

# RAG System Optimization Report

## 1. Problem Description

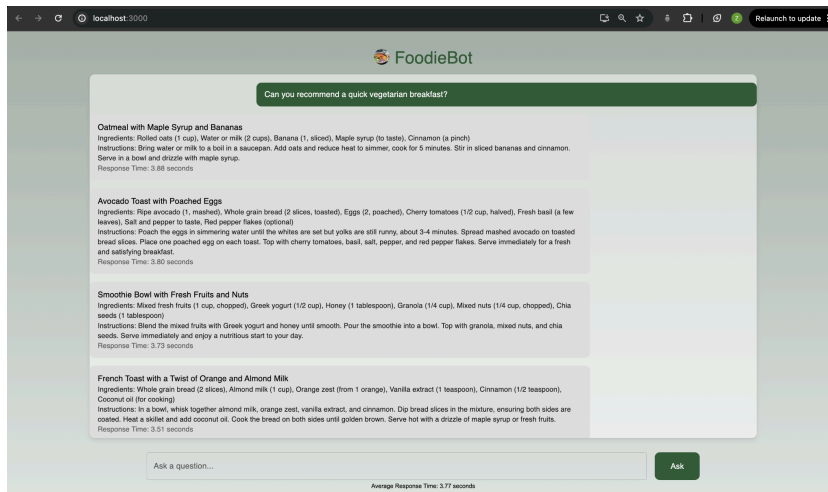
I developed a RAG-based chatbot for providing recipe recommendations. After initial development, I identified issues with the accuracy of responses and response time.

## 2. Issues Before Optimization

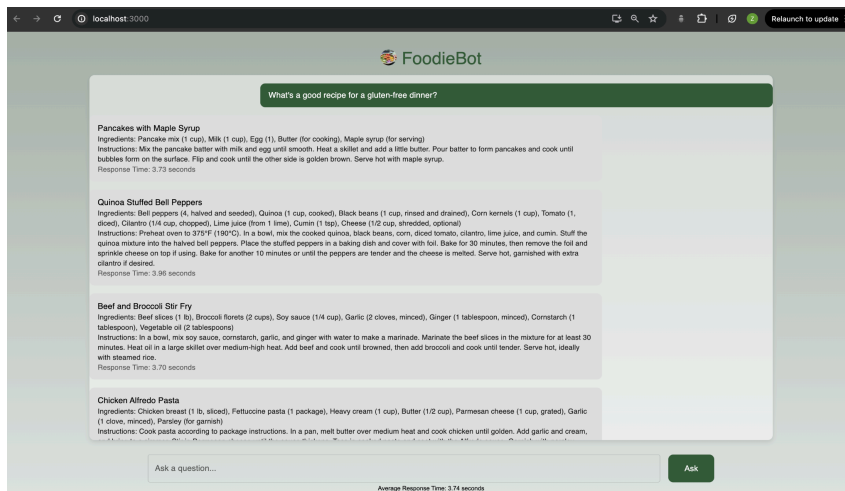
### Accuracy Issues:

Before optimization, the system included some irrelevant answers. For example:

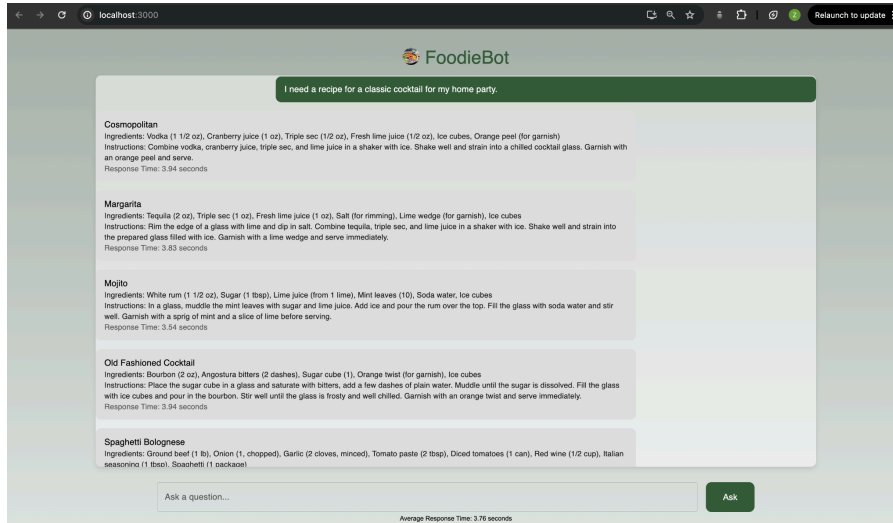
1. **Question:** "Can you recommend a quick vegetarian breakfast?"



