Customer Support Chatbot With RAG Model

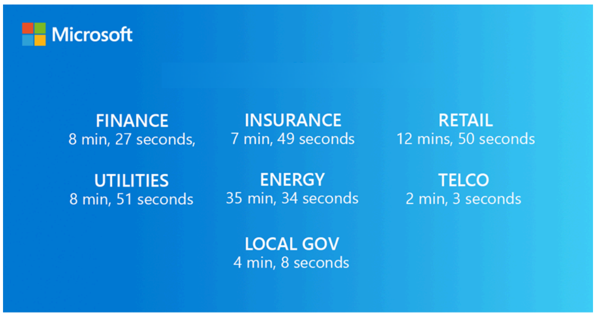
Instructor : Suat Özdemir
Burak Kurt | Kemal Şahin | İlbey Gülmez | İzzet Ahmet

1. Problem Definition

The goal is to enhance the efficiency and accuracy of customer support services by implementing a Retrieval-Augmented Generation (RAG) model. Traditional customer support relies heavily on human agents, leading to high operational costs and inefficiencies. With increasing customer inquiries, maintaining high-quality support becomes challenging, especially in time-sensitive scenarios. This system aims to automate and streamline responses, significantly reducing response times, improving accuracy, and lowering operational costs by eliminating the need for human intervention in routine queries. The RAG model will leverage a vast knowledge base to generate accurate and contextually relevant responses to customer inquiries, addressing both frequent and unique issues effectively.

1.1 What We Need To Improve

The average response time in traditional customer support systems is often too high, leading to customer dissatisfaction and increased frustration. Customers today expect quick and accurate responses, yet many support services still rely on manual processes or scripted responses that lack contextual understanding. This delay in providing accurate information not only hampers customer experience but also increases operational costs. To meet the growing demand for instant and precise support, it is crucial to implement systems that can reduce response times significantly while maintaining high levels of accuracy. By utilizing advanced technologies like the Retrieval-Augmented Generation (RAG) model, we can automate the support process, ensuring that customers receive relevant, accurate, and timely responses without the need for human intervention in every interaction. This approach will enhance customer satisfaction, improve operational efficiency, and reduce costs.



Microsoft

FINANCE
8 min, 27 seconds,

INSURANCE
7 min, 49 seconds

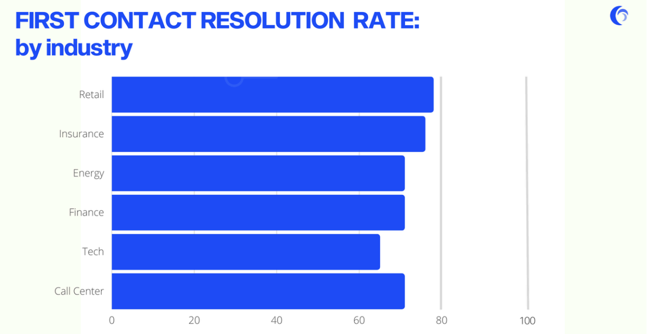
RETAIL
12 mins, 50 seconds

UTILITIES
8 min, 51 seconds

ENERGY
35 min, 34 seconds

TELCO
2 min, 3 seconds

LOCAL GOV
4 min, 8 seconds



2. Solution Methodology

We initially tested Llama 3B and 8B models, but they did not produce satisfactory results, often leading to hallucinations. Afterward, we switched to the 4o mini model and adopted the Advanced RAG approach. In the beginning, we combined Llama with the Naive RAG setup, starting with Llama 3B and then testing Llama 8B. However, the results remained suboptimal. We then transitioned to 4o mini with Naive RAG, and finally moved to the more advanced combination of 4o mini and Advanced RAG. This shift allowed us to leverage the strengths of retrieval-based methods and generative capabilities, ensuring more accurate, context-aware responses. Advanced RAG helps pull the most relevant information from the knowledge base and generate responses grounded in real, relevant data, improving both speed and accuracy in customer support scenarios.

