Evaluation Report

Introduction

In this report, we evaluate and implement methods to enhance the performance of a Retrieval-Augmented Generation (RAG) application, specifically focusing on improving latency and answer relevance. Addressing these aspects is crucial for delivering quick and accurate responses to user queries.

Evaluation Methodology

Our RAG application uses the Llama 3 model. We utilize the Phi-3 model as the evaluating LLM to ensure an unbiased assessment.

We measure the following criteria:

Correctness

The accuracy and truthfulness of the provided information in the response. Importance: Ensures that the information given is factually correct and reliable, maintaining the integrity of the application. Method: We use GPT-4 to generate question-answer pairs, manually evaluate and curate these, and use the GPT answers as ground truth. We employ the correctness function under labeled criteria from the langchain.evaluation library.

Relevance

The degree to which the response directly addresses the query posed by the user. Importance: Ensures that the response is pertinent to the user's question, enhancing user satisfaction and usability. Method: We use the relevance function under criteria from the langchain.evaluation library.

Coherence

The logical flow and clarity of the response, making it easy to understand. Importance: Ensures that the response is well-structured and logically organized, aiding user comprehension. Method: We use the coherence function under criteria from the langchain.evaluation library.

Conciseness

The brevity and efficiency of the response in conveying information without unnecessary detail. Importance: Ensures that responses are succinct, saving users' time and computational

resources. Method: We use the conciseness function under criteria from the langchain.evaluation library.

Noise robustness

The system's ability to handle noisy or irrelevant inputs. Importance: Ensures the system can differentiate between relevant and irrelevant queries, maintaining reliability. Method: We use GPT-4 to generate a list of irrelevant questions and evaluate if the model appropriately refrains from responding to these questions.

Latency

The time taken from query input to response output. Importance: Crucial for user experience, as faster responses lead to better engagement. Method: We use the time function to calculate end-to-end latency.

Improvements

1. Change Prompt to Generate Short, Concise Answers

We modified the prompt used in the generation phase to encourage the model to provide brief and to-the-point answers. This reduces the computational load and response time.

Previous Prompt

You are an AI assistant specialized in dietary guidelines. Use the following pieces of context to answer the question. If you don't know the answer, just say that you don't know, don't try to make up an answer.

Context: {context}

Question: {question}

Answer:

New prompt

You are an AI assistant specialized in dietary guidelines. Provide a brief, concise answer to the question using the given context. If you don't know the answer, simply state "I don't know."

Context: {context}

Question: {question}

Answer (in 50 words or less):

Results:

Metric	Before	After
Correctness Score	1	1
Relevance Score	0.8	1
Coherence Score	0.8	1
Conciseness Score	0.889	1
Latency (seconds)	7.3	6.4

2. Add Threshold to Retrieval

We implemented a threshold in the retrieval process. If the retrieved context's relevance score is below the threshold, the system directly responds with "I don't know" instead of running the

Old Retrieval Function

```
def search_faiss(query, k=3):
    query_vector = model.encode([query])[0].astype('float32')
    query_vector = np.expand_dims(query_vector, axis=0)
    distances, indices = index.search(query_vector, k)

    results = []
    for dist, idx in zip(distances[0], indices[0]):
        results.append({
            'distance': dist,
            'content': sections_data[idx]['content'],
            'metadata': sections_data[idx]['metadata']
        })
    return results
```

New Retrieval Function

```
def new_search_faiss(query, k=3, threshold=0.5):
    query_vector = model.encode([query])[0].astype('float32')
    query_vector = np.expand_dims(query_vector, axis=0)
    distances, indices = index.search(query_vector, k)


    results = []
    for dist, idx in zip(distances[0], indices[0]):
        if dist < threshold:  # Only include results within the threshold distance
            results.append({
                'distance': dist,
                'content': sections_data[idx]['content'],
                'metadata': sections_data[idx]['metadata']
            })
return results
```

Results

Metric	Before	After
Irrelevant Score	1	1
Latency (seconds)	5.499	0.016

Challenges Faced

While adding a threshold can sometimes reduce context, we need to iteratively run this to find the best threshold that balances the tradeoff between accuracy and latency. The optimal threshold may vary depending on the specific dataset and use case.

Conclusion

Implementing these methods to improve latency and answer relevance in our RAG application has significantly enhanced user experience. By generating concise answers and leveraging retrieval thresholds, we achieved a notable reduction in latency without compromising accuracy. The prompt modification improved all measured metrics, while the threshold implementation drastically reduced latency for irrelevant queries.

These enhancements contribute to a more efficient and reliable RAG application. However, continuous monitoring and fine-tuning of these improvements are necessary to maintain optimal performance as the system scales and encounters diverse query types.

Future Work

1. Implement adaptive thresholding based on query complexity.
2. Explore techniques for dynamic prompt adjustment based on user interaction patterns.
3. Investigate methods for further reducing latency, such as model quantization or distillation.
4. Conduct extensive user testing to gather feedback on the improved system performance.

By continuing to refine these aspects, we can further enhance the RAG application's efficiency, accuracy, and overall user satisfaction.