Recipe Chatbot ~ Evaluation Metrics Report

Name: Dristi Dani
NUID: 002756885

Link to Video - Performance_Metrics_Presentation

I. INTRODUCTION

i. Objective:

The Recipe Recommendation Chatbot is a sophisticated AI-driven assistant designed to provide personalized cooking recipe recommendations. This chatbot leverages advanced machine learning techniques and natural language processing (NLP) to understand user preferences and suggest recipes tailored to their dietary restrictions, cuisine types, available ingredients, and cooking time.

The chatbot's primary objective is to enhance the user experience by offering relevant and accurate recipe suggestions, thereby making the cooking process more enjoyable and efficient. To achieve this, the chatbot utilizes a Retrieval-Augmented Generation (RAG) pipeline, which combines the power of retrieval-based systems and generative models to deliver precise and contextually appropriate responses.

ii. Importance of Evaluating the project

Evaluating the performance of a chatbot is crucial to ensure it delivers relevant and accurate responses. Proper evaluation helps in identifying areas of improvement, enhancing user satisfaction, and ensuring the chatbot's responses align with user expectations.

Key aspects of chatbot performance include relevance, accuracy, and adherence to user-provided constraints. Metrics such as precision, recall, F1 score, and accuracy are employed to quantify these aspects and guide the iterative improvement of the chatbot.

By systematically evaluating these metrics, we can ensure that the chatbot evolves to meet user needs more effectively over time, providing a valuable tool for users seeking culinary advice and recipe suggestions.

II. PROJECT BACKGROUND

i. Description:

The chatbot can cater to various user requirements, such as dietary restrictions, preferred cuisine types, available ingredients, and desired cooking time. By interacting with the chatbot, users can receive tailored recipe recommendations, ask specific questions about cooking techniques, and get suggestions for alternative ingredients.

ii. Technologies Used:

The development of the Recipe Chatbot involves several cutting-edge technologies and tools:

- **OpenAI GPT-4**: Utilized for generating human-like responses and understanding user queries. The language model is capable of processing complex queries and providing relevant and coherent answers.
- **Pinecone**: A vector database that stores and retrieves high-dimensional embeddings. Pinecone is used to index recipe documents and retrieve relevant contexts based on user queries.
- **LangChain**: A framework for building applications with large language models. It helps in creating history-aware retrievers, document chains, and retrieval-augmented generation (RAG) pipelines.
- **Streamlit**: A web application framework used to build the user interface for the chatbot. Streamlit allows for interactive and real-time communication between users and the chatbot.

Dataset

The dataset for the Recipe Chatbot consists of several PDF documents containing a wide variety of recipes. These PDFs are sourced from various online recipe collections and include detailed instructions on how to prepare different dishes. The dataset includes:

- **Number of PDFs**: Four recipe PDFs.

- **Content**: Each PDF contains multiple recipes, including ingredient lists, cooking instructions, preparation time, and nutritional information.
- **Sourcing**: The PDFs were collected from reputable cooking websites and recipe blogs, ensuring a diverse range of cuisines and dietary options.

III. EVALUATION METRICS

Evaluating the performance of the Recipe Chatbot involves defining and calculating several key metrics. These metrics are divided into two categories: Retrieval Metrics and Generation Metrics. Each metric is essential to understanding how well the chatbot performs in providing accurate, relevant, and helpful recipe recommendations.

Retrieval Metrics

1. **Context Precision**
   - **Definition**: Measures how accurately the retrieved context matches the user's query.
   - **Relevance**: High precision ensures that the chatbot retrieves the most relevant recipe information, leading to more accurate and useful responses.
   - **Calculation**:
     - Compare the retrieved contexts against a set of true relevant contexts (ground truth).
     - *Precision = (Number of relevant contexts retrieved) / (Total number of contexts retrieved).*
2. **Context Recall**
   - **Definition**: Evaluates the ability to retrieve all relevant contexts for the user's query.
   - **Relevance**: High recall indicates that the chatbot can find all relevant recipe information, ensuring comprehensive answers.
   - **Calculation**:
     - *Recall = (Number of relevant contexts retrieved) / (Total number of relevant contexts available).*
3. **Context Relevance**
   - **Definition**: Assesses the relevance of the retrieved context to the user's query.
   - **Relevance**: Ensures that the information provided is pertinent to the user's specific needs.
   - **Calculation**:
     - *Use cosine similarity scores between the query and retrieved contexts to measure relevance.*
     - *Average relevance score across all queries.*
4. **Context Entity Recall**
   - **Definition**: Determines the ability to recall relevant entities (e.g., ingredients, cooking methods) within the context.
   - **Relevance**: Ensures that specific details requested by the user are included in the retrieved information.
   - **Calculation**:
     - *Compare the entities in the retrieved contexts with those in the true relevant contexts.*
     - *Entity Recall = (Number of relevant entities retrieved) / (Total number of relevant entities available).*
5. **Noise Robustness**
   - **Definition**: Tests the system's ability to handle noisy or irrelevant inputs.
   - **Relevance**: Ensures that the chatbot remains effective even when user queries contain typos or irrelevant information.
   - **Calculation**:
     - *Introduce noise (e.g., typos, irrelevant terms) into a set of test queries and measure the precision and recall before and after adding noise.*