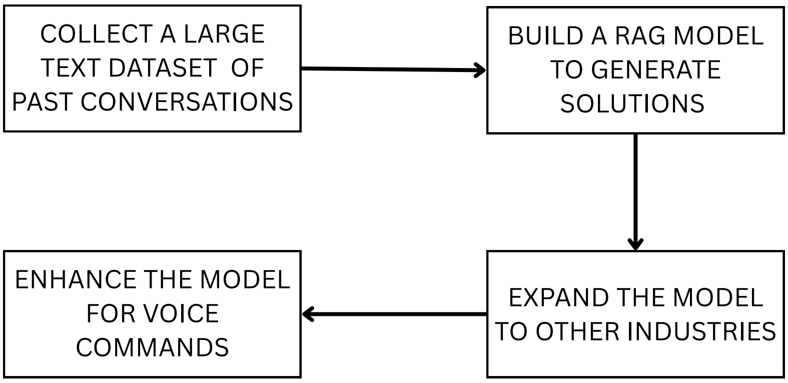
2.1. Why Advanced RAG?

Advanced RAG enhances Naive RAG by integrating advanced techniques like re-ranking and contextual retrieval, leading to improved accuracy, especially for fact-based tasks. By retrieving and processing multiple relevant documents, Advanced RAG provides more precise answers. Although it may be slower due to these additional steps, it excels in complex, knowledge-driven tasks and better handles unknown or out-of-scope information. In contrast, Naive RAG offers faster performance but lacks the sophisticated contextual processing of Advanced RAG, making it less effective for nuanced or fact-intensive queries.

Feature	Advanced RAG	Naive RAG
Accuracy	More accurate for fact-based tasks with real data, using sophisticated retrieval and re-ranking methods.	May be less accurate as it only retrieves a simple top result, without advanced re-ranking or contextual refinement.
Speed	Slower due to additional processing steps such as retrieval, re-ranking, and generation.	Faster as it retrieves and generates in a more straightforward manner, without additional ranking steps.
Use Case	Best for knowledge-intensive tasks like question answering, where context and relevance are key.	Suitable for simpler tasks where basic retrieval and generation suffice, but lacks the nuance for more complex queries.
Handling Unknown Information	Handles unknown information effectively by retrieving and re-ranking multiple relevant documents for better context.	Struggles with unknown or out-of-scope information, as it relies on simple retrieval without advanced contextual handling.

2.2. Solution Steps

The solution for developing the RAG model for customer support will be carried out in several stages. First, we will collect a large text dataset consisting of past conversations to build a foundation for the model. In the next phase, we will focus on the aviation industry, specifically creating and fine-tuning a RAG model tailored to handle customer support queries in this sector. Once the aviation model is optimized, we will expand it to other industries, ensuring the model's adaptability to different contexts and customer needs. Finally, the model will be enhanced to handle voice commands, allowing it to process and respond to audio inputs, thereby providing a comprehensive, multi-modal customer support solution. This phased approach ensures a scalable and efficient deployment of the RAG model across various domains.


COLLECT A LARGE TEXT DATASET OF PAST CONVERSATIONS
BUILD A RAG MODEL TO GENERATE SOLUTIONS
EXPAND THE MODEL TO OTHER INDUSTRIES
ENHANCE THE MODEL FOR VOICE COMMANDS

2.3. Building the RAG Model: A Detailed Overview

The development of our Retrieval-Augmented Generation (RAG) model follows a structured pipeline consisting of several stages to ensure efficiency and accuracy. Here's a detailed breakdown of the process:

1. Data Collection and Preprocessing

- Fetch Data:
    - The primary dataset for this project is the Customer Support on Twitter dataset from Kaggle, which includes consumer conversations with customer support agents on Twitter. Key fields include:
        - tweet_id: A unique, anonymized ID for the Tweet. Referenced by response_tweet_id and in_response_to_tweet_id.
        - author_id: A unique, anonymized user ID. "@s" in the dataset has been replaced with their associated anonymized user ID.
        - inbound: Whether the tweet is "inbound" to a company doing customer support on Twitter. This feature is useful when reorganizing data for training conversational models.
        - created_at: Date and time when the tweet was sent.
        - text: Tweet content. Sensitive information like phone numbers and email addresses is replaced with masked values like __email__.
        - response_tweet_id: IDs of tweets that are responses to this tweet, comma-separated.
        - in_response_to_tweet_id: ID of the tweet this tweet is in response to, if any.

