Analytics Vidhya

Hands-On: End-to-End RAG System Evaluation

Instructor

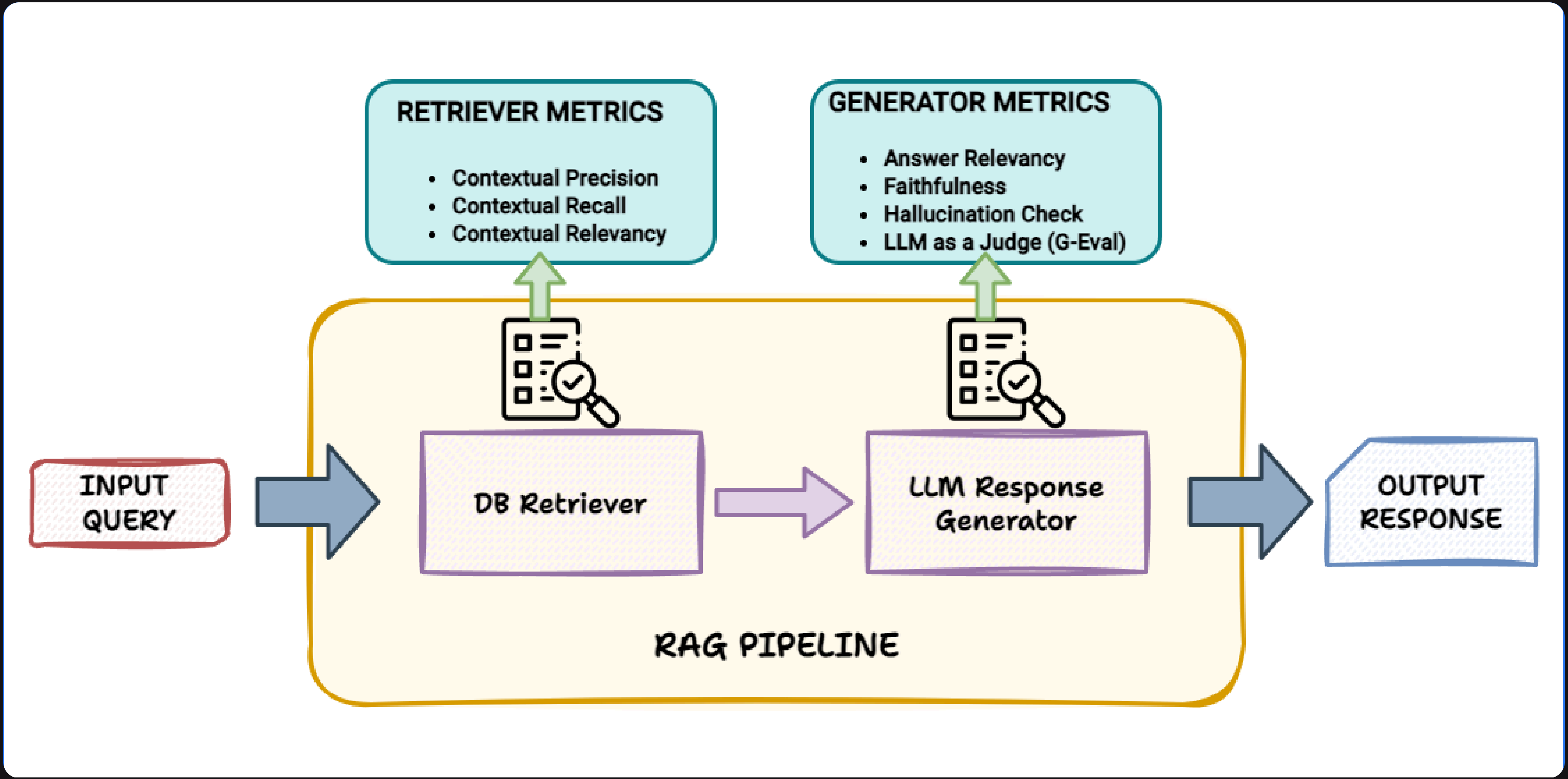
Dipanjan Sarkar

Head of Community & Principal AI Scientist at Analytics Vidhya

Google Developer Expert - ML & Cloud Champion Innovator

Published Author

RAG Evaluation Point & Metrics



Major Points in a RAG System Evaluation

Retriever: Here, we measure retrieval performance from the vector DB for input queries.