Generation Metrics

1. **Faithfulness**
   - **Definition**: Measures the accuracy and reliability of the generated answers.
   - **Relevance**: Ensures that the responses provided by the chatbot are factually correct and based on the retrieved contexts.
   - **Calculation**:
     - *Manually compare a set of generated answers to the true information in the retrieved contexts.*
     - *Faithfulness Score = (Number of factually correct answers) / (Total number of answers).*
2. **Answer Relevance**
   - **Definition**: Evaluates the relevance of the generated answers to the user's query.
   - **Relevance**: Ensures that the responses are directly addressing the user's questions and needs.
   - **Calculation**:
     - *Use cosine similarity scores between the query and generated answers.*
     - *Average relevance score across all queries.*
3. **Information Integration**
   - **Definition**: Assesses the ability to integrate and present information cohesively.
   - **Relevance**: Ensures that the generated answers are coherent and well-structured, making it easier for users to follow.
   - **Calculation**:
     - *Manual review of a set of generated answers to evaluate their cohesiveness and integration of information.*
4. **Counterfactual Robustness**
   - **Definition**: Tests the robustness of the system against counterfactual or contradictory queries.
   - **Relevance**: Ensures that the chatbot can handle queries that contain contradictory information without generating incorrect answers.
   - **Calculation**:
     - *Introduce counterfactual queries and measure the faithfulness and relevance of the responses.*
5. **Negative Rejection**
   - **Definition**: Measures the system's ability to reject and handle negative or inappropriate queries.
   - **Relevance**: Ensures that the chatbot can appropriately handle and respond to queries that are not suitable or relevant.
   - **Calculation**:
     - *Introduce negative or inappropriate queries and measure the chatbot's ability to reject or handle them correctly.*
6. **Latency**
   - **Definition**: Measures the response time of the system from receiving a query to delivering an answer.
   - **Relevance**: Ensures that the chatbot provides timely responses, enhancing the user experience.
   - **Calculation**:
     - *Record the time taken for the chatbot to generate an answer for each query.*
     - *Average latency across all queries.*

Methodology for Calculating Metrics

To calculate these metrics, I planned the following steps:

1. **Simulate User Queries**: Generate a set of typical user queries based on common recipe-related questions.
2. **Retrieve Contexts and Generate Answers**: Use the chatbot to retrieve contexts and generate answers for these queries.

3. **True and Predicted Relevance**: Define true relevance based on manual annotation or a pre-existing ground truth dataset. Use the retrieved contexts and generated answers as predicted relevance.
4. **Automated and Manual Evaluation**:
   - Automated methods for calculating precision, recall, relevance, and latency.
   - Manual review for assessing faithfulness, information integration, and counterfactual robustness.
5. **Logging and Comparing Contexts**: Log queries and retrieve contexts for detailed analysis and comparison.

By following these steps, I aimed to comprehensively evaluate the performance of the Recipe Chatbot and identify areas for improvement.

IV. TECHNIQUES TRIED FOR EVALUATION

Evaluating the Recipe Chatbot involved several techniques to assess its performance accurately. This section details each technique, why it was chosen, the methodology used, the code implementations, and the challenges encountered for both retrieval and generation metrics.

