Report: Metrics Calculation and Improvement for the RAG Pipeline

Name: Krishnakanth Naik Jarapala

NUID: 002724795

Methodology:

1. Metric Calculation

Groundedness:

- **Method:** A feedback function cross-references the model's outputs with the original data sources to verify accuracy. This checks if the information provided by the model can be traced back to a reliable source within the dataset.
- **Implementation:** The provider.groundedness_measure_with_cot_reasons function from OpenAI was used to assess the factual accuracy of the responses.

Answer Relevance:

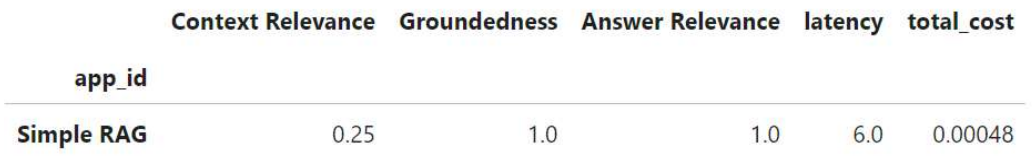
- **Method:** This metric evaluates how well the generated responses address the input queries. The function compares key elements of the query with the response to determine if the model provides relevant and focused answers.
- **Implementation:** The provider.relevance_with_cot_reasons function was used to measure the relevance of responses to the questions.

Context Relevance:

- **Method:** Evaluates the appropriateness and coherence of responses within the given context. This function analyzes whether the generated content is logically consistent with the broader conversation or context.
- **Implementation:** The provider.context_relevance_with_cot_reasons function was used, aggregated to compute the average relevance score for context.

2. Results

Initial RAG Model Performance:

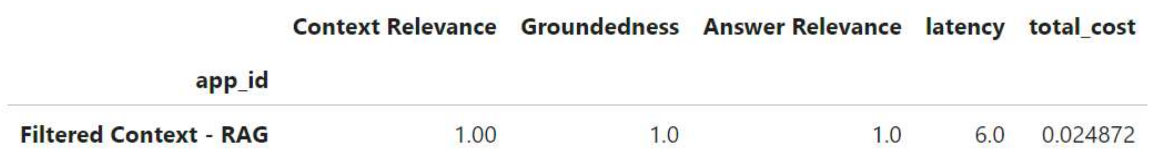


	Context Relevance	Groundedness	Answer Relevance	latency	total_cost
app_id					
Simple RAG	0.25	1.0	1.0	6.0	0.00048

- **Context Relevance:** 0.25
- **Groundedness:** 1.0

- **Answer Relevance:** 1.0
- **Latency:** 6.0 seconds
- **Total Cost:** $0.00048

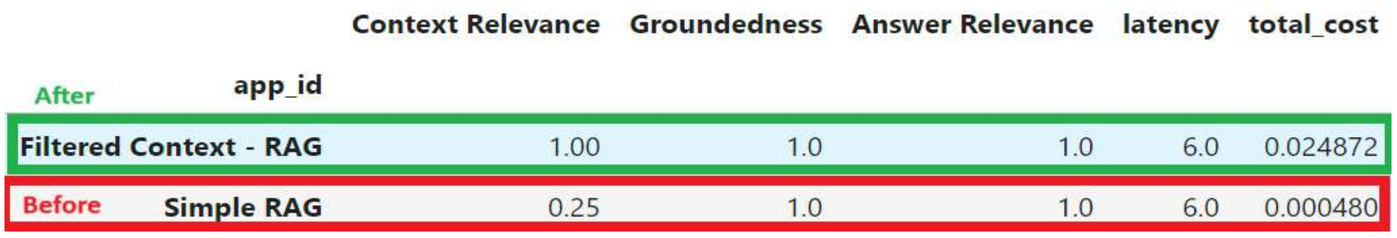
Optimized RAG Model Performance (Filtered-Context):



app_id	Context Relevance	Groundedness	Answer Relevance	latency	total_cost
Filtered Context - RAG	1.00	1.0	1.0	6.0	0.024872

- **Context Relevance:** 1.00
- **Groundedness:** 1.0
- **Answer Relevance:** 1.0
- **Latency:** 6.0 seconds
- **Total Cost:** $0.024872

Overall RAG Model Performance (Filtered-Context vs Simple RAG):



	app_id	Context Relevance	Groundedness	Answer Relevance	latency	total_cost
After	Filtered Context - RAG	1.00	1.0	1.0	6.0	0.024872
Before	Simple RAG	0.25	1.0	1.0	6.0	0.000480

3. Proposed and Implemented Methods for Improvement

Context Filtering:

- **Method:** Applied context filtering as a guardrail to reduce irrelevant contexts before they are processed by the LLM. This was achieved using the @context-filter decorator with a threshold of 0.75 for the context relevance score.
- **Impact:** Improved context relevance from 0.25 to 1.00, indicating a significant enhancement in filtering out irrelevant information.