- **Contextual Precision**

  Relevant retrieved context to input query should rank higher

- **Contextual Recall**

  Retrieved context should align with the expected ground truth response

- **Contextual Relevancy**

  Relevancy of statements in retrieved context to the input query should be more in count

Major Points in a RAG System Evaluation

Generator: This is where we measure the quality of generated responses from the LLM for input queries and retrieved context

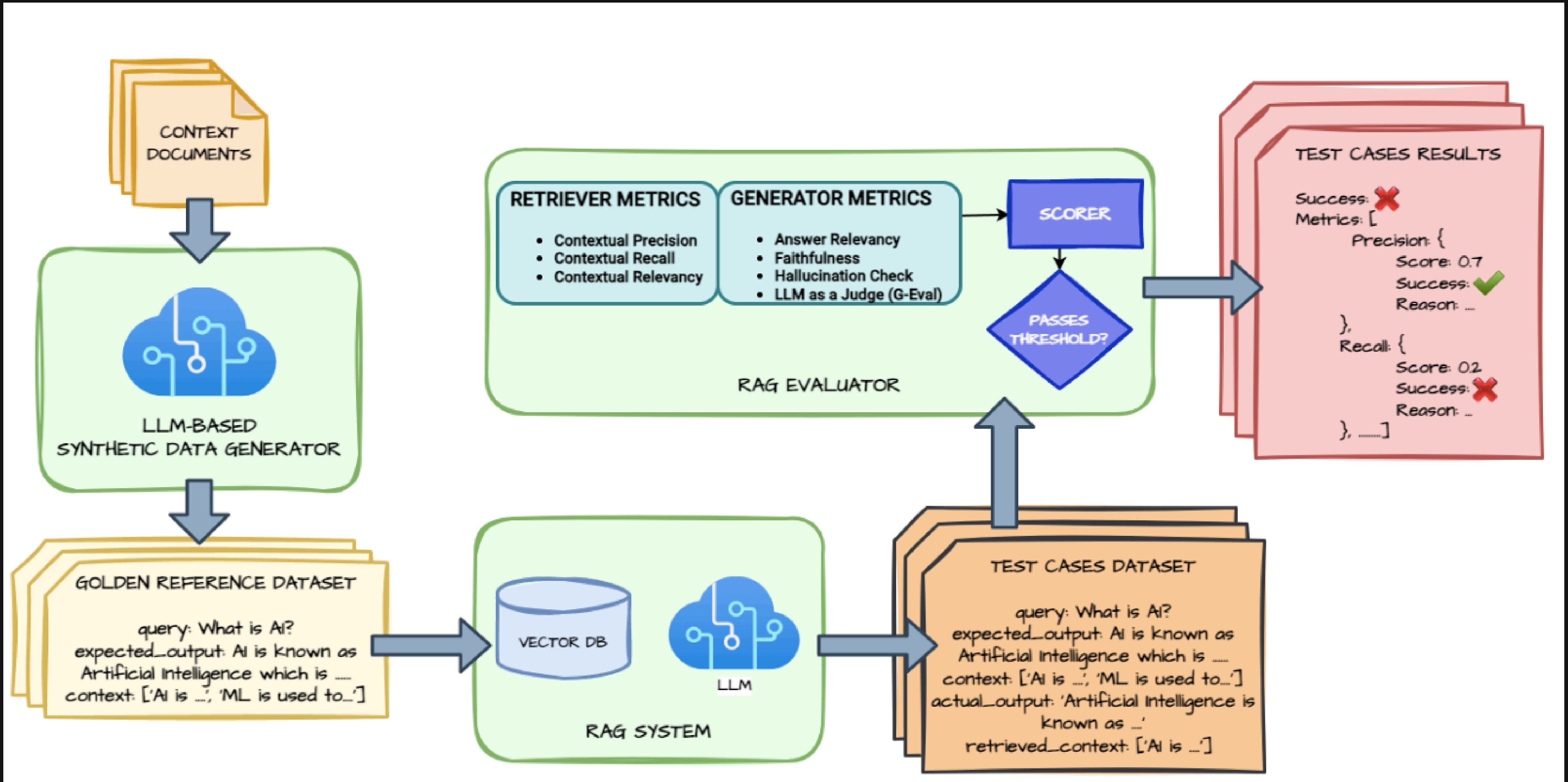
- ## Answer Relevancy
  Relevancy of statements in generated response to the input query should be more in count or semantically similar (LLM-based or semantic similarity)