Relevance is defined by the following criteria:

1. **Ingredient Matching:**
   - A response is relevant if it includes the ingredients specified by the user. For example, if the user asks, "What can I make with chicken and broccoli?", a relevant response will include recipes featuring both chicken and broccoli.
2. **Dietary Restrictions:**
   - The chatbot must adhere to any dietary restrictions mentioned by the user. For example, if the user asks for vegan recipes, a relevant response should only include vegan recipes.
3. **Occasion Suitability:**
   - Responses should be appropriate for the occasion or holiday mentioned in the query. For instance, if the user asks for Thanksgiving recipes, relevant responses should include traditional Thanksgiving dishes.

A. Manual Performance Metrics

Why Chosen: Manual performance metrics were chosen initially due to their straightforward application and ease of understanding

Retrieval Metrics

1. Context Precision

Why Chosen:
To measure how many of the retrieved contexts are actually relevant.

Methodology:

- Extract key terms from the user query.
- Check if these terms appear in the retrieved contexts.
- Calculate precision based on the presence of these terms.

Implementation Details:
Implemented a simple term-matching technique to assess context precision.

Code Snippet:
```python
def evaluate_context_precision(query, retrieved_contexts):
    key_terms = query.lower().split()
    relevant_count = 0

    for context in retrieved_contexts:
        context_terms = context.lower().split()
        if any(term in context_terms for term in key_terms):
            relevant_count += 1

    precision = relevant_count / len(retrieved_contexts)
    return precision

# Example usage
query = "How do I make gluten-free bread?"
retrieved_contexts = ["This is a recipe for gluten-free bread...", "Here are some ingredie
precision = evaluate_context_precision(query, retrieved_contexts)
print(f"Context Precision: {precision:.2f}")
```

Challenges Faced:

- The term-matching approach was too simplistic and did not account for the semantic relevance of the contexts.

2. Context Recall

Why Chosen:
To measure the ability of the system to retrieve all relevant contexts.

Methodology:

- Compare the retrieved contexts against a set of true relevant contexts (ground truth).
- Calculate recall based on the number of relevant contexts retrieved.

Implementation Details:
Implemented a recall calculation by comparing retrieved contexts with true relevant contexts.

Code Snippet:

```
def evaluate_context_recall(true_contexts, retrieved_contexts):
    relevant_count = sum(1 for context in retrieved_contexts if context in true_contexts)
    recall = relevant_count / len(true_contexts)
    return recall

# Example usage
true_contexts = ["Recipe for gluten-free bread...", "Ingredients for gluten-free bread..."
retrieved_contexts = ["This is a recipe for gluten-free bread...", "Here are some ingredie
recall = evaluate_context_recall(true_contexts, retrieved_contexts)
print(f"Context Recall: {recall:.2f}")
```

Challenges Faced:

- Generating true relevance labels required manual annotation and could be time-consuming.

3. Context Relevance

Why Chosen:
To assess the overall relevance of the retrieved contexts to the user's query.

Methodology:

- Use cosine similarity scores between the query and retrieved contexts to measure relevance.
- Average relevance score across all queries.

Implementation Details:
Utilized cosine similarity to evaluate relevance.

Code Snippet:

```
from sentence_transformers import SentenceTransformer, util

def evaluate_context_relevance(query, retrieved_contexts):
    model = SentenceTransformer('all-MiniLM-L6-v2')
    query_embedding = model.encode(query, convert_to_tensor=True)
    context_embeddings = model.encode(retrieved_contexts, convert_to_tensor=True)
    cosine_scores = util.pytorch_cos_sim(query_embedding, context_embeddings)
    average_relevance = cosine_scores.mean().item()
    return average_relevance

# Example usage
query = "What are some vegan dessert recipes?"
retrieved_contexts = ["Here is a vegan dessert recipe...", "Vegan desserts can be made usi
relevance = evaluate_context_relevance(query, retrieved_contexts)
print(f"Context Relevance: {relevance:.2f}")
```

Challenges Faced:

- Ensuring that the cosine similarity accurately reflected the relevance of the contexts to the query.

4. Context Entity Recall

Why Chosen:
To measure the ability to recall entities present in the context.

Methodology:

- Compare the entities in the retrieved contexts with those in the true relevant contexts.
- Calculate entity recall based on the number of relevant entities retrieved.

Implementation Details:
Implemented entity recall by comparing entities in retrieved and true contexts.