- Apply Preprocessing:
  - Once the data is collected, preprocessing is applied to clean and structure the raw text. This step includes removing irrelevant elements like usernames (e.g., "@user"), URLs, and special characters. The text is also normalized by eliminating extra spaces and trimming any leading or trailing whitespace. Additionally, tokenization and lemmatization are applied to standardize words and ensure efficient model processing. Depending on the dataset's requirements, other preprocessing steps, such as stopword removal or stemming, may also be implemented to further optimize the data for analysis.

- JSON Structure:
  - After preprocessing, the text data is transformed into a structured JSON format. This format ensures that the model can easily access and query the data during both training and inference. The JSON structure typically includes various fields such as ChatID, Company_name, Conversation_History, Entities, Relationships, and Embedding. Each conversation is stored with its associated company name, the entities involved, the relationships extracted, and a corresponding embedding generated from the structured conversation data. This structured format is then saved to an Excel file, making it accessible for further processing and analysis.

2. Intent Extraction

Intent extraction is performed sequentially to ensure that each component of the customer query is clearly identified and understood in the right order. By addressing specific aspects like product, service, issue type, and relationships step by step, the model can build a precise understanding of the user's needs. This sequential approach helps avoid confusion, ensures context is captured accurately, and allows for the generation of more relevant and coherent responses.

- Product Extraction:
  - The model identifies the product or service the customer is inquiring about. This can include specific product names, models, or categories. For example, if the customer mentions a specific aircraft, the model recognizes the product reference.
- Service Extraction:
  - This step focuses on identifying the type of service the customer is requesting. It could involve determining if the customer needs assistance with booking, checking flight status, managing baggage, etc.
- Issue Type Extraction:
  - The model classifies the nature of the issue or query. For instance, if a customer is experiencing a delay, the model might categorize the issue as "flight delay." This helps the model understand the urgency and context of the query.
- Relationship Extraction:
  - The model extracts the relationships between different entities in the query, such as the connection between a customer and a flight booking or a product and a service request.

3. Storage and Vectorization

- Vectorization:
  - To efficiently store and search conversational data, we implemented a vector database by leveraging embeddings generated from SentenceTransformer. This approach allows us to represent complex textual data, such as customer conversations, in a numerical form that preserves semantic meaning, making similarity searches faster and more accurate. Additionally, due to the embedding model's lower token limit, this new method enables us to capture more context, further enhancing the accuracy of the searches.

Process Overview:

3.1. Embedding Generation:

- We used the SentenceTransformer model, specifically the all-MiniLM-L6-v2 model, to convert textual conversations into embeddings. These embeddings are high-dimensional vectors that capture the meaning and context of each conversation. Each conversation, along with its associated entities and relationships, is transformed into two embeddings: one for the intent (summary of the conversation) and one for the conversation text itself. At the final stage, these embeddings are combined, ensuring that both the intent and the detailed conversation context are captured together for more accurate similarity measurement and response generation

3.2. Vector Representation:

- The embeddings generated by the model are dense vectors of real numbers that reflect the semantic properties of the text. For example, two similar conversations will have embeddings that are close together in the vector space, while dissimilar conversations will have embeddings that are farther apart. This allows for efficient similarity measurement and retrieval of relevant past conversations.

3.3. Storing in a Vector Database:

- The embeddings are then stored in a vector database (in this case, Elasticsearch) as dense vectors. Elasticsearch is used to index these embeddings, allowing for fast and scalable similarity searches. We configured Elasticsearch to store the embeddings with the appropriate dimensions, ensuring efficient retrieval during similarity queries.

3.4. Similarity Queries:

- To find the most relevant past conversations for a given query, we utilize cosine similarity, a metric that measures the angle between two vectors in the embedding space. When a new query is received (e.g., a customer query), the corresponding query embedding is compared against the stored embeddings in Elasticsearch. The top k most similar conversations are retrieved based on their cosine similarity scores.