- ## Hallucination Check
  Number of statements in generated response which contradict the ground truth context should be minimal

- ## Faithfulness
  Count of truthful claims made in the generated responses w.r.t the retrieved context should be more

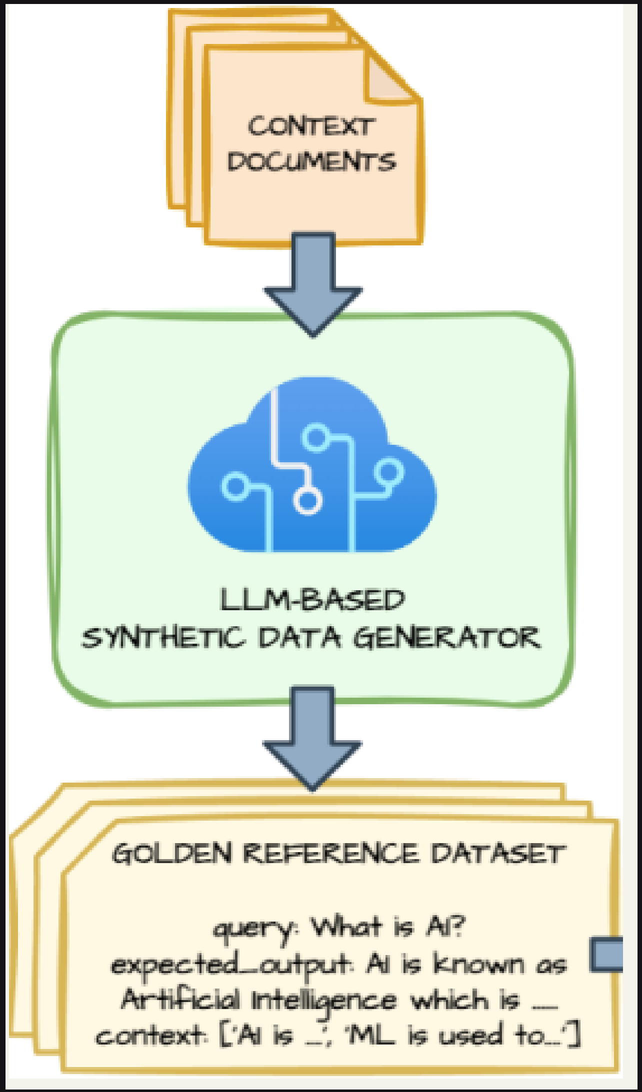
- ## Custom LLM as a Judge
  You can create your own judging metrics based on custom evaluation criteria as needed.

