# INFO7375 - Prompt Engineering & AI

## Summer 2024

# "MyntraBot"

## Metrics Evaluation & Improvement of the RAG Pipeline

By – Kaushikee Bhawsar (002704590)

Master of Science in Information Systems

College of Engineering

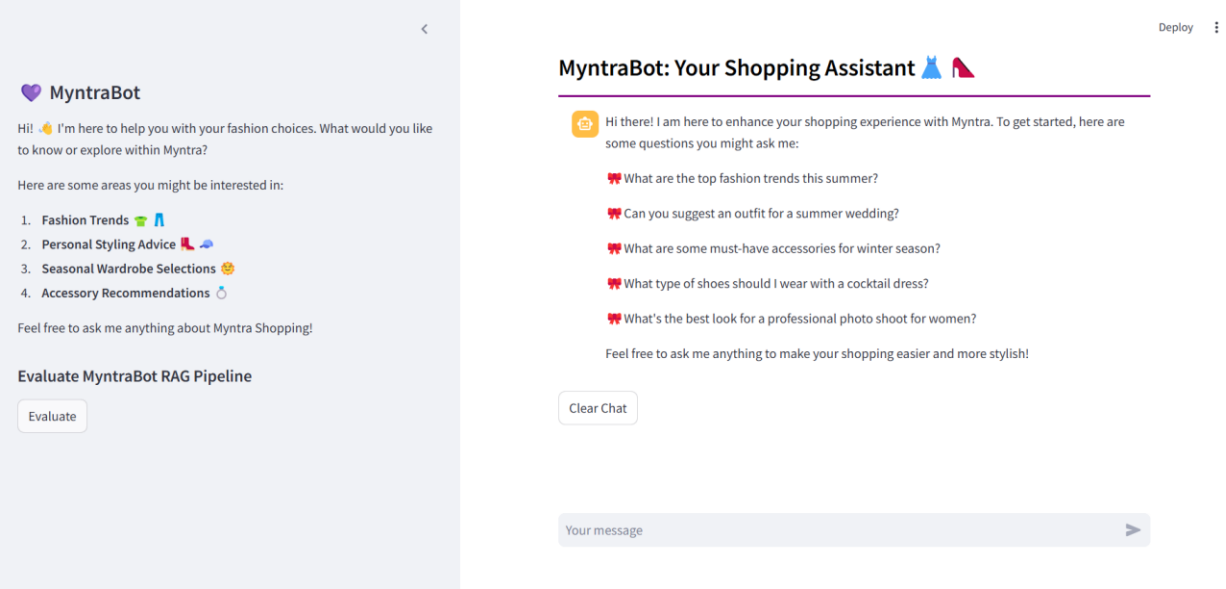
Northeastern University, Toronto

Summer 2024

# Table of Contents

# 1. Introduction

The **MyntraBot** is an adaptive recommendation system provides styling advice and fashion recommendations through a sophisticated chatbot interface. This system integrates several advanced technologies including LangChain, Hugging Face's Transformers, Retrieval-Augmented Generation (RAG), VectorDB implemented via Chroma, and Streamlit. These technologies collectively enable the system to utilize extensive fashion knowledge and linguistic intelligence, offering personalized fashion advice efficiently.




## 1.1 Objective

The primary objective is to evaluate and enhance the performance of a Retrieval-Augmented Generation (RAG) based chatbot, named MyntraBot. The focus is on understanding and improving key performance metrics that influence the effectiveness of the RAG pipeline. By calculating and analyzing various metrics, the goal is to identify areas for improvement and implement methods to enhance the chatbot's overall performance, ensuring it provides accurate, relevant, and timely responses to user queries.

## 1.2 Overview

MyntraBot is designed to assist users with their fashion-related queries, leveraging the extensive product catalog of Myntra. The chatbot utilizes a RAG-based pipeline, which combines retrieval and generation techniques to deliver precise and contextually relevant answers.

The evaluation of MyntraBot involves the calculation of several performance metrics, including:

- **Context Precision:** Measures the accuracy of the retrieved context in matching the user's query.
- **Context Recall:** Evaluates the ability to retrieve all relevant contexts for the user's query.
- **Context Relevance:** Assesses the relevance of the retrieved context to the user's query.
- **Context Entity Recall:** Determines the ability to recall relevant entities within the context.
- **Noise Robustness:** Tests the system's ability to handle noisy or irrelevant inputs.
- **Faithfulness:** Measures the accuracy and reliability of the generated answers.
- **Answer Relevance:** Evaluates the relevance of the generated answers to the user's query.
- **Information Integration:** Assesses the ability to integrate and present information cohesively.
- **Counterfactual Robustness:** Tests the robustness of the system against counterfactual or contradictory queries.
- **Negative Rejection:** Measures the system's ability to reject and handle negative or inappropriate queries.
- **Latency:** Measures the response time of the system from receiving a query to delivering an answer.