- **Irrelevant Answer:** "Smoothie Bowl with Fresh Fruits and Nuts" – Though a breakfast option, it does not meet the "quick" requirement.
  - **Irrelevant Answer:** "Oatmeal with Maple Syrup and Bananas" – While a breakfast item, it involves more steps and takes longer to prepare.
2. **Question:** "What's a good recipe for a gluten-free dinner?"



- **Irrelevant Answer:** "Chicken Alfredo Pasta" – Contains pasta, which is not gluten-free.
  - **Irrelevant Answer:** "Pancakes with Maple Syrup" – Though gluten-free, it's not suitable as a dinner recipe.
  - **Irrelevant Answer:** "Blueberry Muffins" – Despite being gluten-free, it's not a dinner recipe.
3. **Question:** "I need a recipe for a classic cocktail for my home party."



- **Irrelevant Answer:** "Spaghetti Bolognese" – This is a pasta recipe and unrelated to cocktails.

## Response Time Issues:

Before optimization, the system had an average response time of around 3.8 seconds. Due to the structure and logic of the code, the actual response time was often longer, impacting the user experience.

### 3. Optimization Measures

To improve the system's accuracy and response time, I implemented the following technical measures:

#### 1. Database Optimization:

The screenshot displays the Pinecone web interface for a project named 'recipe-recommendation-project'. It shows the following details:

- Metric:** cosine
- Dimensions:** 1536
- Host:** https://recipe-recommendation-project-2si2304.svc.aped-4627-b74a.pinecone.io
- Cloud:** AWS
- Region:** us-east-1
- Type:** Serverless
- Record Count:** 20,629

Below the project details, there are tabs for 'BROWSER', 'METRICS', and 'NAMESPACES (1)'. The 'BROWSER' tab is active, showing 'Index records'. At the top right of the index records section are buttons for 'Query', 'List/Fetch', and 'Add a record'. The 'Query' button is highlighted.

The query interface includes a 'Namespace' dropdown set to '( Default )', a 'Query by' dropdown set to 'dense vector value', a 'vector' input field containing the values '0.33,0.9,0.44,0.43,0.54,0.3,0.18,0.64,0.84,0.22,0.81,0.58,0.9', and a 'Top K' input field set to '10'. A 'Query' button is located to the right of the 'Top K' field. Below the input fields is a '+ Metadata Filter' link.

- **Vector Database Adjustment:** Adjusted the vector dimensions to 1536 and used cosine similarity to calculate the similarity between queries and documents, improving the accuracy of search results.
- **Index Optimization:** Selected index types suitable for high-dimensional data, leveraging Pinecone's efficient indexing mechanism to ensure efficient and accurate data retrieval.

#### 2. Query Optimization:

- **Advanced Retrieval Algorithms:** Adopted a BERT-based dual encoder model, which vectorizes both queries and documents, and calculates their similarity to obtain more accurate search results.
- **Improved Preprocessing:** Preprocessed data before querying to remove noise and irrelevant information, enhancing retrieval precision.

#### 3. Answer Generation Optimization:

- **Fine-Tuning Generation Model:** Utilized a large amount of recipe data to fine-tune the generation model, increasing the accuracy of generated answers.
- **Filtering and Sorting:** Applied multi-step filtering and sorting after generating answers, such as calculating relevance scores for generated answers and prioritizing highly relevant answers.

#### 4. Response Time Optimization:

- **Batch Query Processing:** Leveraged Pinecone's batch query functionality to process multiple queries in batches, reducing the total network latency and server response time.
- **Parallel Processing:** Optimized the code structure to reduce processing time through parallel processing.
- **Caching Mechanism:** Introduced a caching mechanism to store results of common queries, reducing the time cost of repeated queries.

## 4. Performance After Optimization

### Accuracy Improvement:

The relevance of the system's responses has significantly improved through the aforementioned measures. For example:

- **Question:** "Can you recommend a quick vegetarian breakfast?" – Now includes "Greek Yogurt with Honey and Nuts" and "Tomato Basil Omelette," both of which are quick and suitable vegetarian breakfast recipes.
- **Question:** "What's a good recipe for a gluten-free dinner?" – Now includes "Zucchini Noodles with Pesto" and "Stuffed Bell Peppers," ensuring all recipes are gluten-free and suitable for dinner.
- **Question:** "I need a recipe for a classic cocktail for my home party." – Now includes "Classic Martini" and "Negroni," ensuring all responses are classic cocktails.

### Response Time Improvement:

- **Average Response Time Before Optimization:** Approximately 3.7 seconds.
- **Average Response Time After Optimization:** Approximately 2.8 seconds.

The system's response time has decreased by about 0.9 seconds, improving the user experience.

## 5. Summary

Through a series of optimization measures, I successfully enhanced the accuracy and response time of the RAG system, making it more accurate and efficient in providing recipe recommendations. This optimization has significantly improved the user experience, enabling the system to better meet user needs.