End-to-End RAG System Evaluation Pipeline



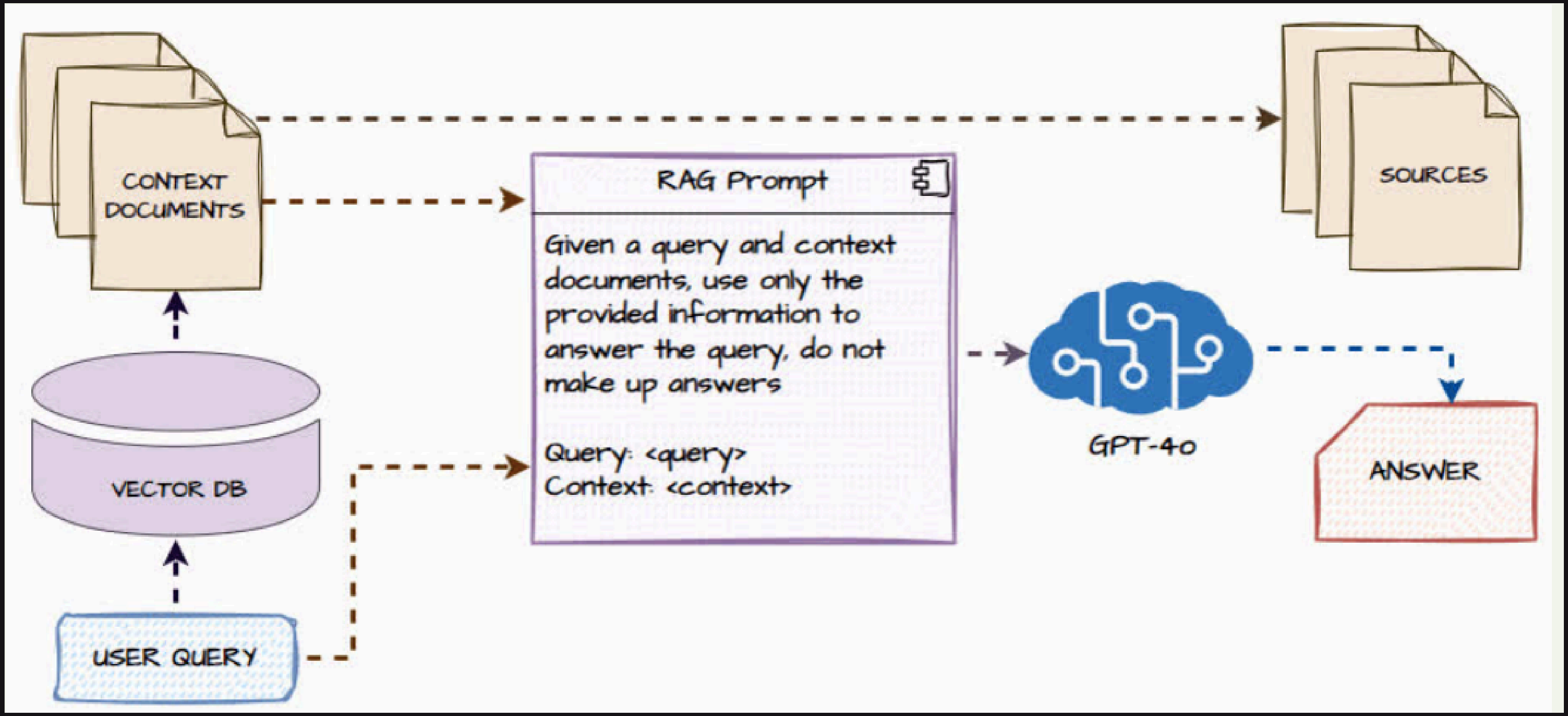
Analytics Vidhya

LLM-based Synthetic Golden Reference Data Generator



- Create Golden Reference Data samples manually or using an LLM synthetically
- Golden reference data samples would consist of the following:
  - Input Query: Input question to the RAG system
  - Expected Output: Ground truth answer to be expected from the LLM Generator
  - Context: Expected ground truth context which should be retrieved