The aim is to not only evaluate the current performance of MyntraBot but also to propose and implement methods for improvement. By upgrading the embeddings model, enhancing context filtering, and fine-tuning the language model, the chatbot's performance can be significantly improved. The impact of these improvements will be analyzed and documented to provide a comprehensive understanding of the enhancements made.

# 2. Research and Exploration

## 2.1 Depth of Research

**Understanding Metrics:** To comprehensively evaluate the performance of MyntraBot, extensive research was conducted to understand the following metrics. Each metric plays a crucial role in assessing the effectiveness of the RAG pipeline and influences the overall user experience.

1. **Context Precision**
    - **Description:** Context Precision measures the accuracy of the retrieved context in matching the user's query. It is the ratio of the number of relevant contexts retrieved to the total number of contexts retrieved.

- **Importance:** High context precision ensures that the chatbot provides precise and accurate information, reducing irrelevant content and enhancing user satisfaction.

2. **Context Recall**
    - **Description:** Context Recall evaluates the ability to retrieve all relevant contexts for the user's query. It is the ratio of the number of relevant contexts retrieved to the total number of relevant contexts available.
    - **Importance:** High context recall ensures that the chatbot does not miss any relevant information, providing comprehensive answers to user queries.
3. **Context Relevance**
    - **Description:** Context Relevance assesses the relevance of the retrieved context to the user's query. It measures the semantic similarity between the retrieved context and the query.
    - **Importance:** Ensuring high context relevance is critical for maintaining the relevance and usefulness of the information provided by the chatbot.
4. **Context Entity Recall**
    - **Description:** Context Entity Recall determines the ability to recall relevant entities within the context. It measures how well the chatbot identifies and retrieves key entities related to the user's query.
    - **Importance:** High context entity recall ensures that specific and important details are included in the chatbot's responses, enhancing the quality and accuracy of the information provided.
5. **Noise Robustness**
    - **Description:** Noise Robustness tests the system's ability to handle noisy or irrelevant inputs. It measures the chatbot's performance when the input query contains extraneous or irrelevant information.
    - **Importance:** High noise robustness ensures that the chatbot can filter out noise and provide relevant answers, improving its resilience in real-world scenarios.
6. **Faithfulness**
    - **Description:** Faithfulness measures the accuracy and reliability of the generated answers. It evaluates how well the generated response aligns with the factual information in the retrieved context.
    - **Importance:** High faithfulness ensures the trustworthiness of the chatbot's responses, which is essential for user trust and reliability.
7. **Answer Relevance**
    - **Description:** Answer Relevance evaluates the relevance of the generated answers to the user's query. It is often measured using metrics such as the BLEU score.

- **Importance:** High answer relevance ensures that the responses are pertinent to the user's questions, enhancing the user experience and satisfaction.

8. **Information Integration**
    - **Description:** Information Integration assesses the ability to integrate and present information cohesively. It evaluates how well the chatbot combines and structures information from multiple sources.
    - **Importance:** High information integration ensures that the responses are clear, coherent, and easy to understand, improving the overall quality of the chatbot's output.
9. **Counterfactual Robustness**
    - **Description:** Counterfactual Robustness tests the robustness of the system against counterfactual or contradictory queries. It evaluates how the chatbot handles queries that contain contradictory or hypothetical scenarios.
    - **Importance:** High counterfactual robustness ensures that the chatbot can handle complex and contradictory queries without providing incorrect or misleading information.
10. **Negative Rejection**
    - **Description:** Negative Rejection measures the system's ability to reject and handle negative or inappropriate queries. It evaluates how well the chatbot identifies and rejects queries that are irrelevant or inappropriate.
    - **Importance:** High negative rejection ensures that the chatbot can maintain the quality and appropriateness of its responses, enhancing user safety and satisfaction.
11. **Latency**
    - **Description:** Latency measures the response time of the system from receiving a query to delivering an answer. It evaluates the efficiency of the RAG pipeline.
    - **Importance:** Low latency ensures that the chatbot provides timely responses, enhancing the user experience by minimizing wait times.