3.5. Elastic Search Push:

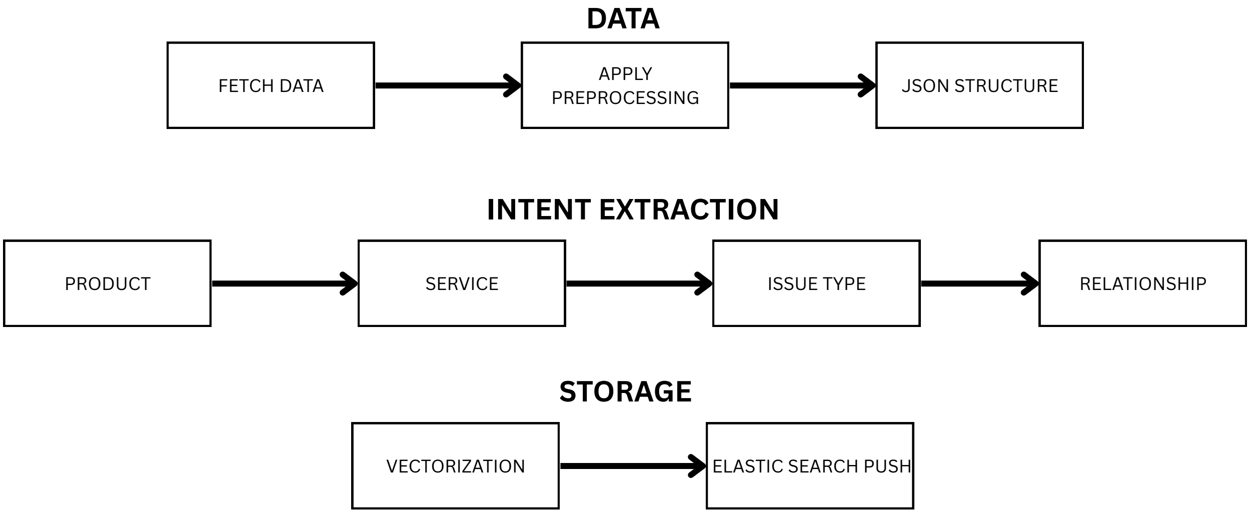
- After vectorizing the data, we push it into an Elasticsearch index. Elasticsearch is a powerful, distributed search engine that allows for quick retrieval of relevant information. By indexing the vectorized data, the model can perform fast, efficient searches and find the most relevant responses during inference. This process allows the RAG model to retrieve similar or matching conversations from the dataset to generate a relevant response based on the customer query.

4. Model Training and Inference

At this stage, the RAG model leverages the retrieved information from Elasticsearch, augmented with generative capabilities, to craft the final response. The retrieval step ensures that the model's responses are grounded in real, relevant past conversations, while the generation step creates responses tailored to the specific query, improving both accuracy and relevance.

This systematic approach ensures that the RAG model is both fast and accurate, effectively handling customer support queries by leveraging both retrieval and generation techniques. By utilizing advanced storage solutions and machine learning techniques, this model can continuously improve as more data is processed and indexed, ensuring an ever-improving customer experience.

Model Building Scheme



2.4. Query Processing in RAG Model: Detailed Explanation

The Query Processing step is a critical component of the Retrieval-Augmented Generation (RAG) model. It ensures that the customer's query is processed efficiently, relevant information is retrieved, and a coherent and accurate response is generated. The data processing and intent extraction steps are similar to those in the model-building phase. However, in this stage, the response generation step is added. Here's how it works:

1. Vectorization (Similar to Model Building Step)
Vectorization in this context refers to transforming the query and relevant data (such as past conversations, entities, and relationships) into vector embeddings. These embeddings represent the semantic meaning of the text in a high-dimensional space.

- Purpose: To convert the raw text (e.g., customer queries) into numerical representations that can be used for similarity measurement and retrieval.
- How: We use pre-trained models like SentenceTransformer to create embeddings for the query and the data, capturing the semantic meaning of the text.
- Output: A dense vector for each query or conversation, which can be compared with other vectors to measure similarity.

This step is identical to the model-building phase where embeddings were created for conversations, entities, and relationships.

2. Elasticsearch Top 50 Retrieval
Once the query is vectorized, the next step is to use the vectorized query to search the vector database (usually Elasticsearch) for the most similar historical conversations or documents.

- Purpose: To retrieve the top 50 most relevant past conversations based on their similarity to the input query.
- How: We use the cosine similarity between the query vector and the stored conversation vectors. Elasticsearch is configured to store embeddings as dense vectors, and it efficiently computes similarity scores for each document.
- Output: A list of the top 50 hits (conversations) from the Elasticsearch index, ranked by relevance to the query.

Elasticsearch is ideal for this task due to its ability to scale and perform fast, approximate nearest neighbor (ANN) searches on high-dimensional vectors.