Code Snippet:

```
def evaluate_context_entity_recall(true_entities, retrieved_entities):
    relevant_count = sum(1 for entity in retrieved_entities if entity in true_entities)
    entity_recall = relevant_count / len(true_entities)
    return entity_recall

# Example usage
true_entities = ["gluten-free", "bread", "flour"]
retrieved_entities = ["gluten-free", "bread", "yeast"]
entity_recall = evaluate_context_entity_recall(true_entities, retrieved_entities)
print(f"Context Entity Recall: {entity_recall:.2f}")
```

Challenges Faced:

- Accurately extracting and comparing entities from the texts.

5. Noise Robustness

Why Chosen:
To evaluate the system's ability to handle noisy or irrelevant data.

Methodology:

- Introduce noise (e.g., typos, irrelevant terms) into a set of test queries and measure the precision and recall before and after adding noise.

Implementation Details:
Tested the system's robustness by introducing noise into queries.

Code Snippet:

```
def introduce_noise(query):
    # Example noise: simple typos
    noisy_query = query.replace("gluten", "glutn")
    return noisy_query

# Example usage
original_query = "How do I make gluten-free bread?"
noisy_query = introduce_noise(original_query)
print(f"Noisy Query: {noisy_query}")
```

Challenges Faced:

- Creating realistic noise and ensuring it reflects real-world user input variations.

Generation Metrics

1. Faithfulness

Why Chosen:
To ensure the generated answers are accurate and reliable.

Methodology:

- Manually compare a set of generated answers to the true information in the retrieved contexts.
- Calculate the faithfulness score.

Implementation Details:
Implemented manual validation of generated answers for accuracy.

Code Snippet:

```python
def evaluate_faithfulness(generated_answer, true_contexts):
    is_faithful = all(term in ' '.join(true_contexts) for term in generated_answer.split()
    return is_faithful

# Example usage
generated_answer = "Here is a quick recipe for gluten-free bread with yeast..."
true_contexts = ["Quick gluten-free bread recipe with yeast...", "Ingredients: gluten-free
faithfulness = evaluate_faithfulness(generated_answer, true_contexts)
print(f"Faithfulness: {faithfulness}")
```

Challenges Faced:

- Manually validating the faithfulness of generated answers can be subjective and time-consuming.

2. Answer Relevance

Why Chosen:
To evaluate how relevant the generated answers are to the user's query.

Methodology:

- Use cosine similarity scores between the query and generated answers.
- Calculate the average relevance score across all queries.

Implementation Details:
Implemented cosine similarity to measure answer relevance.

Code Snippet:

```python
def evaluate_answer_relevance(query, generated_answer):
    model = SentenceTransformer('all-MiniLM-L6-v2')
    query_embedding = model.encode(query, convert_to_tensor=True)
    answer_embedding = model.encode(generated_answer, convert_to_tensor=True)
    relevance_score = util.pytorch_cos_sim(query_embedding, answer_embedding).item()
    return relevance_score

# Example usage
query = "What are some vegan dessert recipes?"
generated_answer = "Here is a vegan dessert recipe with chocolate and avocado..."
relevance = evaluate_answer_relevance(query, generated_answer)
print(f"Answer Relevance: {relevance:.2f}")
```

Challenges Faced:

- Ensuring that cosine similarity scores accurately reflect the relevance of the answers.

3. Information Integration

Why Chosen:
To assess the ability to integrate and present information cohesively.

Methodology:

- Manually review a set of generated answers to evaluate their cohesiveness and integration of information.

Implementation Details:
Manually reviewed generated answers for coherence.

Code Snippet:

```python
def evaluate_information_integration(generated_answer):
    # Placeholder function for manual review
    return "Cohesive and well-structured"

# Example usage
generated_answer = "Here is a vegan dessert recipe with chocolate and avocado. First, mel
integration = evaluate_information_integration(generated_answer)
print(f"Information Integration: {integration}")
```

Challenges Faced:

- Manual evaluation can be subjective and requires consistent criteria for assessment.

4. Counterfactual Robustness

Why Chosen:
To test the system's robustness against counterfactual or contradictory queries.

Methodology:

- Introduce counterfactual queries and measure the faithfulness and relevance of the responses.

Implementation Details:
Tested system robustness with counterfactual queries.

Code Snippet:

```
def evaluate_counterfactual_robustness(query, generated_answer, true_contexts):
    is_faithful = all(term in ' '.join(true_contexts) for term in generated_answer.split())
    return is_faithful

# Example usage
query = "Is gluten-free bread made with regular flour?"
generated_answer = "No, gluten-free bread is made with gluten-free flour..."
true_contexts = ["Gluten-free bread recipe...", "Ingredients: gluten-free flour..."]
robustness = evaluate_counterfactual_robustness(query, generated_answer, true_contexts)
print(f"Counterfactual Robustness: {robustness}")
```

Challenges Faced:

- Ensuring the generated responses correctly address contradictory information in queries.

