)
- _invoke_azure(prompt:str, context: list)
- _invoke_aws(prompt:str, context: list)
+ generate(user_prompt: str, relevant_context: dict): str
0..1
2
Constants
+ ALPHA: float
+ BETA: float
ChromaContextStore
+ embedder: Any
+ top_k: int
+ db: (langchain_chroma. vectorstores.Chroma)
+ retriever: (langchain_core. vectorstores.base. VectorStoreRetriever)
+ splitter: Any
- _get_embedder(embedding: str)
- _get_db_and_retriever()
- _get_splitter(chunk_strategy: str)
- _read_files_and_create_docs(file_paths: list): list
- _get_all_file_paths(directory_path: str): list
+ retrieve_context(prompt: str): list
+ store_documents(directory_path: str): bool
+ process_retriever_response(documents: list, context_store: str): dict

Figure 3.5 Class Diagram

3.2.5 State Machine Diagram

A state machine diagram for this model shown in Figure 3.6 illustrates various states of the system can be in, such as idle, processing query, and generating response.

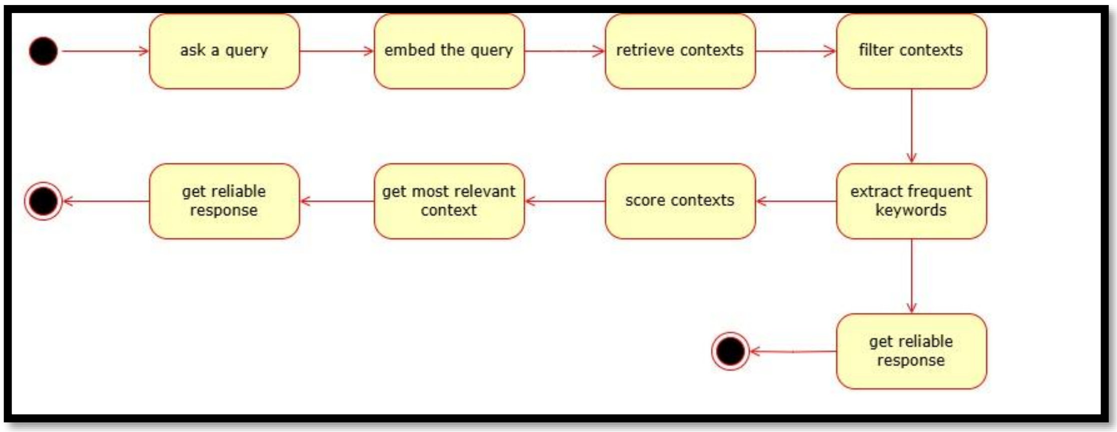


Figure 3.6 State Machine Diagram of Robust RAG model

Initially, the model is in idle state, then when a user prompt and query is embedded, then the system is in processing query state, then until the keywords are extracted from contexts, it is under processing contexts state, then until the final response is generated, the system is in generating response state.

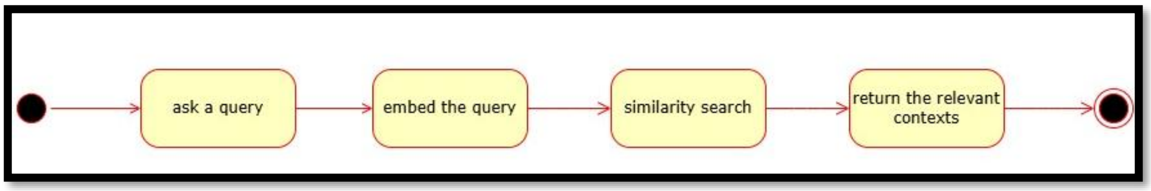


Figure 3.7 State Machine Diagram of Retriever Model

The state machine diagram for the retriever in RAG systems depicts the states involved in retrieving relevant contexts, such as idle, query received, processing query, searching, context retrieved and finally idle again. Each of these states corresponds to the component in Figure 3.7 from start notation to end notation respectively.

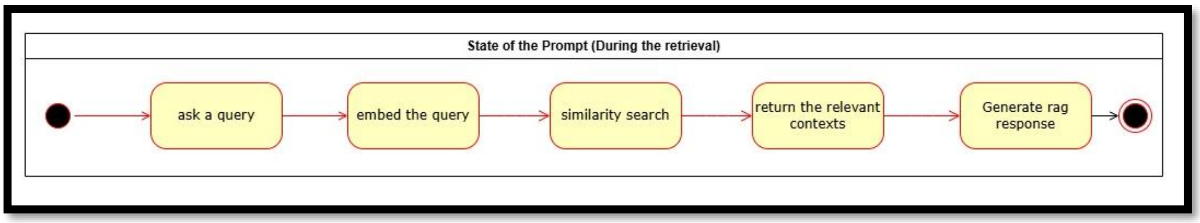

State of the Prompt (During the retrieval)
ask a query
embed the query
similarity search
return the relevant contexts
Generate rag response

Figure 3.8 State Machine Diagram of RAG Model

A RAG (Retrieval-Augmented Generation) model's state diagram depicts the flow from a user query: Query Processing embeds the user input. This leads to Retrieval, finding relevant information from an external knowledge base. Finally, Generation combines the query and retrieved context to produce a refined, informed answer.

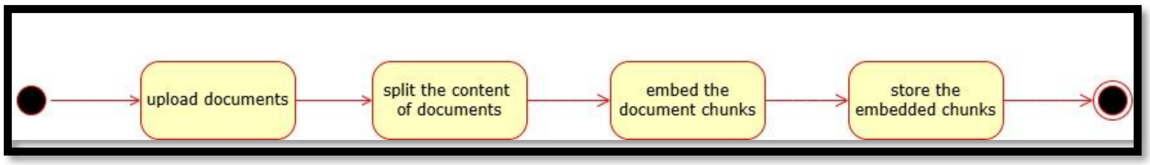

upload documents
split the content of documents
embed the document chunks
store the embedded chunks

Figure 3.9 State Machine Diagram of Upload Documents

The state machine diagram for storing documents in a context store outlines the states involved in the process, such as idle, document received, splitting, embedding, storing and again idle. It shows the transitions between these states, ensuring that documents are properly processed, embedded, and stored for future retrieval and response generation. Each of these states corresponds to each component in Figure 3.9 from start notation to end notation respectively with start and end notation representing idle state.