Brance Founding ML EngineerTask

Name:Harshvardhan Chaturvedi
Linkedin Profile:**https://www.linkedin.com/in/harsh-vardhan-a3a364140/**
Date Challenge Received:06-07-23(08-07-23 work started)
Date Solution Delivered:11-07-23

1. Problem Statement

What was the task and how you understood it.

The task was to build a RAG(Retrieval Augmented Generation) chatbot, i.e, fetch relevant information about user queries based on the knowledge document present.

2. Approach

Your approach to the problem. Mention any assumptions made.

My Approach was as follows:

1. Created a Redis Client, and stored some information about Pan Card, which was provided in the knowledge statement.
2. Take the query as input from the user.
3. Search the Redis database for the answer. If it's present then return the response, or else I was returning "Answer not found."
4. Created a Flask Application for the above.

3. Solution

Details about your solution. Illustrate performance and design with diagrams.

Design for a scalable and efficient system will have the following components:

1. Cloud Compute Platform: We will use AWS as the cloud computing platform for hosting the question-answering AI model.

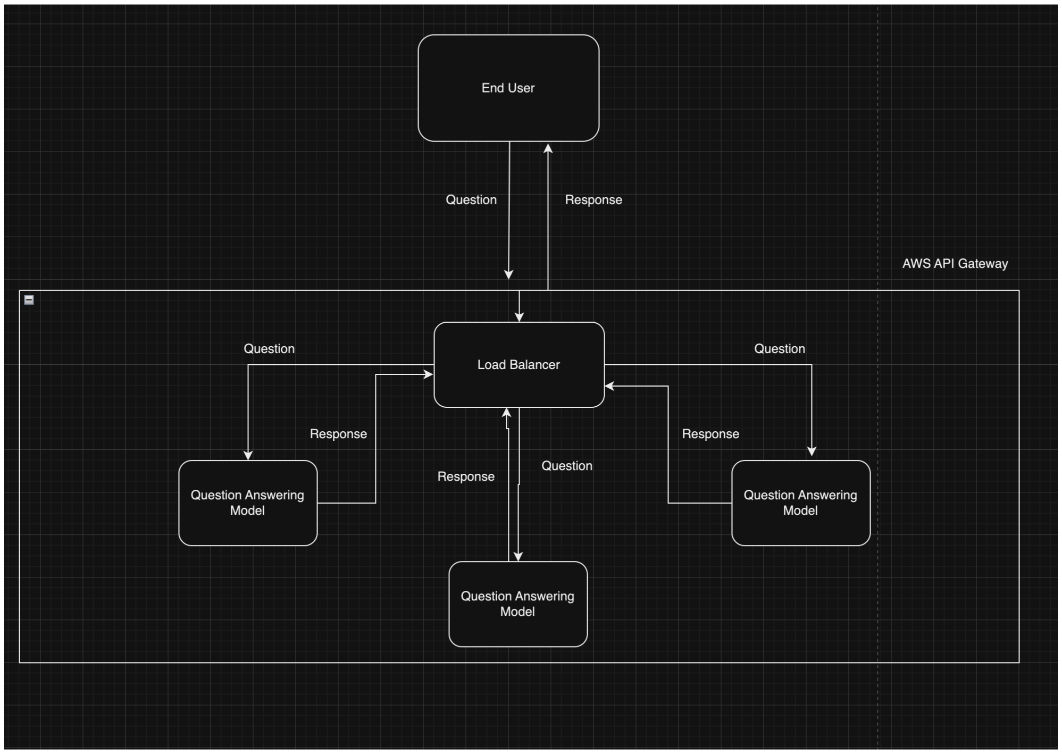
2. Load Balancer: To distribute the incoming requests evenly across multiple instances of the question-answering AI model, we will employ a load balancer. AWS Elastic Load Balancer (ELB) or Application Load Balancer (ALB) can be used for this purpose.

3. Auto Scaling: By utilizing AWS Auto Scaling, we can automatically adjust the number of instances running the question-answering AI model based on the incoming traffic.

4. Model Hosting: The question-answering AI model can be hosted using AWS Elastic Compute Cloud (EC2) instance.

5. REST API: The question-answering AI model will be exposed through a REST API endpoint.
6. Caching and Result Storage: To improve response times and reduce the load on the question-answering AI model, we can incorporate a caching layer. For this purpose, a vector database like Redis can be used.

Design to Handle 1M requests:



Flow:

1. The client sends a question to the REST API endpoint exposed by AWS API Gateway.
2. The load balancer distributes the incoming requests across multiple instances of the question-answering AI model.
3. Each instance processes the question and generates a response.
4. Before querying the question-answering AI model, the instances check the Redis cache for a cached answer corresponding to the received question.
5. If the answer is found in the cache, it is returned directly to the client.
6. If the answer is not in the cache, the question-answering AI model processes the question and generates a response.
7. The response is then stored in the Redis cache for future requests.
8. The generated answer is returned to the client through the REST API.
9. The system automatically scales the number of instances based on the incoming traffic, ensuring efficient handling of over 1 million daily requests.

Justification for Redis as a Vector Database:

Redis is a popular choice for handling high volumes of requests because of its in-memory data storage and support for data structures such as lists, sets, and hashes. Additionally, it has built-in support for caching and pub/sub messaging, which can be useful for high-performance applications.

4. Future Scope

Thoughts on how you could have improved the solution.

Due to time-constraint, I couldn’t build a very efficient solution and also deploy it on AWS. But it can surely be done be in the future. Moreover, to improve the solution, we can:

- Add an LLM model in the middle to which we can send the user query as a prompt, to get a better understanding of the user query. Also, before returning the response, we can properly frame the response with the help of this LLM model.

- Supporting multi-lingual model: Using the LLM model, we can create a solution which can support multi-lingual requests.