3. Reranking

After retrieving the top 50 candidate conversations from Elasticsearch, the next step is reranking these results. Reranking ensures that the results returned from the vector database are refined further to provide the most contextually appropriate answers.

- Purpose: To improve the relevance of the top retrieved documents by applying a cross-encoder model (or any other sophisticated reranking model).
- How: The cross-encoder model processes both the query and each candidate conversation together, considering both simultaneously to generate a more accurate relevance score. This model is trained to evaluate the pair (query, candidate conversation) and rank it accordingly.
- Output: The reranked list of conversations, now with higher accuracy and relevance.
- Reranking provides an additional layer of refinement after the initial retrieval from Elasticsearch, ensuring that only the most relevant information is kept.

4. Top 5 Selection

Once the reranking process has been completed, the next step is to select the top 5 most relevant conversations for the response generation step.

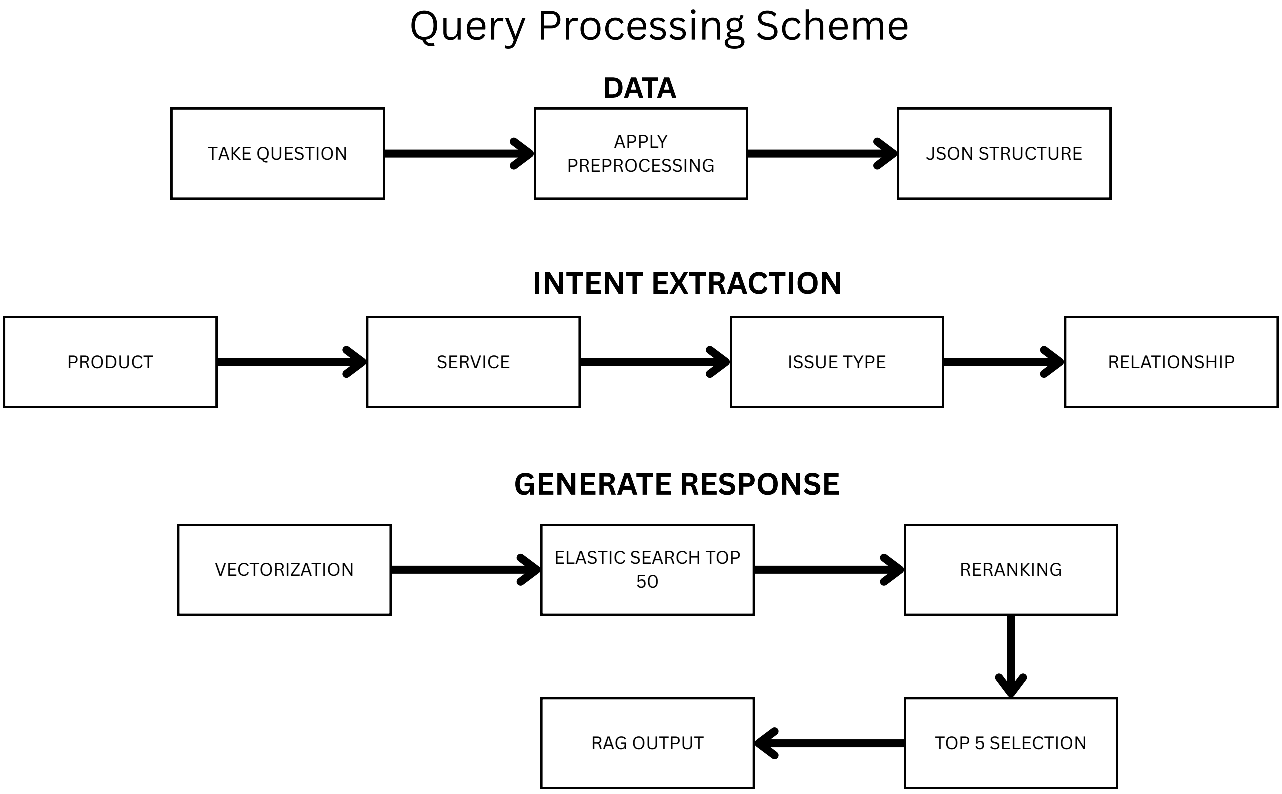
- Purpose: To narrow down the results to the 5 most relevant and high-ranking conversations that will be used to generate the final response.
- How: The top 5 conversations are selected based on their rerank scores. These conversations are the most relevant to the user's query and are likely to provide the best context for generating a meaningful response.
- Output: A subset of the top 5 selected conversations, ready to be passed on for response generation.