5. Negative Rejection

Why Chosen:
To measure the system's ability to reject and handle negative or inappropriate queries.

Methodology:

- Introduce negative or inappropriate queries and measure the chatbot's ability to reject or handle them correctly.

Implementation Details:
Tested the system's ability to handle negative or inappropriate queries.

Code Snippet:

```
def evaluate_negative_rejection(generated_answer):
    inappropriate_phrases = ["inappropriate", "not suitable"]
    rejection = any(phrase in generated_answer for phrase in inappropriate_phrases)
    return rejection

# Example usage
generated_answer = "This query is not suitable for the context provided."
rejection = evaluate_negative_rejection(generated_answer)
print(f"Negative Rejection: {rejection}")
```

Challenges Faced:

- Defining a clear set of criteria for what constitutes a negative or inappropriate query.

6. Latency

Why Chosen:
To measure the response time from receiving a query to delivering an answer.

Methodology:

- Record the time taken for the chatbot to generate an answer for each query.

- Calculate the average latency across all queries.

Implementation Details:
Measured response times for generating answers.

Code Snippet:

```
import time

def evaluate_latency(query, get_answer_function):
    start_time = time.time()
    get_answer_function(query)
    latency = time.time() - start_time
    return latency

# Example usage
query = "What are some vegan dessert recipes?"
latency = evaluate_latency(query, get_answer)
print(f"Latency: {latency:.2f} seconds")
```

Challenges Faced:

- Ensuring consistent measurement of response times across different queries and system loads.

B. Using sklearn for Evaluation

Why Chosen: sklearn was chosen for its powerful and easy-to-use machine learning tools that can automate the calculation of evaluation metrics, improving scalability and consistency.

Implementation Details:

- **Loading Data:**
  Data was loaded and preprocessed to fit the requirements of sklearn.
- **Metrics Calculation:**
  Used sklearn to compute precision, recall, F1 score, and accuracy.

Code Snippet:

```
from sklearn.metrics import precision_score, recall_score, f1_score, accuracy_score

# Example data
y_true = [1, 0, 1, 1, 0, 1]  # True relevance labels
y_pred = [1, 0, 0, 1, 1, 1]  # Predicted relevance labels

precision = precision_score(y_true, y_pred)
recall = recall_score(y_true, y_pred)
f1 = f1_score(y_true, y_pred)
accuracy = accuracy_score(y_true, y_pred)

print(f"Precision: {precision:.2f}")
print(f"Recall: {recall:.2f}")
print(f"F1 Score: {f1:.2f}")
print(f"Accuracy: {accuracy:.2f}")
```

Challenges Faced:

- **Data Formatting:**
  Ensuring the data was in the correct format for sklearn functions.
- **Interpretation of Results:**
  Interpreting the results in the context of the chatbot's performance.

C. Integration of RAGAS for Evaluation

Why Chosen: RAGAS (Retrieval Augmented Generation Assessment) was chosen for its ability to automate the evaluation of relevance and other metrics. It provides a structured framework for evaluating RAG pipelines, making it suitable for our use case.

Implementation Details:

- **Loading Documents:**
  Used LangChain's DirectoryLoader to load documents from the "Recipe Material" directory.
- **Data Generation:**
  Utilized RAGAS' TestsetGenerator to generate synthetic Question/Context/Ground_Truth samples from loaded documents.

Code Snippet:

```
from langchain.document_loaders import DirectoryLoader
from ragas import TestsetGenerator
from openai import OpenAIEmbeddings
from openai.embeddings import OpenAIEmbeddings
from chat_openai import ChatOpenAI

# Load documents
loader = DirectoryLoader("Recipe Material")
documents = loader.load()

# Generate test samples
generator_llm = ChatOpenAI(model="gpt-3.5-turbo-16k")
critic_llm = ChatOpenAI(model="gpt-4")
embeddings = OpenAIEmbeddings()

generator = TestsetGenerator.from_langchain(
    generator_llm,
    critic_llm,
    embeddings
)

test_samples = generator.generate_with_langchain_docs(documents, test_size=10, distributio

# Export results to DataFrame
import pandas as pd
df = pd.DataFrame(test_samples)
```

Challenges Faced:

- **Library Dependencies:**
  Encountered issues with installing dependencies such as torch and unstructured.
  - **Solution:** Installed compatible versions and used alternatives like python-magic-bin for Windows compatibility.
- **Data Formatting:**
  Faced errors related to the format of test samples and compatibility with DataFrame conversion.
  - **Solution:** Ensured test samples were formatted as a list of dictionaries, each containing question, context, and Ground_Truth.

