Designing a chatbot using LLM Model

And

Designing Agent Mechanism using AWS Bedrock

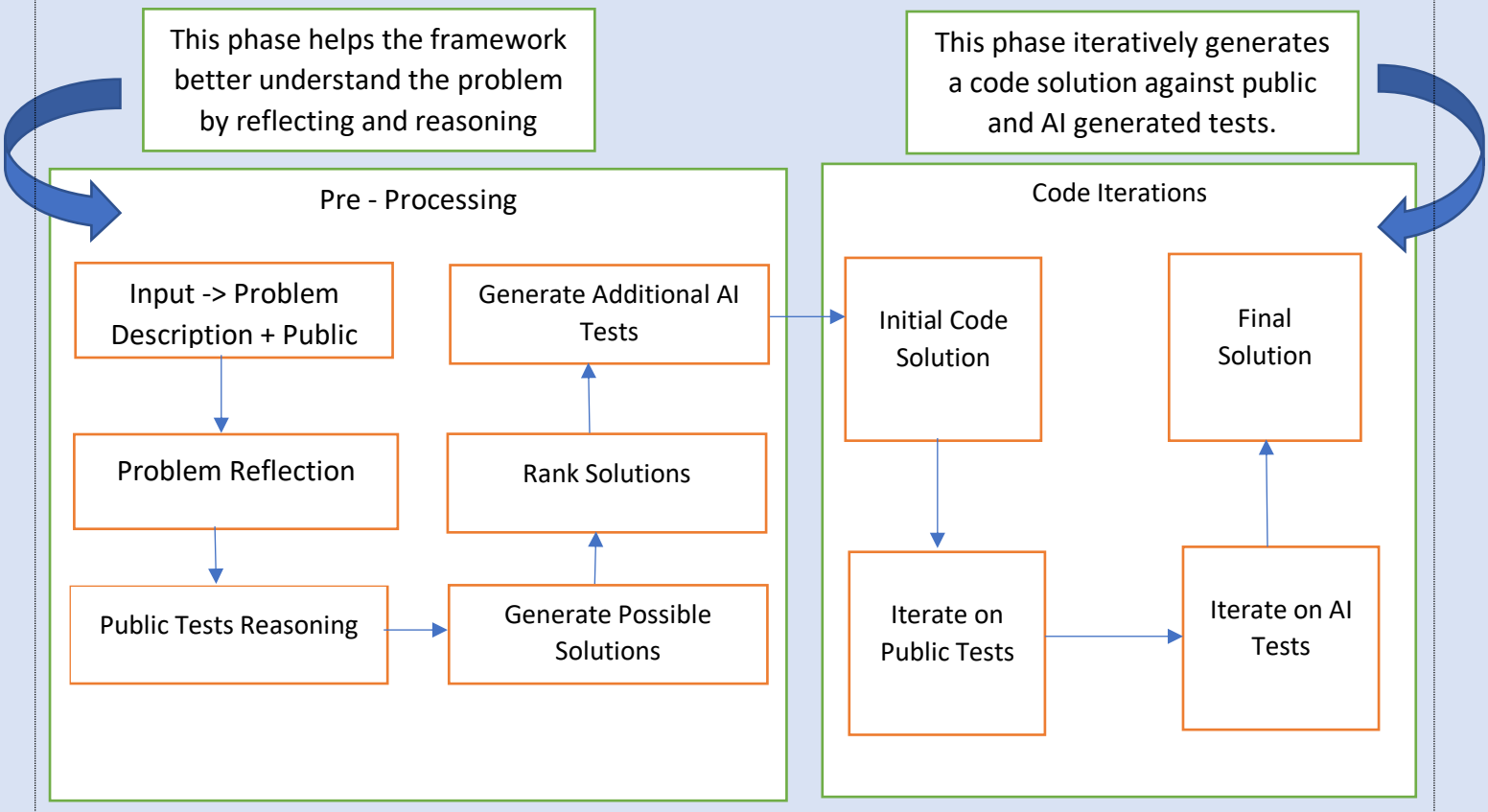
Usecase:

1. Question answer model
2. Suggestion model
3. Answer from document model
4. Understanding users prompt and response back to user.
5. AI generated response using GPT 3.5 Turbo

- 9 Important Module use in Chatbot to response back to user

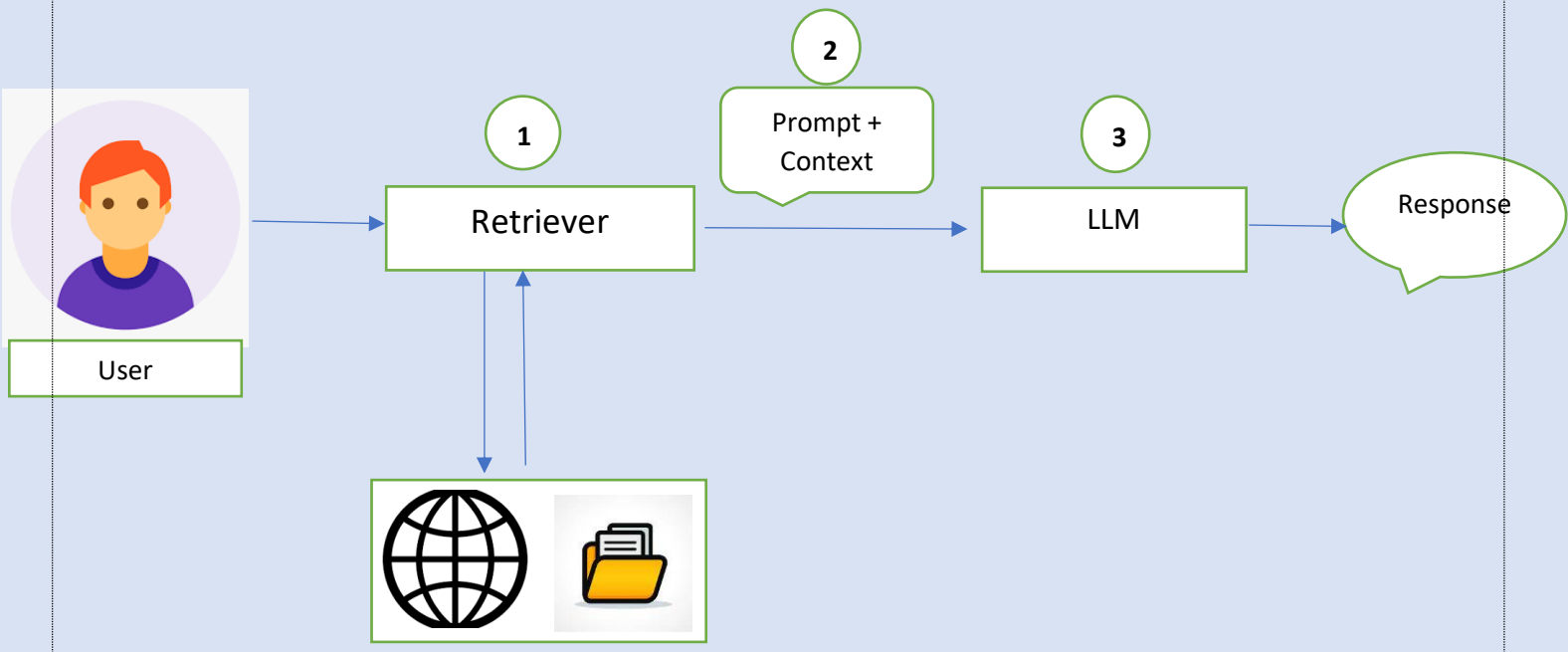
1. For Taking and understanding – Speech Recognition and generation
2. Understanding human language – NLP
3. Work on Human Language – NLU
4. Processing users query – LLM and Open AI
5. Handling positive and negative scenarios/ Categorizing users response – Generative AI
6. Answering users questions – Transformation model
7. Giving suggestions and handling unexpected scenarios – experts system
8. Store the answers and data – knowledge extraction and representation.
9. Summarization to conversational agents to content generation - Retrieval Augmented Generation