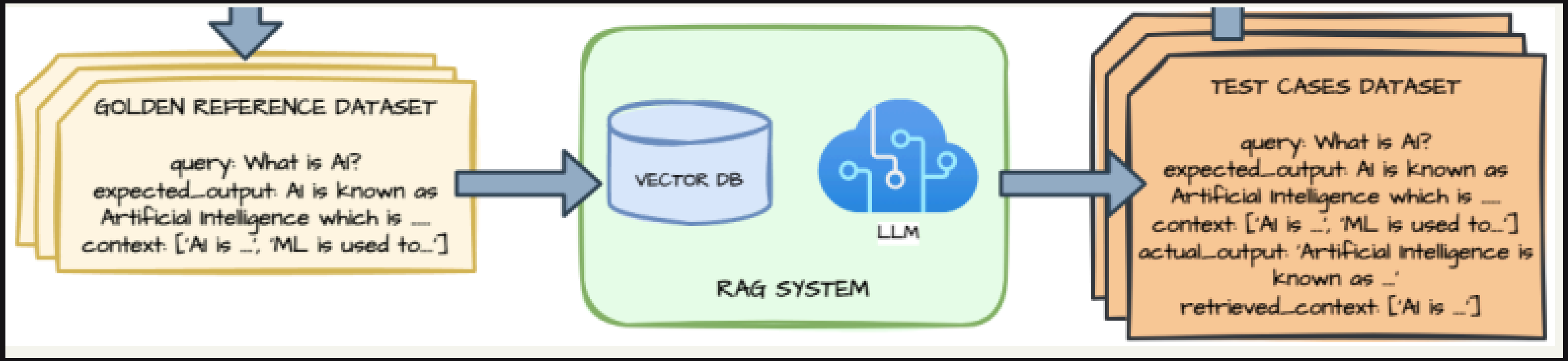
RAG System with Sources



- Build a RAG system as usual which can return the generated responses to any input query
- Besides the response also return the retrieved source context
- This helps in evaluating retriever and generator metrics in one go
- Avoids having to run separate queries on vector DB for evaluating retriever metrics and RAG system for generator metrics for each reference data samples.