5. RAG Output (Response Generation)

In this final step, the Retrieval-Augmented Generation (RAG) model generates a response based on the retrieved and reranked conversations.

- Purpose: To generate a coherent, contextually appropriate, and informative response based on the top 5 selected conversations.
- How: The system passes the query and the top 5 retrieved conversations (along with any additional contextual information, such as entities and relationships) to a generative model like GPT. The model uses both the retrieved data and the query to generate a detailed response.
- Output: A response that is contextually relevant to the query, leveraging both the retrieved information and the generation capabilities of the model.

This process ensures that the system not only retrieves relevant data but also generates responses that are contextually accurate and informative, significantly improving the quality of customer interactions in real-time.



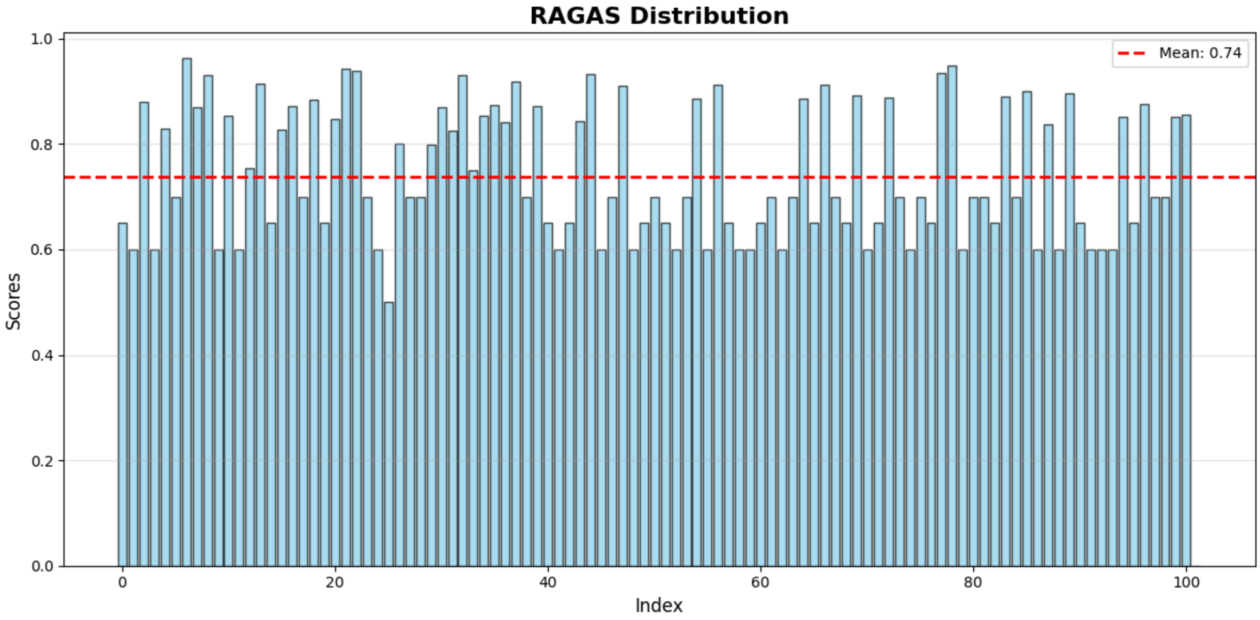
3. Results

The performance of our Retrieval-Augmented Generation (RAG) model was evaluated using the RAGAS (Retrieval-Augmented Generation Accuracy Score), a custom metric designed to assess both the accuracy and relevance of responses generated by the model in customer support scenarios.