1. Understanding Users Prompt and Intent detection (Understand user query to work on it and follow step by step approach)



2. Question Answer Model

Usecase – Understanding users query and response back to them.



Phase 1: Lookup the external source to retrieve the relevant information

Phase 2: Add the retrieved information to the user prompt

Phase 3: Use LLM to generate response to user prompt with the context

Step 1: User Enters a prompt/query

Step 2: Retriever searches and fetches information relevant to the prompt (e.g. from the internet or internet data warehouse)

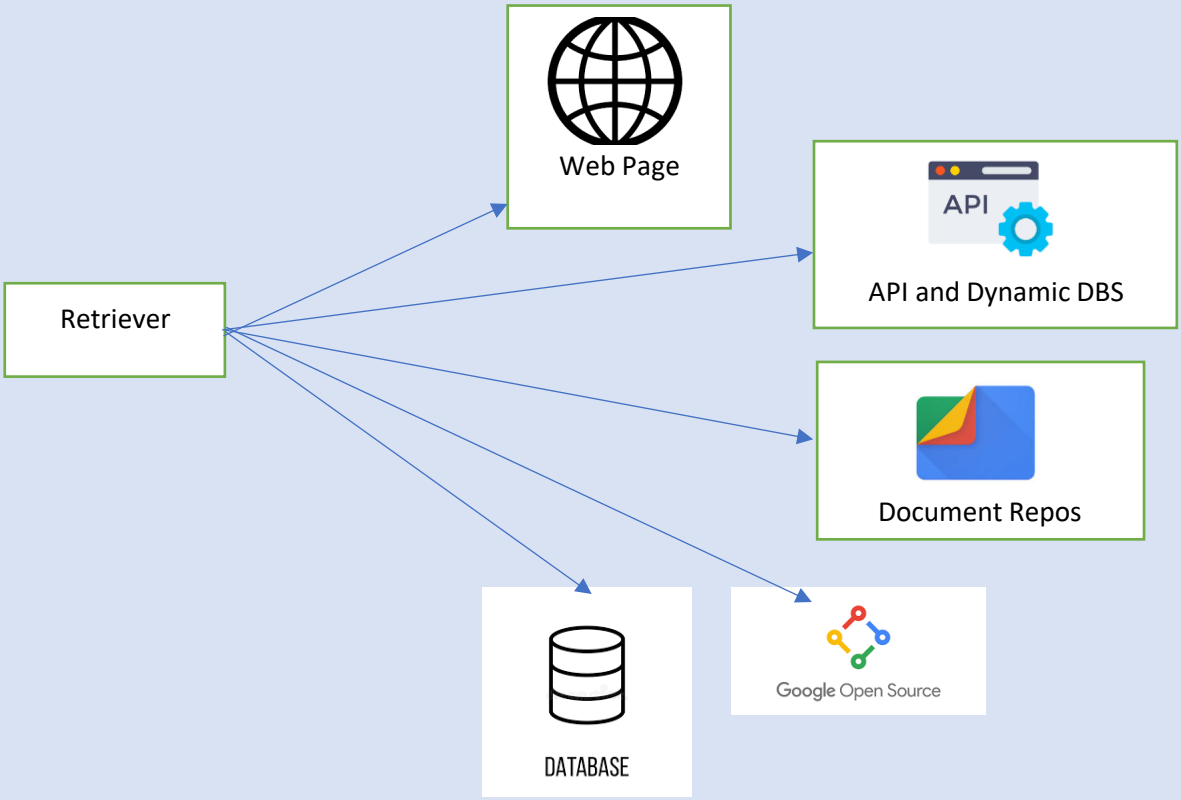
Step 3: Retrieved relevant information is augmented to the prompt as context

Step 4: LLM is asked to generate response to the prompt in the context(augmented information)

Step 5: Users Response generation.

- Information and Retriever Search from various source

A retriever is an interface that returns documents based on an unstructured query



Source:

1. Document Question Answering Systems

By Providing access to proprietary enterprise document to an LLM, the response are limited to what is provided within them. A retriever can search for the most relevant document and provide the information to the LLM.

2. Conversational Agent

LLMs can be customised to product/service manuals, domain knowledge, guidelines, etc. The agent can also route users to more specialised agent depending on their query.

3. Content Generation

The widest use of LLMs has probably been in content generation. Using RAG, the generation can be personalised to readers, incorporate real-time trends and be contextually appropriate.

4. Real Time Event Commentary

A retriever can connect to real-time updates/data via APIs and pass this information to the LLM to create a virtual commentator. These can further be augmented with Text to speech model.

