**Llamaindex + RAGChecker + GPT-4o = AI RAG Chatbot**


If you have been following me, you know that choosing a RAG retrieval method that suits your use can be daunting.

The RAG pipeline is quickly becoming the industry standard for developing chatbots that retrieve and inject external information into the prompt of an LLM call to reduce hallucinations, maintain up-to-date information, and leverage domain-specific knowledge.

However, optimizing RAG pipelines presents unique challenges, as does any cutting-edge technology—these range from ensuring the relevance of retrieved information to balancing computational efficiency with output quality.

Enter RAGChecker is a comprehensive evaluation framework for Retrieval-Augmented Generation (RAG) systems. It provides a suite of metrics to assess RAG systems' retrieval and generation components.

This article will provide a comprehensive overview of RAGChecker, exploring its key features, the metrics it utilizes, the difference between RAGChecker and Ragas, and how you can leverage RAGChecker to build an effective chatbot.

## What are RAGChecker

RAGCheker is a framework for detailed diagnosis of Retrieval-Augmented Generation (RAG) systems and evaluation of the performance of each retriever and generator module. While

traditional evaluation metrics (e.g., Recall@k, BLEU, ROUGE, BERTScore, etc.) are specialized for short responses, RAGCheker enables detailed evaluation of each claim (an individual assertion or piece of information in a response generated by a RAG system) and more accurately evaluates a RAG system's performance.

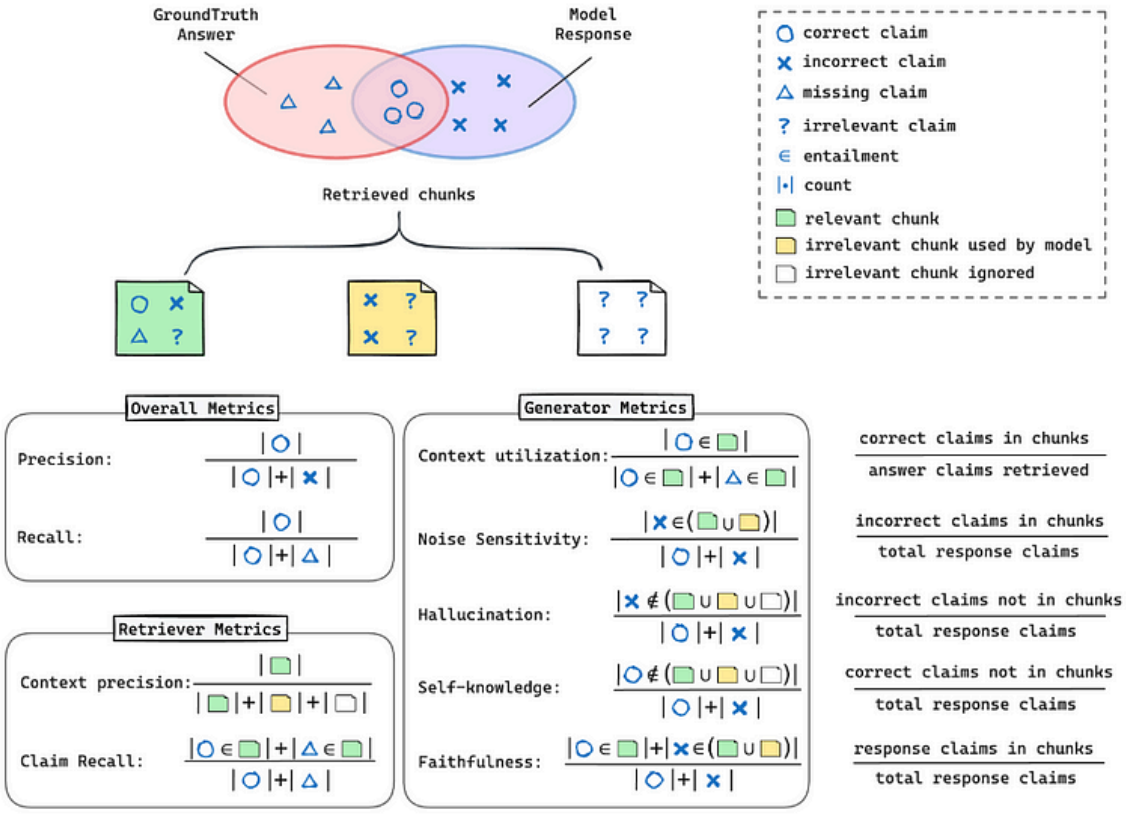



Figure: RAGChecker Metrics

## Features

1. **Purpose**: RAGChecker was developed to address the need for robust evaluation tools tailored for RAG models, combining retrieval and generation capabilities. It helps researchers and practitioners understand the effectiveness of their models in real-world applications.
2. **Evaluation Metrics**: The framework includes various evaluation metrics that assess both the retrieval and generation components. These include traditional metrics like precision, recall, and F1 score, as well as generation-specific metrics such as BLEU and ROUGE.
3. **Benchmarking**: RAGChecker allows for benchmarking against established datasets, enabling users to compare their models' performance with state-of-the-art systems. This benchmarking capability is crucial for advancing research in the field.
4. **User-Friendly Interface**: The tool is designed to be accessible, providing a user-friendly interface that simplifies the evaluation process. This makes it easier for users with varying levels of technical expertise to utilize the framework.