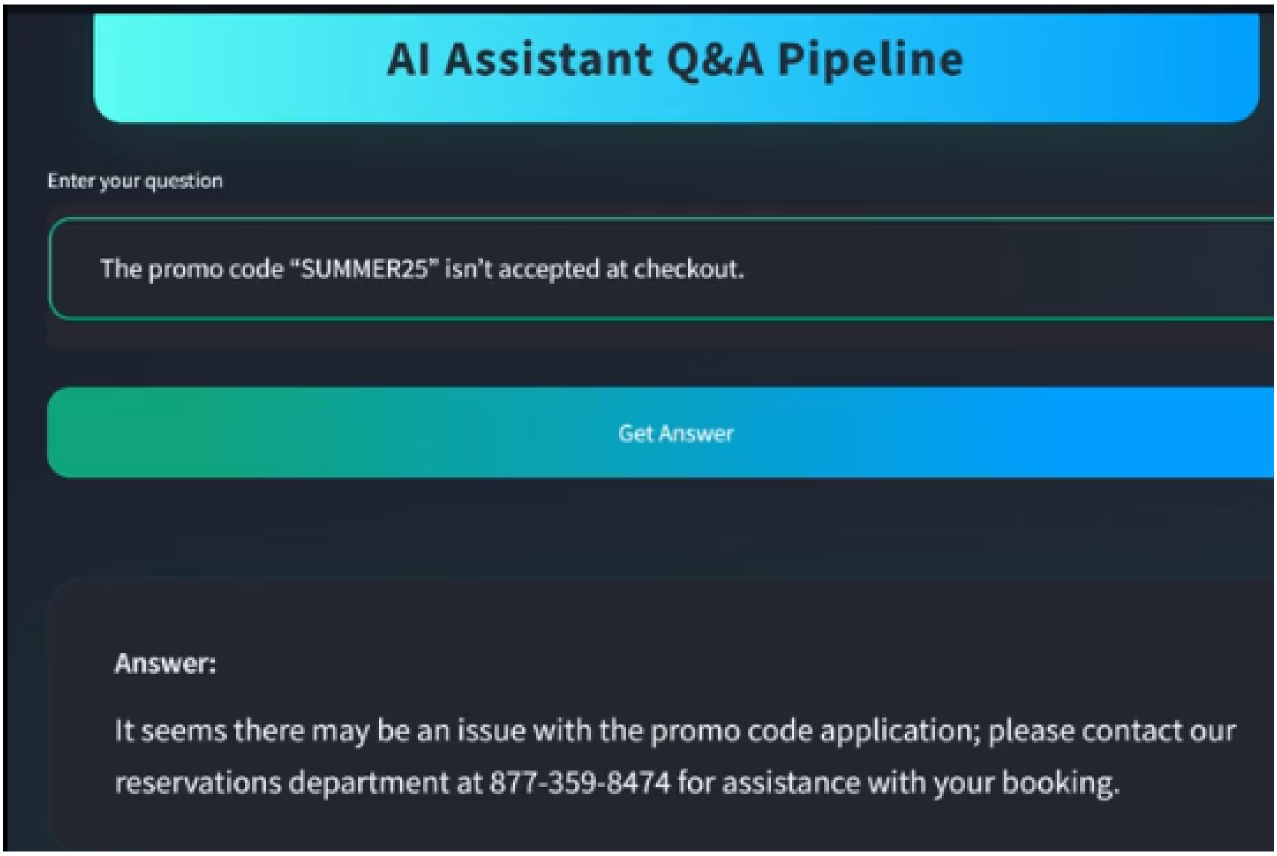
- RAGAS Score: The RAGAS score is a composite measure that combines retrieval accuracy and generative response relevance. It accounts for the model's ability to retrieve relevant documents from the knowledge base, as well as the quality and context alignment of the generated responses. The RAGAS score ranges from 0 to 100, with higher scores indicating better performance in terms of response accuracy and relevance.

- Test Setup: We tested the model on a dataset of 100 customer queries, each representing a variety of different customer support issues within the aviation sector. For each query, the model retrieved the top documents, re-ranked them, and generated a response. The responses were then evaluated against the ground truth answers by human evaluators who scored the responses based on accuracy, relevance, and contextual correctness.

- Performance: The model achieved a RAGAS score of 74, indicating that it provides highly accurate and relevant responses in 74% of the cases. This score reflects the model's ability to successfully retrieve and generate appropriate responses, while also demonstrating room for further improvement in response generation and retrieval accuracy.



Distribution of RAGAS score per test dataset



The promo code “SUMMER25” isn’t accepted at checkout.

Answer:

It seems there may be an issue with the promo code application; please contact our reservations department at 877-359-8474 for assistance with your booking.

Sample of execution