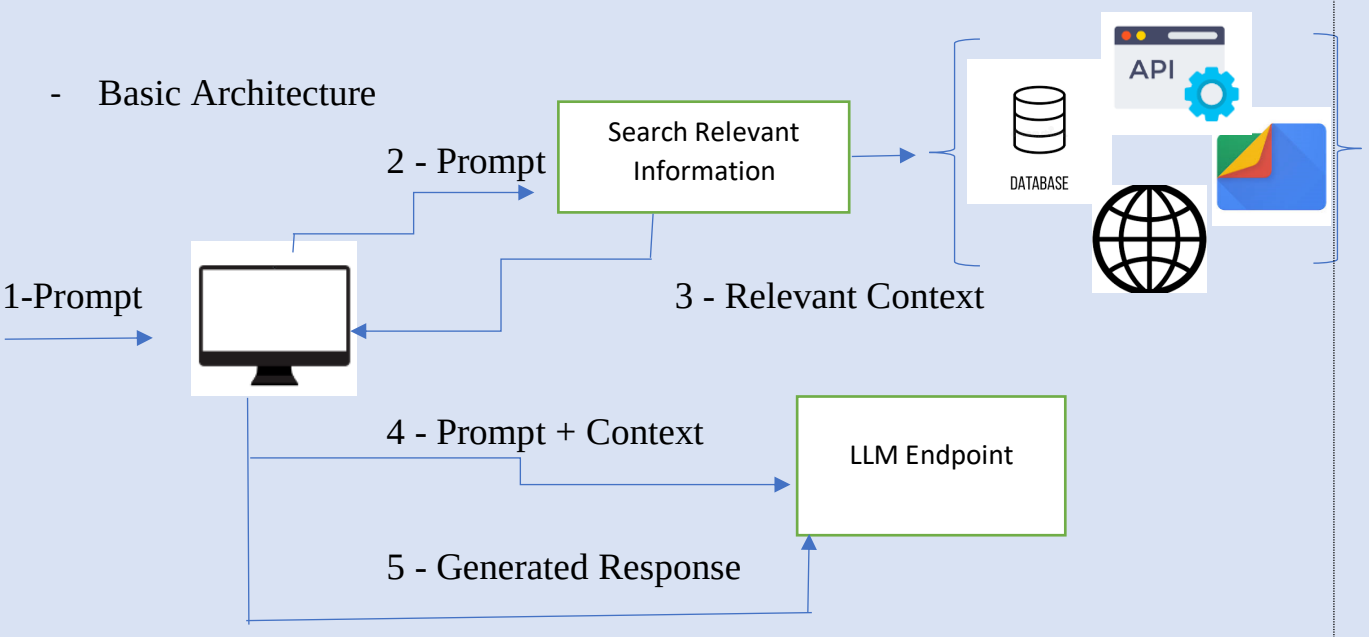
5. Personalised Recommendation

Recommendation engines have been a game changes in the digital economy. LLMs are capable of powering the next evolution in content recommendation.

6. Virtual Assistants

Virtual Personal assistant like Siri, Alexa and others are in plans to use LLMs to enhance the experience. Coupled with more context on user behaviour, these assistants can become highly personalised.

- Basic Architecture



1. User writes a prompt or a query that is passed to the Chatbot.
2. Chatbot sends a search query to the retriever
3. Retriever fetches the relevant information from the knowledge sources and sends back
4. Chatbot augments the prompt with the context and sends to the LLM
5. LLM responds with the generated text which is displayed to the user via the Chatbot.

3. Storing User Prompt and transcripts to learning. [Reinforcement Learning Mechanism – Learn from the users feedback]

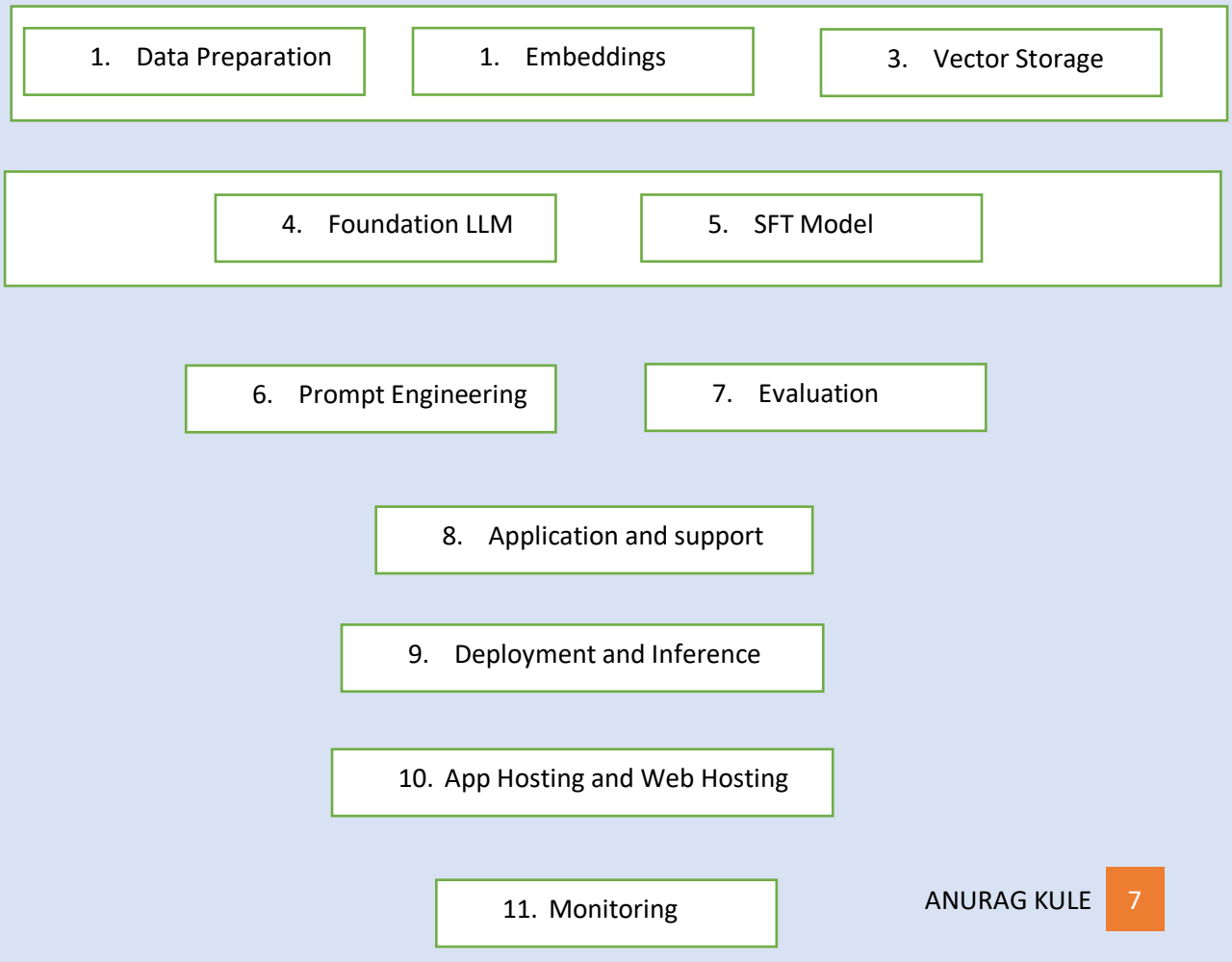
Mechanism to Store -> Vector Database.

Persistent Vector DBs: When a large volume of data is stored in vector databases, the retrieval and search needs to be quick. The relevance and accuracy of the search can be tested.