5. **Custom Dataset Support**: Users can input their datasets for evaluation, allowing for tailored assessments that align with specific use cases or domains. This flexibility enhances the applicability of RAGChecker across different applications.
6. **Insights for Model Improvement**: Users can identify strengths and weaknesses in their models by analyzing the results generated by RAGChecker. This feedback loop is essential for the iterative development and optimization of RAG systems.
7. **Contribution to NLP Research**: The introduction of RAGChecker contributes to the broader field of natural language processing (NLP) by providing a standardized method for evaluating RAG models, thereby facilitating progress and innovation in this area.

## Evaluate the RAG pipeline

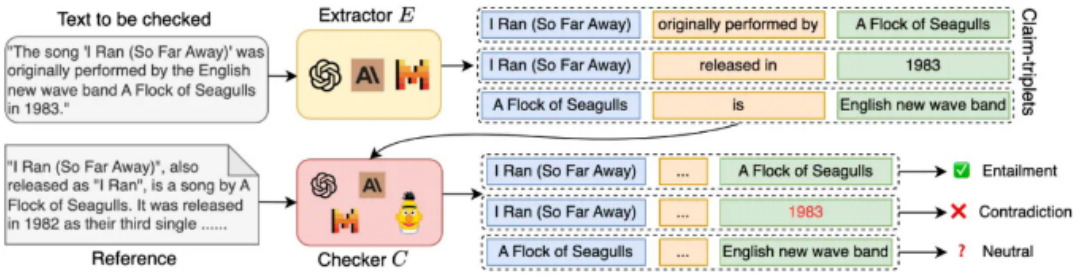
The Retrieval-Augmented Generation (RAG) pipeline consists of two main components: the retriever and the generator. Evaluating the RAG pipeline involves assessing both components' performance and their interaction. Here's a breakdown of how to evaluate the RAG pipeline:

1. **Retriever Evaluation**:

- **Effectiveness**: Measure how well the retriever identifies and retrieves relevant documents or information from a knowledge base. Metrics such as precision, recall, and F1-score can be used to evaluate the relevance of the retrieved documents.
- **Diagnostic Metrics**: Use specific metrics to identify strengths and weaknesses in the retrieval process. For example, metrics that assess the diversity of retrieved documents or the context's completeness can help diagnose retrieval errors 2.

**2. Generator Evaluation:**

- **Response Quality**: Evaluate the quality of the generated responses based on coherence, relevance, and factual accuracy. Traditional metrics like BLEU, ROUGE, and newer embedding-based metrics (e.g., BERTScore) can be employed, but they may need help with more extended responses 1.
- **Diagnostic Metrics**: Assess how well the generator utilizes the retrieved context. Metrics that evaluate the generator's ability to handle noisy information and produce accurate responses are crucial for understanding.




## RAGChecker Metrics

RAGCheker offers a set of metrics for assessing each component of an RAG pipeline and evaluating the entire pipeline from start to finish.

Here are some evaluation metrics commonly used in the context of RAG systems:

1. Overall evaluation indicators

**Precision**

- **Definition**: Evaluate the percentage of correct claims (claims that match the correct response) among the generated responses. Precision is a metric to evaluate how accurately the RAG system generates information. High Precision indicates that most generated claims are based on correct information.

Formula :

$$\text{Precision} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}}$$

**Recall**

- **Definition**: Evaluate the proportion of generated claims among correct responses. Recall is a metric that evaluates how well a system can recall correct information and is essential when evaluating the coverage of a system.

Formula :

$$\text{Recall} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}}$$

**F1 Score**

- **Definition**: It measures the harmonic mean of Precision and Recall. The F1 score is a metric that balances Precision and Recall and is used to evaluate the overall performance of a system without bias towards one side.

**Formula :**

$$\text{F1 Score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

**2. Generation Metrics**

**Faithfulness**

- **Definition**: Measures how faithful the generator's generated claims are to the chunks returned by the retriever. Faithfulness measures whether the generator is generating responses based on accurate information. A high value indicates less false information is generated.

Formula :

$$\text{Faithfulness} = \frac{\text{Number of correct claims included in the retriever's chunks}}{\text{Total number of generated claims}}$$

**Hallucination**

- **Definition**: Measures the percentage of errors the generator generates based on information not returned by the retriever. Hallucination measures the frequency of false claims the generator generates and is an index to evaluate the model's reliability. A lower value indicates less generation of false information.