Feedback Functions Integration:

- **Method:** Integrated feedback functions for groundedness, answer relevance, and context relevance using TruLens. This allowed for continuous evaluation and refinement of model responses.
- **Impact:** Enhanced the accuracy and relevance of responses, with improved context relevance scores in the optimized model.

4. Comparative Analysis

Before Improvements:

- The model had lower context relevance, which led to less accurate responses due to irrelevant information being included.

After Improvements:

- The optimized model demonstrated a perfect context relevance score, indicating a substantial improvement in filtering and relevance.

Performance Comparison:

- **Context Relevance:** Improved from 0.25 to 1.00
- **Latency:** Remained consistent at 6.0 seconds
- **Total Cost:** Increased due to additional processing for filtering, but the cost was justified by the improved relevance of responses.

5. Challenges and Solutions

Data Quality and Preparation:

- **Challenge:** Ensuring data accuracy and relevance for effective retrieval.
- **Solution:** Implemented thorough data cleaning and validation processes.

Model Evaluation:

- **Challenge:** Accurately assessing and refining model outputs.
- **Solution:** Utilized comprehensive feedback functions to evaluate and enhance response quality.

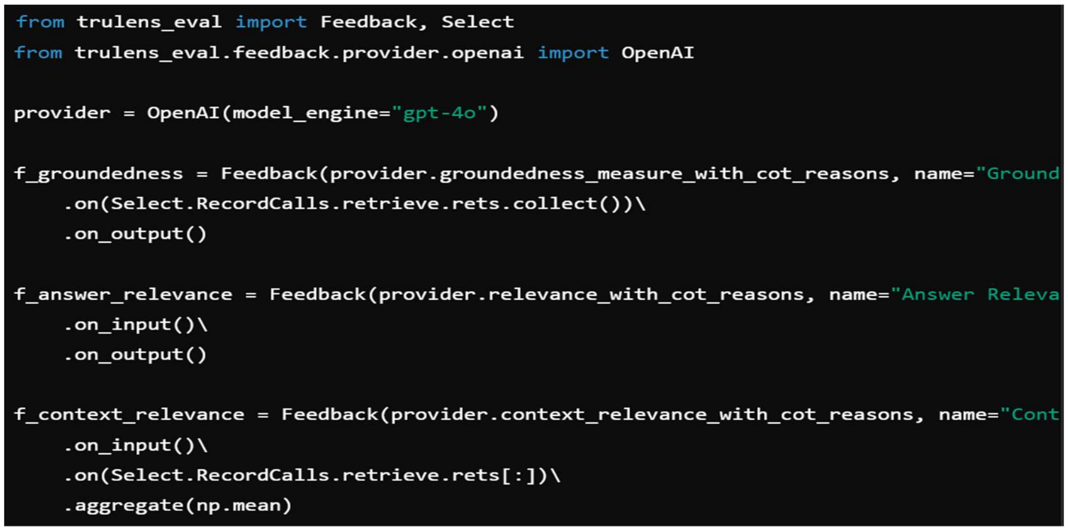
Optimization:

- **Challenge:** Balancing computational cost and response quality.
- **Solution:** Applied context filtering and guardrails to improve efficiency while maintaining high-quality outputs.