Steps:

1. Loading our text file using TextLoader
2. Splitting the text into chunks using RecursiveCharacterTextSplitter
3. Creating embeddings using OpenAI Embeddings
4. Storing the embeddings into Qdrant
5. Set the LLM as ChatOpenAI (gpt 3.5)
6. Set up logging to see the query variations generated by the LLM
7. use MultiQueryRetriever & get_relevant_documents functions

4. Deployment of Chatbot one customers environment(LLMOps)



1. Data Layer

1. Data Preparation – Sourcing, Cleaning, Loading and Chunking

2. Creating of Embeddings

3. Storing the embeddings in vector store

Data Preparation	Embeddings	Vector Storage
1. LangChain 2. LlamaIndex	1. OpenAI 2. Hugging Face	1. Pinecone 2. Chroma 3. Milvus 4. Drant

LLM Model

1. Proprietary Foundation Model - Developed and maintained by providers (like OpenAI, Anthropic, Google) and is generally available via an API [OpenAI – GPT3.5/GPT4]
2. Open Source Foundation Model - Available in public domain (like Falcon, Llama, Mistral) . [Llama 2 by Meta, Mistral & Mixtral and Falcom phi2 by MicroSoft]
3. A Supervised Fine-Tuned Proprietary Model - Providers enable fine-tuning of their proprietary models with your data. The fine-tuned models are still hosted and maintained by the providers and are available via an API
4. A Supervised Fine-Tuned Open Source Model - All Open Source models can be fine-tuned by you on your data using full fine-tuning or PEFT methods.

2. Prompt Layer

Prompt Engineering is more than writing questions in natural language. There are several prompting techniques and developers need to create prompts tailored to the use cases. This process often involves experimentation: the developer creates a prompt, observes the results and then iterates on the prompts to improve the effectiveness of the app.

Model – PromptLayer, PromptLeo, Promptly and Promptient

3. App Orchestration

Model – Langchain and LlamaIndex.

4. Deployment Layer

Factors should be consider while deployment

- Security and Governance
- Logging
- Inference costs and latency

5. Application Layer

The Application need to be hosted for the intended users or system to interact with it.

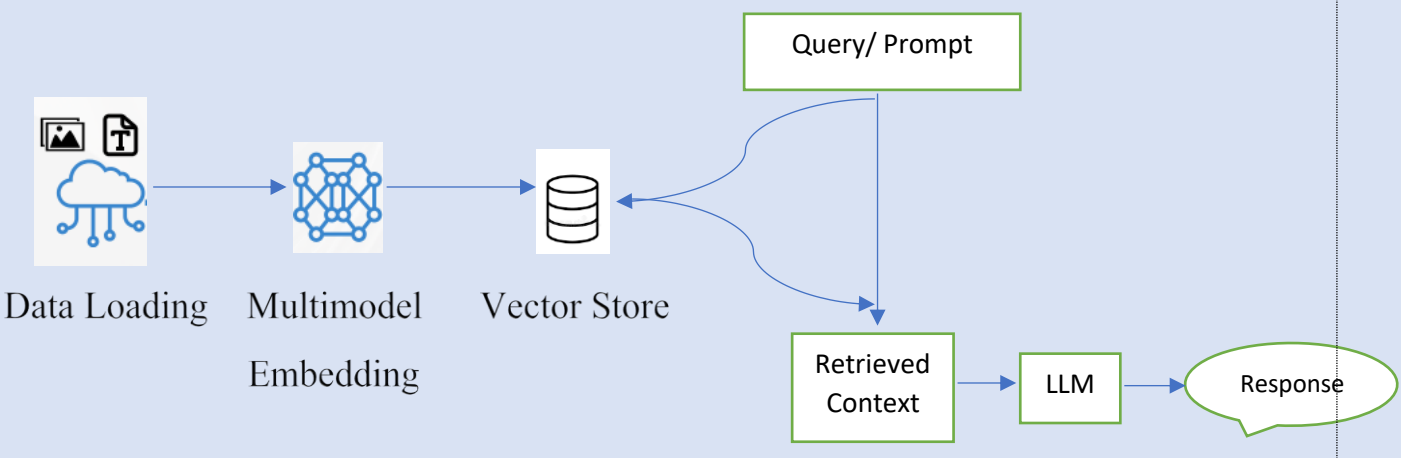
Model – Streamlit, Streamship and Heroku

6. Monitoring

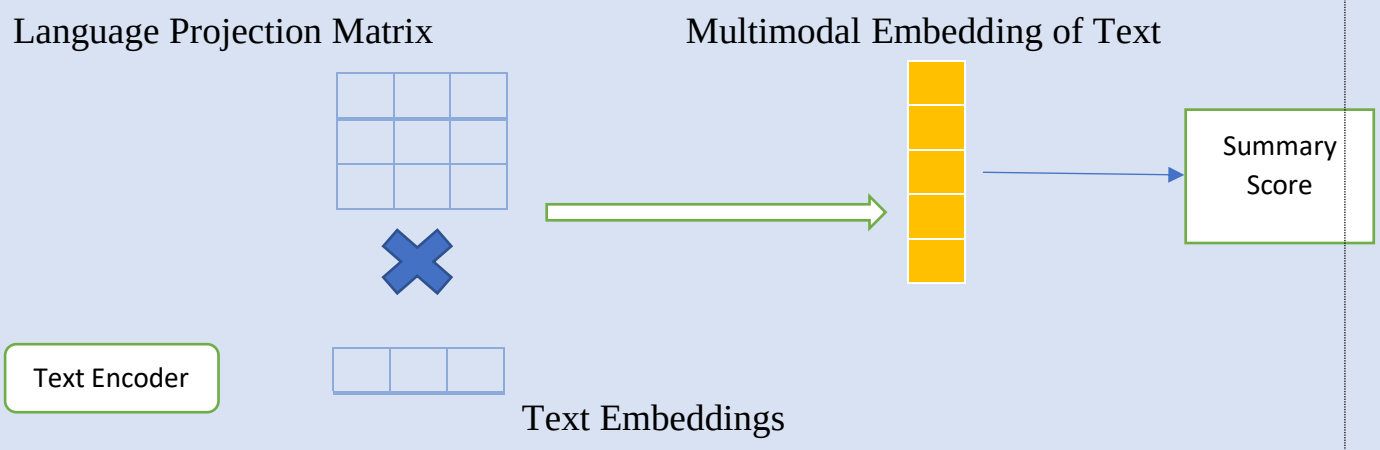
Deployed application needs to be continuously monitored for both accuracy and relevance as well as cost and latency.

Model – HoneyHive and new relic

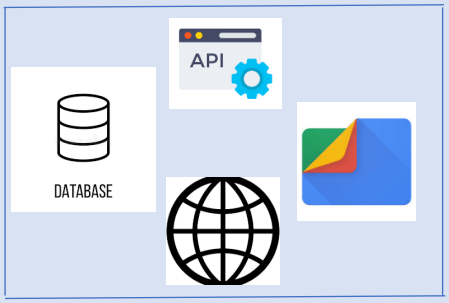
Multimodel Embedding

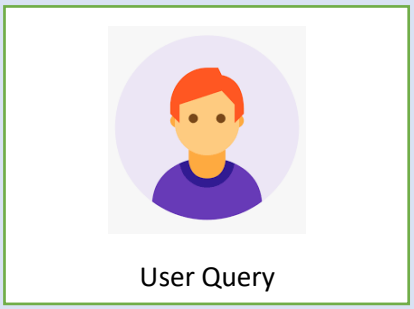


CLIP: Contrastive Language-Image Pre-training

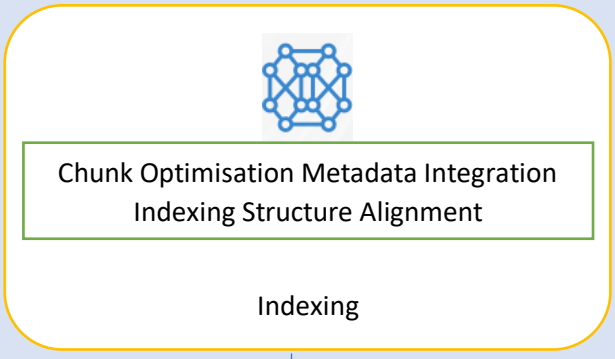

Language Projection Matrix
Multimodal Embedding of Text
Summary Score
Text Encoder
Text Embeddings