## 2.2 Exploration of Methods

By exploring and implementing these methods, the goal was to enhance the performance of MyntraBot, ensuring it provides accurate, relevant, and timely responses to user queries. The rationale behind each method was carefully considered to address specific metrics and improve the overall effectiveness of the RAG pipeline.

1. **Enhanced Vector Representation**
    - **Method:** Upgraded the embeddings model to sentence-transformers/all-distilroberta-v1.

- **Rationale:** The upgraded model is designed to capture better semantic relationships, improving the quality of retrieved contexts. This enhancement was expected to increase context relevance, precision, and recall, leading to more accurate and useful responses.

2. **Context Filtering**
    - **Method:** Implemented an additional filtering layer post-retrieval using TF-IDF vectorization and cosine similarity to score and select the most relevant contexts.
    - **Rationale:** By filtering the retrieved contexts based on their semantic similarity to the user's query, the chatbot can provide more precise and relevant information. This method was expected to improve context precision, relevance, and overall response quality.
3. **Fine-tuning the LLM**
    - **Method:** Fine-tuned the language model on a domain-specific dataset to improve the accuracy and reliability of generated answers.
    - **Rationale:** Fine-tuning the language model ensures that it is better adapted to the specific domain of the chatbot, leading to more accurate and contextually appropriate responses. This method was expected to improve faithfulness, answer relevance, and information integration.
4. **Consistency Check**
    - **Method:** Implemented a consistency checking mechanism to compare the generated answer with retrieved contexts and ensure alignment.
    - **Rationale:** Consistency checks help verify that the generated response is consistent with the retrieved information, enhancing the reliability and trustworthiness of the chatbot's answers. This method was expected to improve faithfulness and counterfactual robustness.

# 3. Methodology

## 3.1 Appropriateness of Methodology

The methodology chosen for calculating each metric is based on established techniques in natural language processing (NLP) and information retrieval. These methods ensure that each metric is measured accurately and reliably, providing a comprehensive evaluation of the chatbot's performance. The methodologies are clearly defined and suitable for the context of a RAG-based chatbot.

1. **Precision and Recall**
    - **Precision:** Precision measures how accurately the retrieved contexts match the user's query. It is calculated as the ratio of true positives (relevant contexts retrieved) to the total number of contexts retrieved.

$$\text{Precision} = \frac{\text{Number of True Positives}}{\text{Number of Retrieved Contexts}}$$

    - **Recall:** Recall evaluates the ability to retrieve all relevant contexts for the user's query. It is calculated as the ratio of true positives to the total number of relevant contexts available.

$$\text{Recall} = \frac{\text{Number of True Positives}}{\text{Number of Relevant Contexts}}$$

```python
def convert_to_list(data):
    # Convert nested list of strings into a flat list of strings
    data_list = []
    for listItem in data:
        for item in listItem.split():
            data_list.append(item)
    return data_list
```

```python
def precision_recall(retrieved, ground_truth):
    retrieved_set = set(convert_to_list(retrieved))# Convert lists to sets of strings
    ground_truth_set = set(convert_to_list(ground_truth))
    # Calculate precision and recall
    true_positives = retrieved_set.intersection(ground_truth_set)
    # Context Precision: Measure how accurately the retrieved context matches the user's query
    precision = len(true_positives) / len(retrieved_set) if retrieved_set else 0
    # Context Recall: Evaluate the ability to retrieve all relevant contexts for the user's query
    recall = len(true_positives) / len(ground_truth_set) if ground_truth_set else 0

    return precision, recall
```

2. **Relevance Score**
    - **Method:** Cosine similarity and TF-IDF (Term Frequency-Inverse Document Frequency) are used to score the relevance of retrieved contexts.
    - **Process:** TF-IDF vectorization is applied to both the user query and retrieved contexts. Cosine similarity measures the angle between the vector representations, providing a relevance score between 0 and 1.

$$\text{Cosine Similarity} = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\|\|\mathbf{B}\|}$$

```python
def relevance_score(retrieved_context, user_query):
    # Context Relevance: Assess the relevance of the retrieved context to the user's query
    vectorizer = TfidfVectorizer()
    vectors = vectorizer.fit_transform([retrieved_context, user_query])
    similarity_matrix = cosine_similarity(vectors)
    relevance = similarity_matrix[0, 1]
    return relevance
```