Formula :

$$\text{Hallucination} = \frac{\text{Number of erroneous claims generated without relying on the retriever}}{\text{Total number of generated claims}}$$

**Self-knowledge**

- **Definition**: Measures the percentage of correct claims the generator generates based on knowledge. Self-knowledge measures the generator's ability to generate accurate responses based on its training data without relying on information from the retriever. A high value indicates that the generator has reliable self-knowledge and can be used appropriately.

Formula :

$$\text{Self-knowledge} = \frac{\text{Number of correct claims generated based on self-knowledge}}{\text{Total number of generated claims}}$$

**Context Utilization**

- **Definition**: Measures how effectively the generator utilizes retrieved context information. Context Utilization measures the generator's ability to leverage information from the retriever to generate accurate claims properly. High values indicate responses that accurately reflect the information obtained from the retriever.

Formula :

$$\text{Context Utilization} = \frac{\text{Number of correct claims (based on retriever's information)}}{\text{Total number of generated claims}}$$

**Relevant Noise Sensitivity**

- **Definition**: Measures the sensitivity to noise mixed in with chunks of correct information. This metric measures the generator's susceptibility to noise (unwanted information). A low value indicates that the generator is tolerant to noise.

**Formula :**

$$\text{Relevant Noise Sensitivity} = \frac{\text{Number of erroneous claims due to noise in correct chunks}}{\text{Total number of generated claims}}$$

**3. Retriever Metrics**

**Claim Recall**

- **Definition**: Measures how well the chunks returned by the retriever cover claims in the correct response. Claim Recall is an index that evaluates how much accurate information the retriever retrieves. A high value indicates that the retriever is selecting the correct information.

**Formula :**

$$\text{Claim Recall} = \frac{\text{Number of correct claims returned by the retriever}}{\text{Total number of correct claims}}$$

**Context Precision**

- **Definition**: Measures the percentage of chunks returned by the retriever that contain correct claims. Context Precision measures how relevant the retriever is in returning information. A high value indicates that the retriever can extract accurate information.

Formula :

$$\text{Context Precision} = \frac{\text{Number of chunks containing correct claims}}{\text{Total number of chunks returned by the retriever}}$$

## Ragas Vs RAGChecker

RAGChecker and RAGAs evaluate Retrieval-Augmented Generation (RAG) models but serve different purposes. RAGChecker provides a broad framework for benchmarking

overall RAG model performance, focusing on metrics like accuracy and relevance. RAGAs, on the other hand, offer specific metrics to assess individual components of the RAG pipeline, such as Faithfulness and Context Precision. RAGChecker is more about the general evaluation of the model, while RAGAs focus on detailed, component-level assessment. Both tools are valuable for different aspects of evaluating and improving RAG models.

Let's start coding

## Evaluating a RAG Application with RAGAs

to start, we need to install dependencies and set up our environment

Install all the necessary dependencies

```
%pip install -qU ragchecker llama-index
```

To use the RAGChecker, you prepare a JSON structure with a `results` array. Each item in the array corresponds to a query and includes details like the query itself, the expected answer, the RAG-generated response, and the documents retrieved by the model to inform that response.

```
{
 "results": [
   {
     "query_id": "<query id>",
     "query": "<input query>",
     "gt_answer": "<ground truth answer>",
     "response": "<response generated by the RAG generator>",
     "retrieved_context": [
       {
         "doc_id": "<doc id>",
         "text": "<content of the chunk>"
       }
     ]
   }
 ]
}
```

With the prerequisites out of the way, let's dive into LlamaIndex. First, we'll create an index using a document set and then query it. In this example, we assume we have a directory called `Your documents` containing our documents.

The `VectorStoreIndex.from_documents()` function takes our loaded documents and creates an index. We then create a query engine from this index using the `as_query_engine()` function.

```
from llama_index.core import VectorStoreIndex, SimpleDirectoryReader

# Load documents
documents = SimpleDirectoryReader("path/to/your/documents").load_data()

# Create index
index = VectorStoreIndex.from_documents(documents)

# Create query engine
rag_application = index.as_query_engine()
```

Now, we'll demonstrate how to use the **response_to_rag_results** function, which is essential for converting the raw output from retrieval-augmented generation models like LlamaIndex into a structured format that RAGChecker can use to perform detailed evaluations of the model's performance, ensuring the response accuracy and relevance of the retrieved documents.

```
from ragchecker.integrations.llama_index import response_to_rag_results
# User query and groud truth answer
user_query = "What is RAGChecker?"
gt_answer = "RAGChecker is an advanced automatic evaluation framework
designed
to assess and diagnose Retrieval-Augmented Generation (RAG) systems.
It provides a comprehensive suite of metrics and tools for in-depth
analysis
of RAG performance."


# Get response from LlamaIndex
response_object = rag_application.query(user_query)

# Convert to RAGChecker format
rag_result = response_to_rag_results(
   query=user_query,
   gt_answer=gt_answer,
   response_object=response_object,
)

# Create RAGResults object
rag_results = RAGResults.from_dict({"results": [rag_result]})
print(rag_results)
```