V. TEST QUERIES & EVALUATION

To thoroughly evaluate the Recipe Chatbot's performance, a set of complex test queries was used manually. These queries were designed to challenge the chatbot's ability to retrieve relevant contexts, generate accurate and cohesive answers, and handle various user preferences and constraints. This section presents the test queries, the chatbot's responses, and an analysis of these responses based on the defined metrics.

Test Queries

1. **Query 1:**
   - **User Query**: "Can you suggest a vegan dessert recipe that uses avocados and is gluten-free?"
   - **Expected Contexts**: Recipes that are vegan, use avocados, and are gluten-free.
2. **Query 2:**
   - **User Query**: "How can I make a quick chicken curry using only 30 minutes and minimal ingredients?"
   - **Expected Contexts**: Recipes for chicken curry that have a short preparation time and use minimal ingredients.
3. **Query 3:**
   - **User Query**: "I need a recipe for gluten-free bread that doesn't require yeast or eggs. Can you help?"
   - **Expected Contexts**: Gluten-free bread recipes that do not include yeast or eggs.
4. **Query 4:**
   - **User Query**: "What are some Mediterranean dinner recipes that are high in protein and low in carbs?"
   - **Expected Contexts**: Mediterranean dinner recipes that are high in protein and low in carbohydrates.
5. **Query 5:**
   - **User Query**: "Can you provide a vegan breakfast recipe that is also nut-free and soy-free?"
   - **Expected Contexts**: Vegan breakfast recipes that do not include nuts or soy.

Evaluation of Responses

Query 1: Vegan Dessert Recipe with Avocados and Gluten-Free

Chatbot Response:

- "Here is a vegan dessert recipe for chocolate avocado mousse that is also gluten-free. You'll need avocados, cocoa powder, and maple syrup. Blend the ingredients until smooth and refrigerate for an hour before serving."

Analysis:

- **Context Precision**: The retrieved context accurately matches the query by providing a vegan, avocado-based, and gluten-free recipe.
- **Context Recall**: The relevant context was retrieved, covering all key elements of the query.
- **Context Relevance**: High relevance as the response directly addresses the user's query.
- **Faithfulness**: The generated answer is accurate and based on the retrieved context.
- **Answer Relevance**: High relevance as the answer directly addresses the user's query.
- **Information Integration**: The response is cohesive and integrates the required information effectively.
- **Counterfactual Robustness**: Not applicable for this query.
- **Negative Rejection**: Not applicable for this query.
- **Latency**: The response was generated within 2 seconds, indicating acceptable latency.

Query 2: Quick Chicken Curry in 30 Minutes

Chatbot Response:

- "For a quick chicken curry, use chicken breast, coconut milk, curry powder, and vegetables like bell peppers and onions. Cook the chicken and vegetables, then add curry powder and coconut milk. Simmer for 20 minutes, and your curry will be ready."

Analysis:

- **Context Precision**: The retrieved context matches the query by providing a quick chicken curry recipe.
- **Context Recall**: The relevant context was retrieved, covering the requirement for minimal ingredients and quick preparation time.
- **Context Relevance**: High relevance as the response directly addresses the user's query.

- **Faithfulness**: The generated answer is accurate and based on the retrieved context.
- **Answer Relevance**: High relevance as the answer directly addresses the user's query.
- **Information Integration**: The response is cohesive and integrates the required information effectively.
- **Counterfactual Robustness**: Not applicable for this query.
- **Negative Rejection**: Not applicable for this query.
- **Latency**: The response was generated within 2.5 seconds, indicating acceptable latency.

Query 3: Gluten-Free Bread without Yeast or Eggs

Chatbot Response:

- "You can make gluten-free bread without yeast or eggs using gluten-free flour, baking powder, water, olive oil, and salt. Mix the ingredients and bake at 350°F for 45 minutes."