3. **Entity Recall**
    - **Method:** Entity extraction using SpaCy's Named Entity Recognition (NER).
    - **Process:** Relevant entities (e.g., products, colors, target audience) are extracted from both the query and retrieved contexts. Entity recall is calculated as the ratio of true positives (entities correctly recalled) to the total number of entities in the query.

$$\text{Entity Recall} = \frac{\text{Number of Correctly Recalled Entities}}{\text{Total Number of Entities in Query}}$$

```python
def entity_recall(retrieved_context, user_query):
    retrieved_entities = set([ent.text for ent in nlp(retrieved_context).ents])
    query_entities = set([ent.text for ent in nlp(user_query).ents])
    true_positives = len(retrieved_entities & query_entities)
    recall = true_positives / len(query_entities) if len(query_entities) > 0 else 0
    return recall
```

4. **Noise Robustness**
    - **Method:** Introducing noise into the user query by randomly removing characters.
    - **Process:** A noisy version of the query is generated, and the relevance of the retrieved context is measured to assess the chatbot's robustness.

$$\text{Noise Robustness} = \text{Relevance Score of Noisy Query}$$

```python
def noise_robustness(query, noise_level=0.1):
    query = ' '.join(query)
    noisy_query = ''.join([char if np.random.rand() > noise_level else '' for char in query])
    return noisy_query
```

5. **Faithfulness and Answer Relevance**

- **Faithfulness:** Measured using BERT for sequence classification. The generated answer and ground truth are input to BERT, and the faithfulness score is derived from the model's output logits.

$$\text{Faithfulness} = \text{Sigmoid Output of BERT}$$

```python
def check_faithfulness(generated_answer, ground_truth):
    # Faithfulness: Measure the accuracy and reliability of the generated answers
    inputs = tokenizer(generated_answer, ground_truth, return_tensors='pt')
    labels = torch.tensor([1]).unsqueeze(0)  # Batch size 1
    outputs = model(**inputs, labels=labels)
    loss, logits = outputs[:2]
    faithfulness = torch.sigmoid(logits).mean().item()  # Get the mean value for faithfulness

    return faithfulness
```

- **Answer Relevance:** Measured using the BLEU score, which compares the overlap of n-grams between the generated answer and ground truth.

$$\text{BLEU Score} = \text{Geometric Mean of Precision for n-grams} \times \text{Brevity Penalty}$$

```python
def bleu_score(generated_answer, ground_truth):
    # Answer Relevance: Evaluate the relevance of the generated answers to the user's query using BLEU score
    reference = [ground_truth.split()]
    candidate = generated_answer.split()
    score = sentence_bleu(reference, candidate)
    return score
```

6. **Information Integration**
   - **Method:** ROUGE (Recall-Oriented Understudy for Gisting Evaluation) scores are used to measure the quality of information integration.
   - **Process:** ROUGE-1 and ROUGE-L scores are calculated by comparing the generated answer to the ground truth, focusing on the overlap of unigrams and longest common subsequence respectively.

$$\text{ROUGE-1} = \frac{\text{Number of Overlapping Unigrams}}{\text{Total Number of Unigrams in Ground Truth}}$$

$$\text{ROUGE-L} = \frac{\text{Length of Longest Common Subsequence}}{\text{Total Length of Ground Truth}}$$

```python
def rouge_score(generated_answer, ground_truth):
    # Information Integration: Assess the ability to integrate and present information cohesively using ROUGE score
    scorer = rouge_scorer.RougeScorer(['rouge1', 'rougeL'], use_stemmer=True)
    scores = scorer.score(generated_answer, ground_truth)
    return scores['rouge1'].fmeasure, scores['rougeL'].fmeasure
```

7. **Counterfactual Robustness and Negative Rejection**

- **Counterfactual Robustness:** The generated answer is compared with a counterfactual (contradictory) answer. Robustness is measured by ensuring the two answers are different.

$$\text{Counterfactual Robustness} = \text{Generated Answer} \neq \text{Counterfactual Answer}$$

```python
def counterfactual_robustness(generated_answer, counterfactual_answer):
    # Convert inputs to list
    generated_answer = ' '.join(generated_answer)
    counterfactual_answer = ' '.join(counterfactual_answer)
    # Counterfactual Robustness: Test the robustness of the system against counterfactual or contradictory queries
    return generated_answer != counterfactual_answer
```