The RAGChecker is configured to use a specific model for extracting and checking responses, ensuring consistency across evaluations. The use of batch processing indicates that the system is likely designed to handle large datasets efficiently.

```
from ragchecker import RAGResults, RAGChecker
from ragchecker.metrics import all_metrics
# Initialize RAGChecker
evaluator = RAGChecker(
   extractor_name="bedrock/meta.llama3-70b-instruct-v1:0",
   checker_name="bedrock/meta.llama3-70b-instruct-v1:0",
   batch_size_extractor=32,
   batch_size_checker=32,
)

# Evaluate using RAGChecker
evaluator.evaluate(rag_results, all_metrics)

# Print detailed results
print(rag_results)
```

This output provides a comprehensive view of the RAG system's performance, detailing overall, retriever, and generator metrics.


```
RAGResults(
 1 RAG results,
 Metrics:
 {
   "overall_metrics": {
     "precision": 66.7,
     "recall": 27.3,
     "f1": 38.7
   },
   "retriever_metrics": {
     "claim_recall": 54.5,
     "context_precision": 100.0
   },
   "generator_metrics": {
     "context_utilization": 16.7,
     "noise_sensitivity_in_relevant": 0.0,
     "noise_sensitivity_in_irrelevant": 0.0,
     "hallucination": 33.3,
     "self_knowledge": 0.0,
     "faithfulness": 66.7
   }
 }
)
```


## Analysis of the experimental results in this paper

In this study, we evaluated 4,162 queries across 10 domains using eight RAG systems that combine two retrievers, BM25 and E5-Mistral, and four generators, GPT-4, Mixtral-8x7B, Llama3-8B, and Llama3-70B.

**1. The impact of retrievers**

E5-Mistral showed better retriever performance than BM25, with claim recall reaching 83.5% for E5-Mistral compared to 67.2% for BM25. This significant improvement indicates that E5-Mistral can accurately retrieve more relevant information, and the F1 score also improved to 53.0% for the E5-Mistral system compared to an average of 42.7% for the BM25 system.
This suggests that using a more powerful retriever (e.g., E5-Mistral) is an effective means of increasing claim recall and improving F1 scores.

**2. Generator Influence**

The size of the generator had a direct effect on performance. In particular, Llama3-70B had the highest context utilization rate of 57.6%, surpassing Llama3-8B's 54.9%. Furthermore, the hallucination rate of Llama3-70B dropped to 6.8%, an improvement over Llama3-8B's 8.3%. It was confirmed that a large generator contributes to accurate context utilization and reduced hallucination.

**3. The Importance of Context**

We observed that context utilization is strongly correlated with the overall F1 score, with a 0.9-point improvement in the F1 score for every 1% improvement in context utilization. This shows the impact of the quality of context information obtained from the retriever on the overall system performance. This means that providing more context information improves the fidelity of the generator and reduces hallucination. Increasing k and chunk size is expected to improve recall and F1 score but may increase noise sensitivity.

**4. Reliability and reduced hallucination**

By using the E5-Mistral, the faithfulness increased from 88.1% in the BM25 system to 93.7% in the E5-Mistral system, and the hallucination rate also decreased by 1.5% in the E5-Mistral system. This confirmed that the accuracy of the retriever improves the reliability of the generator.

### 5. Trade-off in noise sensitivity

As the retriever's claim recall improved, the generator's noise sensitivity also increased, from 28.5% for the BM25 system to 31.7% for the E5-Mistral system. As claim recall improves, so does noise sensitivity. The generator is more sensitive to noise because the retriever retrieves relevant information together with the noise. Proper context processing is needed to mitigate noise sensitivity.

### 6. Challenges of the Open Source Model

GPT-4 performed better than other open-source models, with a context utilization rate of 62.3% and a noise sensitivity of 27.1%. However, open-source models tend to blindly trust the context and need help to identify noise, which remains a challenge.

## Conclusion :

RAGChecker represents a significant advancement in the evaluation of RAG models. It provides essential tools and metrics for researchers and developers to enhance their systems and contribute to the ongoing evolution of NLP technologies.

**Reference :**


- https://docs.llamaindex.ai/en/latest/examples/evaluation/RAGChecker/
- https://github.com/amazon-science/RAGChecker
- https://arxiv.org/abs/2408.08067