Analytics Vidhya

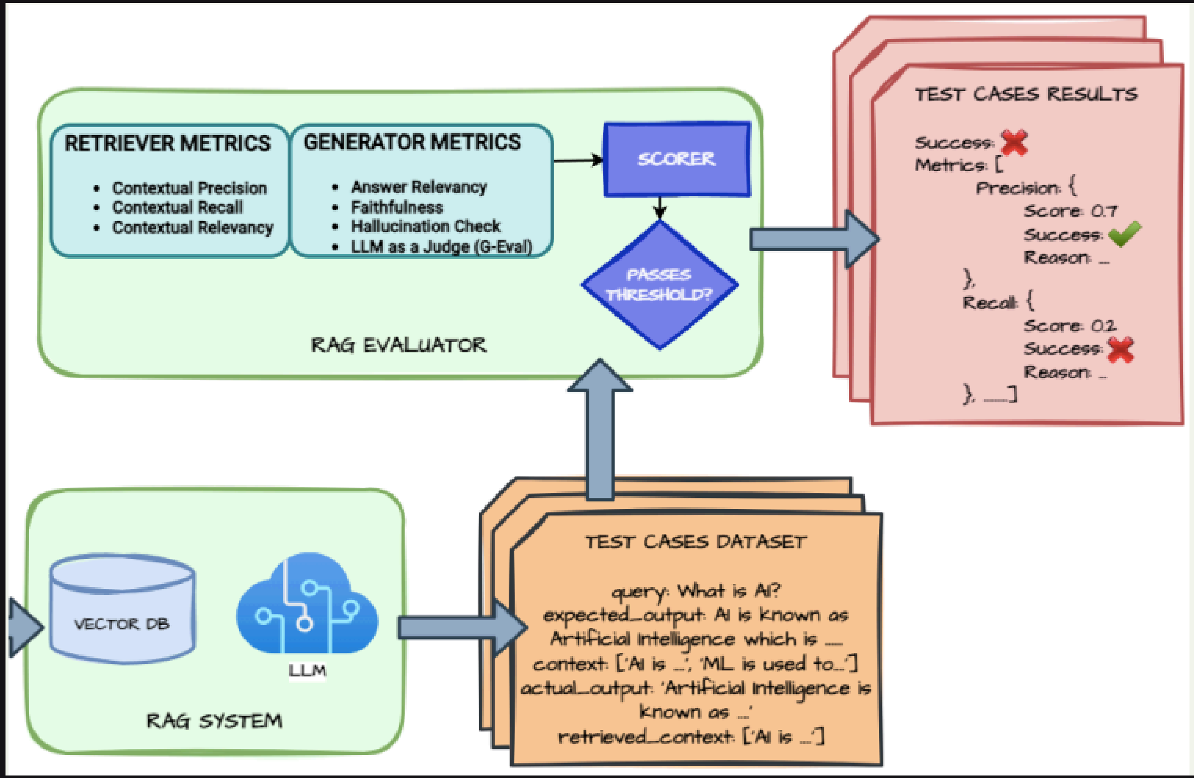
Create Evaluation Test Cases



- Here we take the input query of each golden reference data sample
- Pass the query to the RAG system and take the Retrieved Context and LLM Response as output
- Append them to each golden reference data samples to create a test case
- Each Test Case Sample will consist of the following:
  - Input Query: Input question to the RAG system
  - Expected Output: Ground truth answer to be expected from the LLM generator
  - Context: Expected ground truth context which should be retrieved
  - Actual Output: The actual response from the RAG system's LLM Generator
  - Retrieved Context: The actual retrieved context from the RAG System's Vector DB Retriever.

Analytics Vidhya

Run RAG Evaluation on Test Cases



- Define the RAG Metrics you want to evaluate each test case on in terms of:
  - Metric definition
  - Pass or fail threshold
  - Specific evaluation instructions in case of custom metrics
- Evaluate each test case and store the metrics
- Visualize on your dashboard as needed and improve system over time

RAG Evaluation Example with DeepEval

```
from deepeval import evaluate
from deepeval.metrics import ContextualPrecisionMetric, ContextualRecallMetric, ContextualRelevancyMetric
from deepeval.metrics import AnswerRelevancyMetric, FaithfulnessMetric, HallucinationMetric
from deepeval.metrics.ragas import RAGASAnswerRelevancyMetric

eval_dataset.test_cases = [....] # create your test cases
contextual_precision = ContextualPrecisionMetric(threshold=0.5, include_reason=True, model="gpt-4o")
contextual_recall = ContextualRecallMetric(threshold=0.5, include_reason=True, model="gpt-4o")
contextual_relevancy = ContextualRelevancyMetric(threshold=0.5, include_reason=True, model="gpt-4o")
answer_relevancy = AnswerRelevancyMetric(threshold=0.5, include_reason=True, model="gpt-4o")
faithfulness = FaithfulnessMetric(threshold=0.5, include_reason=True, model="gpt-4o")
hallucination = HallucinationMetric(threshold=0.5, include_reason=True, model="gpt-4o")
ragas_answer_relevancy = RAGASAnswerRelevancyMetric(threshold=0.5, embeddings=OpenAIEmbeddings(),
                                                    model="gpt-4o")

eval_results = evaluate(test_cases=eval_dataset.test_cases,
                        metrics=[contextual_precision, contextual_recall, contextual_relevancy,
                                 answer_relevancy, ragas_answer_relevancy, faithfulness, hallucination])


## EVAL OUTPUT ##

Evaluating 10 test case(s) in parallel: |██████████|100% (10/10) [Time Taken: 00:39,  3.98s/test case]
======================================================================

Metrics Summary

  - ✅ Contextual Precision (score: 1.0, threshold: 0.5, strict: False, ....)
  - ❌ Contextual Recall (score: 0.25, threshold: 0.5, strict: False, ....)
  - ❌ Contextual Relevancy (score: 0.3333333333333333, threshold: 0.5, strict: False, ....)
  - ✅ Answer Relevancy (score: 1.0, threshold: 0.5, strict: False, ....)
  - ❌ Answer Relevancy (ragas) (score: 0.0, threshold: 0.5, strict: False, ....)
  - ✅ Faithfulness (score: 1.0, threshold: 0.5, strict: False, ....)
  - ❌ Hallucination (score: 1.0, threshold: 0.5, strict: False, ....)
```

You can leverage libraries like DeepEval and Ragas to create your own custom eval metrics

Thank You