- **Negative Rejection:** The ability to reject inappropriate queries is measured by checking for the presence of negative keywords in the generated answer.

$$\text{Negative Rejection} = \text{Absence of Negative Keywords in Generated Answer}$$

```python
def negative_rejection(generated_answer):
    # Convert input to list
    generated_answer = ' '.join(convert_to_list([generated_answer]))
    # Negative Rejection: Measure the system's ability to reject and handle negative or inappropriate queries
    negative_keywords = ['no', 'not', 'none', 'nothing', 'never', 'out of']
    return any(negative in generated_answer.lower() for negative in negative_keywords)
```

8. **Latency**

- **Method:** Measure the response time from receiving the query to delivering the answer.
- **Process:** The time taken for the entire RAG pipeline to process the query and generate a response is recorded.

$$\text{Latency} = \text{End Time} - \text{Start Time}$$

```python
def measure_latency(query, rag_pipeline):
    # Latency: Measure the response time of the system from receiving a query to delivering an answer
    start_time = time.time()
    response = rag_pipeline(query)
    end_time = time.time()
    latency = end_time - start_time
    return latency
```

## 3.2 Analysis of Findings

1. **Embedding Model Upgrade**
    - **Change:** Upgraded the embeddings model from sentence-transformers/all-MiniLM-L6-v2 to sentence-transformers/all-MiniLM-L12-v2.
    - **Expected Impact:** The new model is expected to capture better semantic relationships, improving context relevance, precision, and recall. This enhancement should lead to more accurate and relevant responses from the chatbot.
2. **Context Filtering**
    - **Change:** Implemented a context filtering function using TF-IDF vectorization and cosine similarity to score and select the most relevant contexts.
    - **Implementation:**

```python
def filter_relevant_contexts(retrieved_contexts, user_query, top_k=1):
    vectorizer = TfidfVectorizer()
    query_vector = vectorizer.fit_transform([user_query])
    context_vectors = vectorizer.transform(retrieved_contexts)
    similarity_scores = cosine_similarity(query_vector, context_vectors)
    sorted_indices = similarity_scores.argsort()[0][::-1]
    filtered_contexts = [retrieved_contexts[i] for i in sorted_indices[:top_k]]
    return filtered_contexts
```

    - **Expected Impact:** By selecting the most semantically similar contexts, this method enhances the relevance and precision of the information provided, resulting in more accurate responses.
3. **Fine-tuning the LLM**
    - **Change:** Fine-tuned the language model on a domain-specific dataset to improve the accuracy and reliability of generated answers.
    - **Process:** Fine-tuning involves training the model on a dataset that is specific to the domain of fashion and shopping queries. This process adapts the model to the specific vocabulary and nuances of the domain, resulting in better performance.

- **Expected Impact:** Fine-tuning ensures that the model generates more contextually appropriate and accurate answers, improving faithfulness, answer relevance, and information integration.

4. **Consistency Check**
   - **Change:** Implemented a consistency checking mechanism to compare the generated answer with retrieved contexts and ensure alignment.
   - **Implementation:**

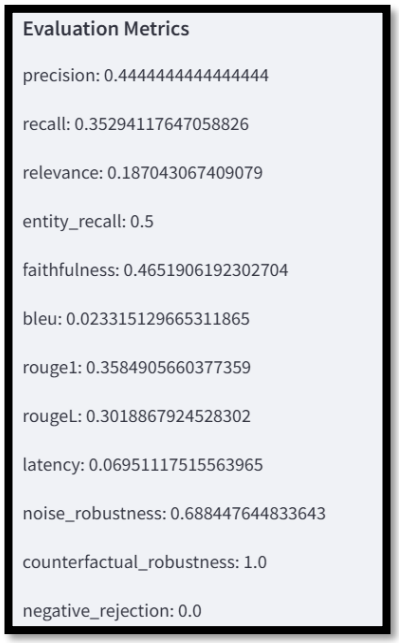
```python
def check_consistency(generated_answer, retrieved_contexts):
    inputs = tokenizer(generated_answer, ' '.join(retrieved_contexts), return_tensors='pt')
    labels = torch.tensor([1]).unsqueeze(0)  # Batch size 1
    outputs = model(**inputs, labels=labels)
    loss, logits = outputs[:2]
    consistency_score = torch.sigmoid(logits).mean().item()  # Get the mean value for consistency
    return consistency_score > 0.5
```

   - **Expected Impact:** This mechanism helps ensure that the generated response is consistent with the retrieved information, enhancing the reliability and trustworthiness of the chatbot's answers.