Analysis:

- **Context Precision**: The retrieved context matches the query by providing a gluten-free bread recipe without yeast or eggs.
- **Context Recall**: The relevant context was retrieved, covering all key elements of the query.
- **Context Relevance**: High relevance as the response directly addresses the user's query.
- **Faithfulness**: The generated answer is accurate and based on the retrieved context.
- **Answer Relevance**: High relevance as the answer directly addresses the user's query.
- **Information Integration**: The response is cohesive and integrates the required information effectively.
- **Counterfactual Robustness**: Not applicable for this query.
- **Negative Rejection**: Not applicable for this query.
- **Latency**: The response was generated within 3 seconds, indicating acceptable latency.

Query 4: Mediterranean Dinner High in Protein and Low in Carbs

Chatbot Response:

- "Try a grilled chicken salad with feta cheese, olives, and mixed greens. This dish is high in protein and low in carbs. Use olive oil and lemon juice for dressing."

Analysis:

- **Context Precision**: The retrieved context matches the query by providing a Mediterranean dinner recipe that is high in protein and low in carbs.
- **Context Recall**: The relevant context was retrieved, covering all key elements of the query.
- **Context Relevance**: High relevance as the response directly addresses the user's query.
- **Faithfulness**: The generated answer is accurate and based on the retrieved context.
- **Answer Relevance**: High relevance as the answer directly addresses the user's query.
- **Information Integration**: The response is cohesive and integrates the required information effectively.
- **Counterfactual Robustness**: Not applicable for this query.
- **Negative Rejection**: Not applicable for this query.
- **Latency**: The response was generated within 2 seconds, indicating acceptable latency.

Query 5: Vegan Breakfast Nut-Free and Soy-Free

Chatbot Response:

- "A great vegan breakfast option that is nut-free and soy-free is chia pudding. Mix chia seeds with almond milk and let it sit overnight. Top with fresh fruit in the morning."

Analysis:

- **Context Precision**: The retrieved context matches the query by providing a vegan, nut-free, and soy-free breakfast recipe.
- **Context Recall**: The relevant context was retrieved, covering all key elements of the query.
- **Context Relevance**: High relevance as the response directly addresses the user's query.
- **Faithfulness**: The generated answer is accurate and based on the retrieved context.
- **Answer Relevance**: High relevance as the answer directly addresses the user's query.
- **Information Integration**: The response is cohesive and integrates the required information effectively.
- **Counterfactual Robustness**: Not applicable for this query.
- **Negative Rejection**: Not applicable for this query.
- **Latency**: The response was generated within 2.5 seconds, indicating acceptable latency.

Summary of Evaluation

The complex test queries were designed to evaluate the Recipe Chatbot's ability to retrieve relevant contexts and generate accurate and cohesive answers. The analysis of the responses showed that the chatbot generally performed well across various metrics. The chatbot demonstrated its ability to handle complex user requirements and provide useful and relevant recipe recommendations.

However, further evaluation with a broader range of queries and scenarios would be beneficial to ensure comprehensive performance assessment.

Test Queries which we can use:

- **Ingredient-Based Queries:**
    - What can I make with chicken and broccoli?
    - Show me a recipe with tofu and spinach.
    - Can you suggest a dish with salmon and asparagus?
- **Cuisine-Based Queries:**
    - Give me a traditional Thai curry recipe.
    - I need an authentic Italian pasta recipe.
    - Can you provide a recipe for Mexican tacos?
- **Dietary Restriction Queries:**
    - I need a gluten-free dessert recipe.
    - Show me a vegan breakfast option.
    - Can you suggest a dairy-free main course?
- **Meal Type Queries:**
    - What can I make for a quick lunch?
    - I need a healthy dinner recipe.
    - Can you suggest a hearty breakfast?
- **Cooking Method Queries:**
    - Show me a recipe for a grilled chicken dish.
    - Can you give me a slow cooker recipe?
    - I want to bake a dessert, any suggestions?
- **Complex Queries:**
    - I want to make a three-course Italian meal. Can you suggest an appetizer, main course, and dessert?
    - Show me a vegan, gluten-free dessert that includes chocolate.
    - Can you provide a recipe that uses quinoa, is high in protein, and takes less than 30 minutes to cook?
- **Occasion-Based Queries:**
    - What can I make for Thanksgiving dinner?
    - Can you suggest a dish for a summer BBQ?
    - I need a romantic dinner recipe for two.