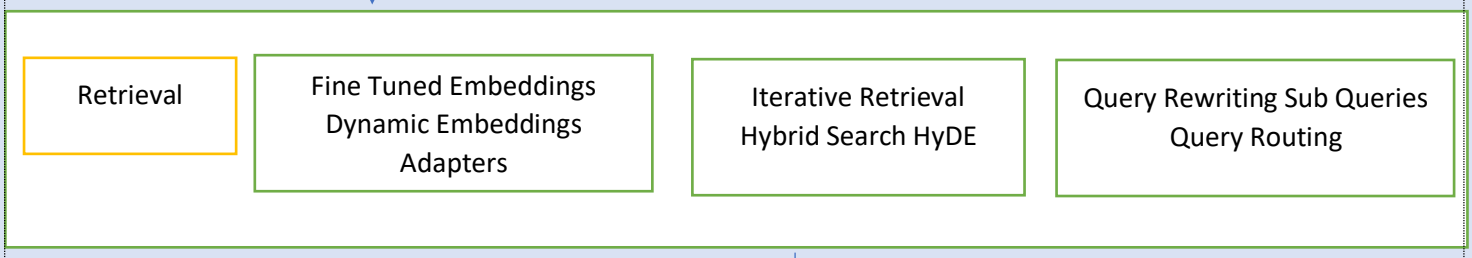
5. Handling Complex Scenario and Users Query. Understanding the Intent and working on it.

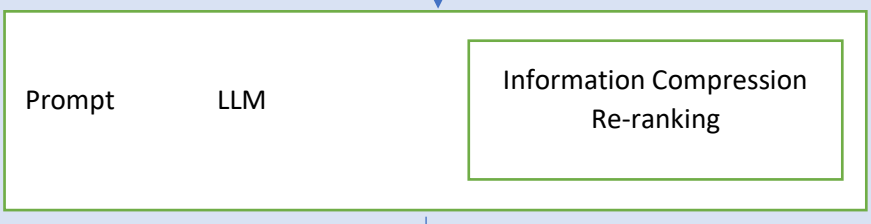




Document(Source)

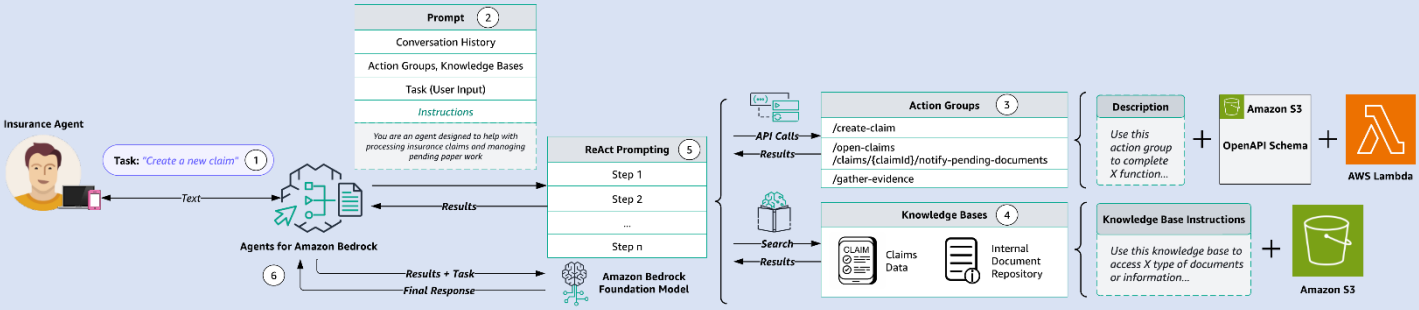






Usecase 2 : Automate the insurance claim lifecycle using Agents and Knowledge Bases for Amazon Bedrock

Solution Architecture



Step 1. Users provide natural language inputs to the agent. Following are some sample prompts by users

a. *What is the repair estimate total for that same claim?*
b. *How can I lower my car insurance rates?*
c. *Which claims have open status?*

Step 2. During preprocessing, the agent validates, contextualizes, and categorizes user input. The user input (or task) is interpreted by the agent using chat history and the instructions and underlying FM that were specified during agent creation. The agent's instructions are descriptive guidelines outlining the agent's intended actions. Also, you can optionally configure advanced prompts, which allow you to boost your agent's precision by employing more detailed configurations and offering manually selected examples for few-shot prompting. This method allows you to enhance the model's performance by providing labeled examples associated with a particular task.

Step 3. Action groups are a set of APIs and corresponding business logic, whose OpenAPI schema is defined as JSON files stored in Amazon Simple Storage Service (Amazon S3). The schema allows the agent to reason around the function of each API. Each action group can specify one or more API paths, whose business logic is run through the AWS Lambda function associated with the action group.

Step 4. Knowledge Bases for Amazon Bedrock provides fully managed RAG to supply the agent with access to your data. You first configure the knowledge base by specifying a description that instructs the agent when to use your knowledge base. Then you point the knowledge base to your Amazon S3 data source. Finally,

you specify an embedding model and choose to use your existing vector store or allow Amazon Bedrock to create the vector store on your behalf. After it's configured, each data source sync creates vector embeddings of your data that the agent can use to return information to the user or augment subsequent FM prompts.

Step 7. During orchestration, the agent develops a rationale with the logical steps of which action group API invocations and knowledge base queries are needed to generate an observation that can be used to augment the base prompt for the underlying FM. This ReAct style prompting serves as the input for activating the FM, which then anticipates the most optimal sequence of actions to complete the user's task.

Step 6. During postprocessing, after all orchestration iterations are complete, the agent curates a final response. Postprocessing is disabled by default.