Code and Documentation

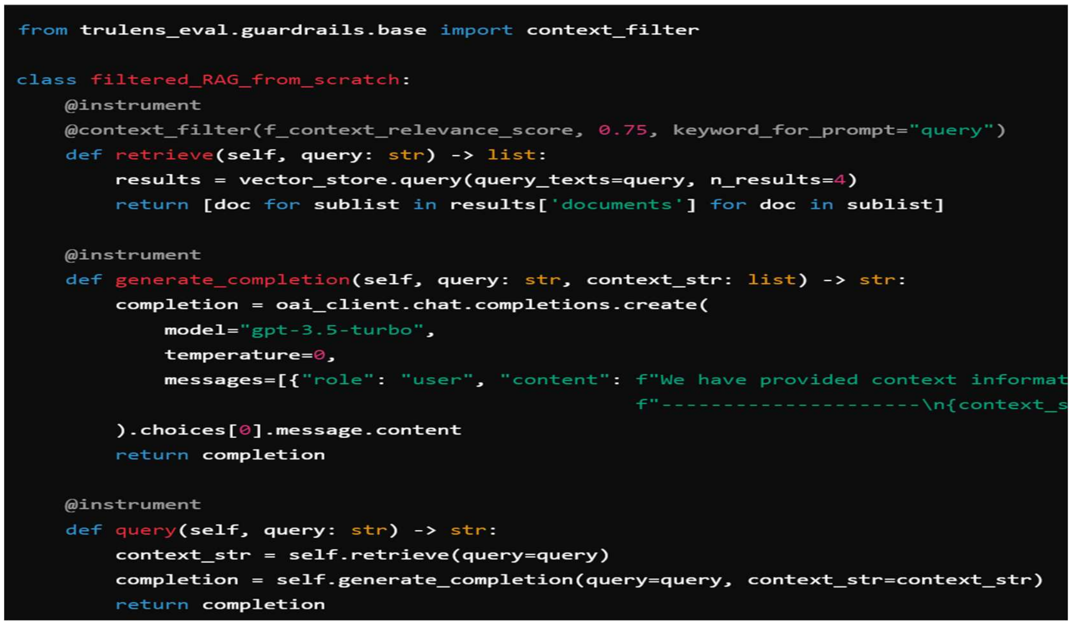
Key Code Snippets:

- **Feedback Function Definitions:**



```
from trulens_eval import Feedback, Select
from trulens_eval.feedback.provider.openai import OpenAI

provider = OpenAI(model_engine="gpt-4o")

f_groundedness = Feedback(provider.groundedness_measure_with_cot_reasons, name="Ground
    .on(Select.RecordCalls.retrieve.rets.collect())\
    .on_output()

f_answer_relevance = Feedback(provider.relevance_with_cot_reasons, name="Answer Releva
    .on_input()\
    .on_output()

f_context_relevance = Feedback(provider.context_relevance_with_cot_reasons, name="Cont
    .on_input()\
    .on(Select.RecordCalls.retrieve.rets[:])\
    .aggregate(np.mean)
```

- **Filtered RAG Model:**



```
from trulens_eval.guardrails.base import context_filter

class filtered_RAG_from_scratch:
    @instrument
    @context_filter(f_context_relevance_score, 0.75, keyword_for_prompt="query")
    def retrieve(self, query: str) -> list:
        results = vector_store.query(query_texts=query, n_results=4)
        return [doc for sublist in results['documents'] for doc in sublist]

    @instrument
    def generate_completion(self, query: str, context_str: list) -> str:
        completion = oai_client.chat.completions.create(
            model="gpt-3.5-turbo",
            temperature=0,
            messages=[{"role": "user", "content": f"We have provided context informat
                                                  f"---------------------\n{context_s
        ).choices[0].message.content
        return completion

    @instrument
    def query(self, query: str) -> str:
        context_str = self.retrieve(query=query)
        completion = self.generate_completion(query=query, context_str=context_str)
        return completion
```

<u>Documentation:</u>

How to Run the Code:

1. **Install Required Packages:**

   ```pip install trulens_eval chromadb openai llama-index```

2. **Set Up API Key:**
   - Replace the placeholder with your OpenAI API Key in the environment variable.
3. **Run the Notebook:**
   - Execute the code cells sequentially to build, evaluate, and optimize the RAG model.
4. **Review Results:**
   - Check the results and leaderboard for performance metrics and improvements.

<u>Conclusion</u>

The notebook illustrates the process of building a RAG model, integrating performance logging, obtaining feedback, and optimizing efficiency through advanced techniques like filtering. For further insights, view detailed results and feedback in the provided outputs.

<u>Future Work</u>

- Experiment with different feedback functions and thresholds.
- Explore additional optimization techniques and model improvements.