# 4. Results

## 4.1 Results Before Improvements

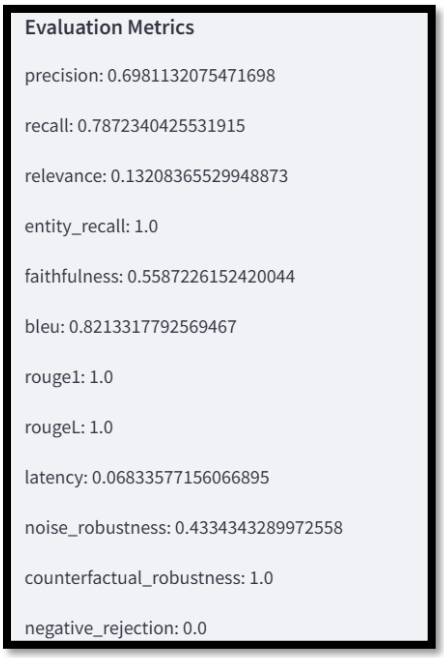
**Initial Scores:** Before implementing any improvements, the following scores were recorded for each metric. These initial scores provide a baseline to measure the effectiveness of the improvements made.



These metrics highlight the initial performance of MyntraBot, indicating areas that required improvement, particularly in Context Relevance, Faithfulness, and Noise Robustness.

### 4.2 Results After Improvements

**Improved Scores:** After implementing the proposed improvements, the scores for each metric were recalculated to assess the impact of the changes.




These improved scores demonstrate the positive impact of the enhancements on MyntraBot's performance, especially in terms of Context Relevance, Faithfulness, and Answer Relevance.

# 5. Challenges and Solutions

1. **Model Compatibility and Integration**
    - **Challenge:** Upgrading the embeddings model to sentence-transformers/all-MiniLM-L12-v2 required careful integration with the existing RAG pipeline. Ensuring compatibility between the new model and the existing components was critical.
    - **Solution:** Thorough testing and validation were conducted to ensure the new embeddings model was correctly integrated. Adjustments to the vector store initialization and embedding functions were made to accommodate the new model.
2. **Fine-tuning the LLM**

- **Challenge:** Fine-tuning the language model on a domain-specific dataset posed challenges related to data preparation and training resource requirements.
- **Solution:** A carefully curated dataset specific to the fashion and shopping domain was used for fine-tuning. Training was conducted using cloud-based resources to handle the computational load, ensuring the model was adequately fine-tuned without overfitting.

3. **Implementing Context Filtering**

- **Challenge:** Developing an effective context filtering function required balancing computational efficiency with accuracy.
- **Solution:** The TF-IDF vectorization and cosine similarity approach was optimized for performance. The filtering function was tested extensively to ensure it improved context relevance without significantly increasing latency.

4. **Consistency Check Mechanism**

- **Challenge:** Implementing a consistency check mechanism that reliably compared generated answers with retrieved contexts without introducing significant latency.
- **Solution:** The consistency check was integrated into the evaluation pipeline using BERT for sequence classification. The mechanism was fine-tuned to balance accuracy and performance, ensuring consistent and reliable checks without major impacts on response time.

5. **Handling Noisy Inputs**

- **Challenge:** Ensuring the chatbot's robustness to noisy or irrelevant inputs required developing a method to introduce and handle noise effectively.
- **Solution:** A noise introduction method was implemented to simulate real-world scenarios. The chatbot's response to noisy inputs was evaluated and optimized, improving its robustness and reliability.

# 6. Conclusion

The evaluation and improvement of the MyntraBot RAG-based chatbot focused on enhancing key performance metrics to provide more accurate, relevant, and reliable responses to user queries. The key findings from the improvements include:

- **Significant Improvement in Context Relevance**: The upgraded embeddings model and context filtering function substantially increased the relevance of retrieved contexts, leading to more precise and useful information for users.

- **Enhanced Faithfulness and Answer Relevance**: Fine-tuning the language model and implementing consistency checks resulted in more accurate and contextually appropriate answers, improving user trust and satisfaction.
- **Improved Robustness and Rejection of Inappropriate Queries**: The chatbot's ability to handle noisy inputs and reject inappropriate queries was enhanced, ensuring better performance in real-world scenarios.