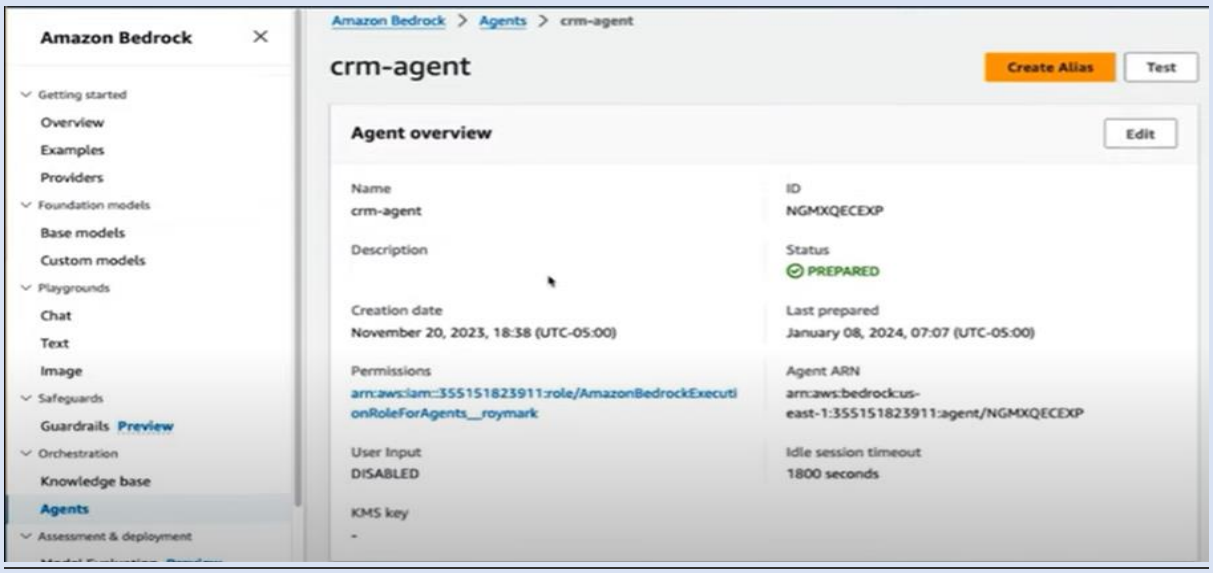
- How to configure the Agent in AWS Bedrock

Reference Link - https://youtu.be/UcehCSSOMQA?si=fTWpjEnDO2hVkT8P

AWS Documentation:
https://docs.aws.amazon.com/bedrock/latest/userguide/agents-create.html

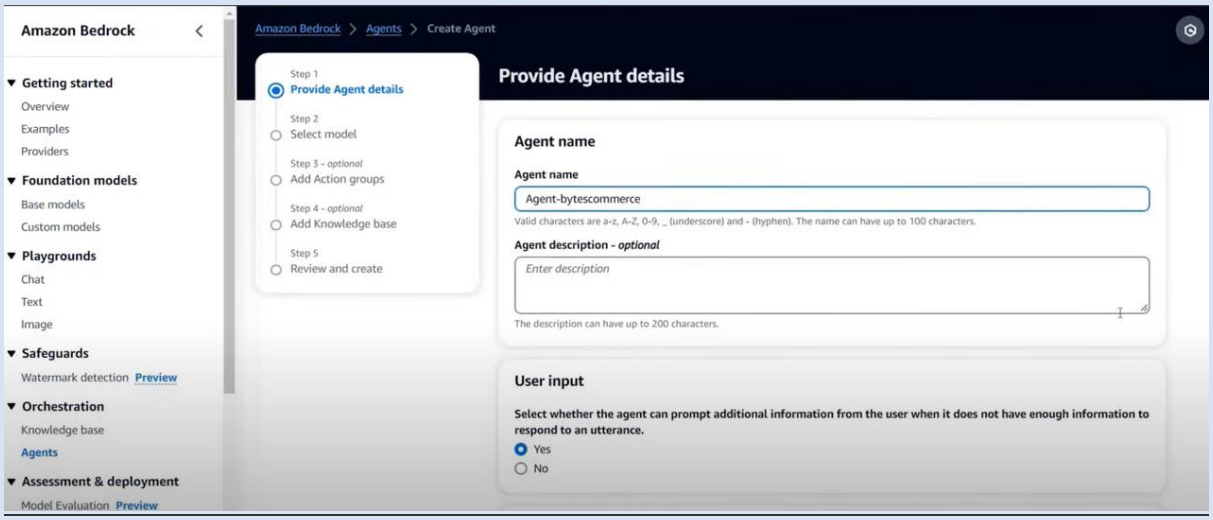
Step 1. Sign in to the AWS Management Console, and open the Amazon Bedrock console

Step 2. Select Agents from the left navigation pane.

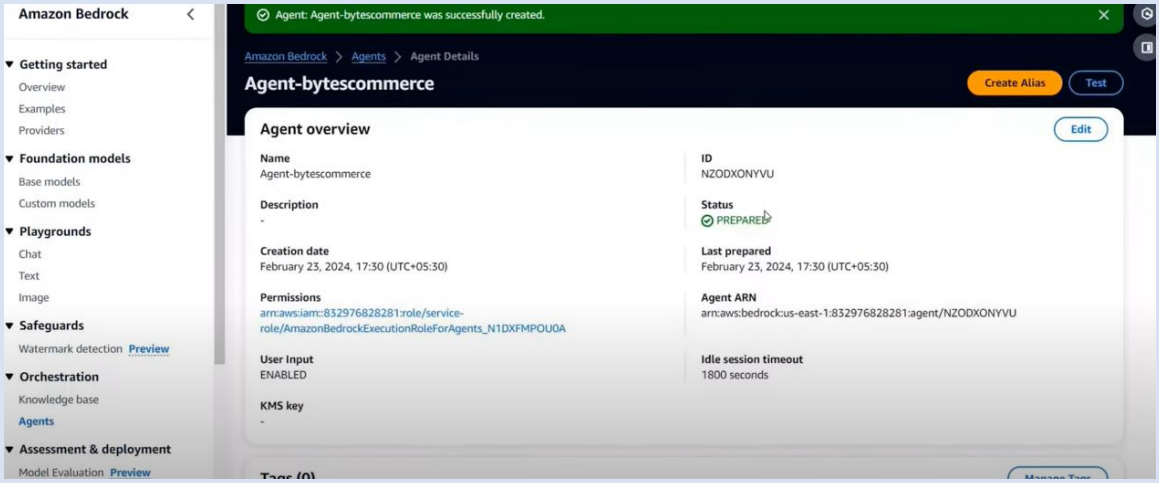


Step 3. In the Agents section, choose Create Agent.

Step 4. Change the automatically generated Name for the agent and provide an optional Description for it.

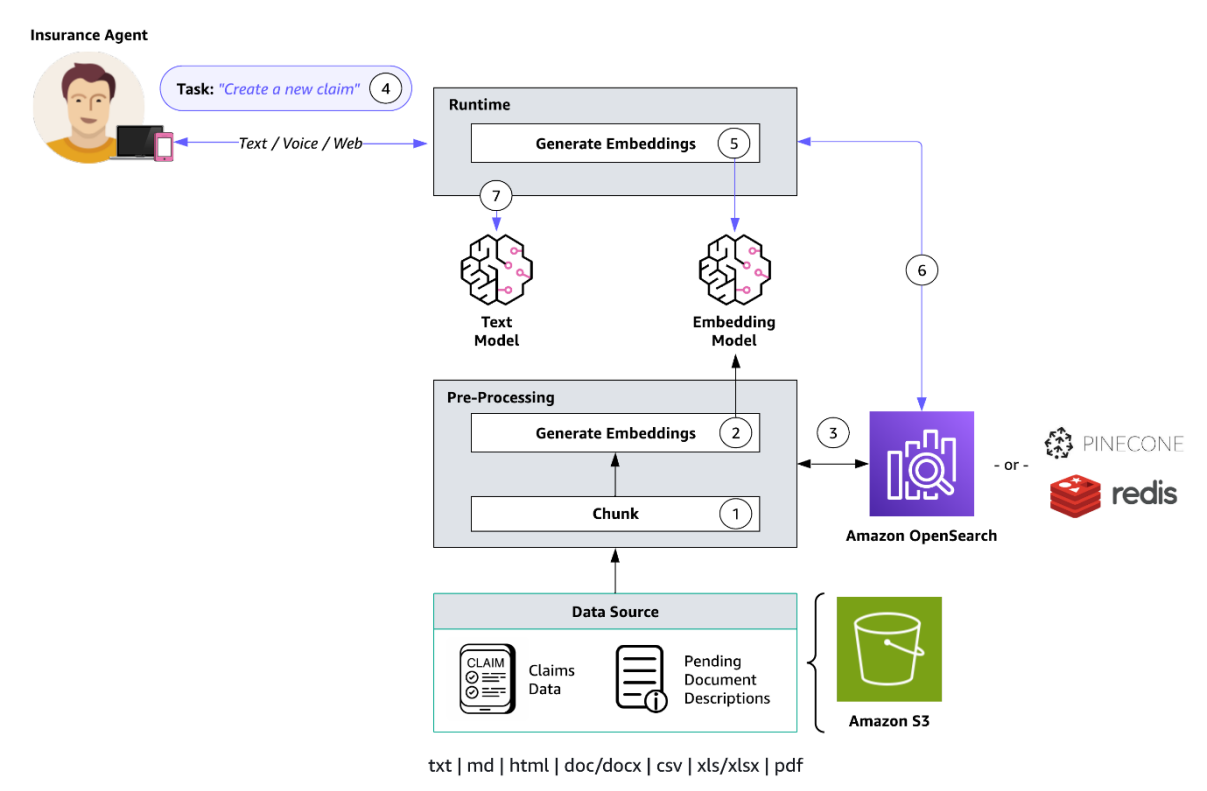


Step 5. Choose Create. Your agent is created and you will be taken to the Agent builder for your newly created agent, where you can configure your agent.



- How the configure the Agent with the knowledge base and Training of Agent.

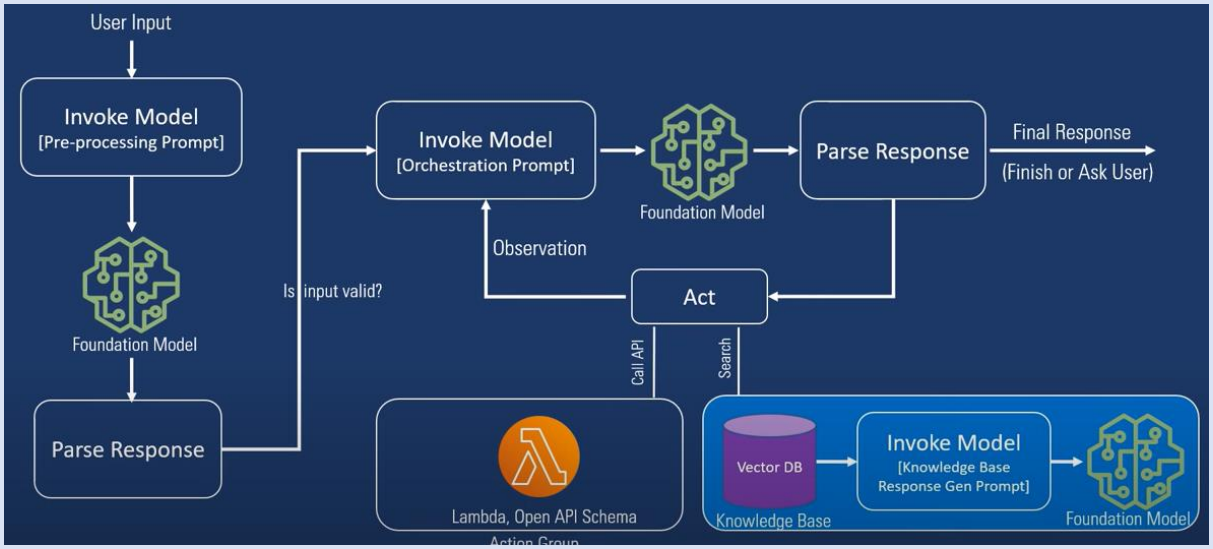
1. Create a knowledge base of the Agent



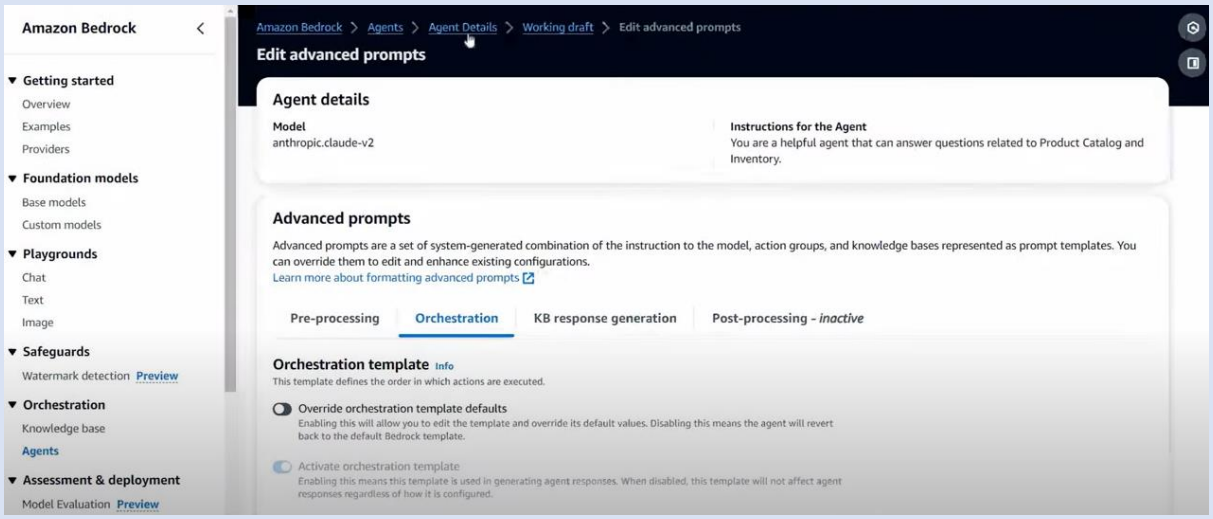
Knowledge base functionality is delineated through two key processes: preprocessing (Steps 1-3) and runtime (Steps 4-7):

1. Documents undergo segmentation (chunking) into manageable sections.
2. Those chunks are converted into embeddings using an Amazon Bedrock embedding model.
3. The embeddings are used to create a vector index, enabling semantic similarity comparisons between user queries and data source text.
4. During runtime, users provide their text input as a prompt.
5. The input text is transformed into vectors using an Amazon Bedrock embedding model.
6. The vector index is queried for chunks related to the user's query, augmenting the user prompt with additional context retrieved from the vector index.
7. The augmented prompt, coupled with the additional context, is used to generate a response for the user.

2. Bedrock Agent Flow Architecture



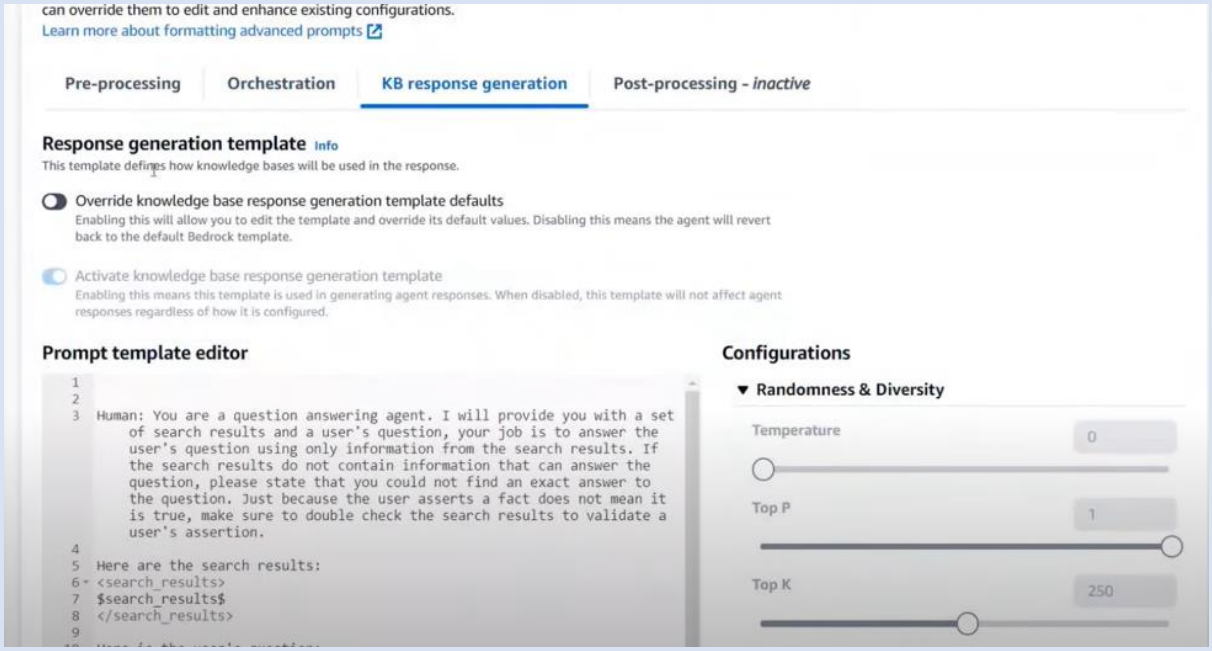
3. Adding Advance Prompts for Agent calling



User input and agent instruction validation includes the following:

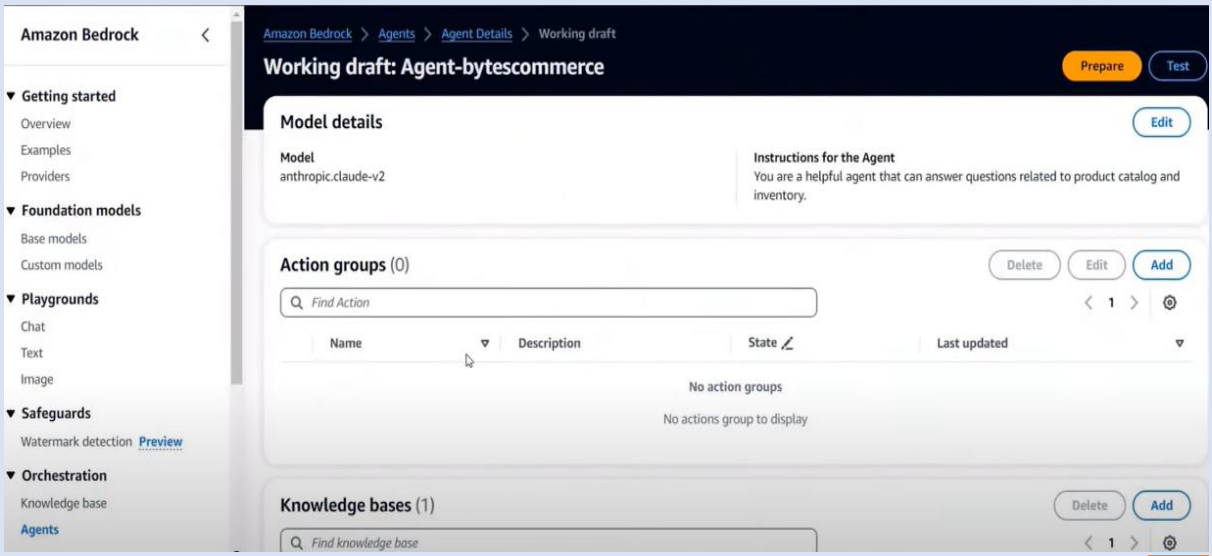
- Preprocessing – Use sample prompts to assess the agent's interpretation, understanding, and responsiveness to diverse user inputs. Validate the agent's adherence to configured instructions for validating, contextualizing, and categorizing user input accurately.
- Orchestration – Evaluate the logical steps the agent follows (for example, "Trace") for action group API invocations and knowledge base queries to enhance the base prompt for the FM.

- Postprocessing – Review the final responses generated by the agent after orchestration iterations to ensure accuracy and relevance. Postprocessing is inactive by default and therefore not included in our agent's tracing.



4. Action group

- API schema validation – Validate that the OpenAPI schema (defined as JSON files stored in Amazon S3) effectively guides the agent's reasoning around each API's purpose.
- Business logic Implementation – Test the implementation of business logic associated with API paths through Lambda functions linked with the action group.



Usecase 3: Deploying the model using Streamlit

Reference Link: https://youtu.be/WLwjvWq0GWA?si=9IsXDadB9WZjPXBk

Step 1. Set up AWS Bedrock and obtain credentials

a. Sign up for AWS if you haven't already.
b. Create an IAM user with the necessary permissions to access AWS Bedrock.
c. Obtain the access key ID and secret access key for the IAM user.

Step 2. Install required packages

Install Streamlit and the AWS SDK for Python (Boto3) on your local machine.

Step 3. Create a Python script for the Streamlit app

Step 4. Configure AWS credentials

Step 5. Deploy the Streamlit app

Run using: streamlit run app.py