**Module 4 Introduction - Retrieval Augmented Generation Evaluation and Observability**

The module "Retrieval Augmented Generation Evaluation and Observability" provides a comprehensive exploration of advanced techniques and tools in the field of AI, focusing on enhancing chatbots and question-answering systems through Retrieval-Augmented Generation (RAG) systems. It explores the critical aspects of evaluating these systems, emphasizing the importance of faithfulness, relevance, and the prevention of hallucinations in AI responses. The module introduces tools like the FaithfulnessEvaluator and RAGAS, along with the Golden Context Dataset, offering insights into effective evaluation methodologies, including indexing, embedding, and generation metrics. Additionally, the module covers the LangChain framework and the LangSmith platform, providing practical knowledge on building and testing LLM-powered applications. Students will learn about the components of LangChain, such as Models, Vector Stores, and Chains, and the functionalities of the LangChain Hub.

**vRAG - Metrics & Evaluation**

In this lesson, you will learn about Retrieval-Augmented Generation (RAG) systems and their evaluation metrics, with a focus on improving chatbots and question-answering systems. The lesson introduces you to different approaches to analysis, the importance of faithfulness and answer relevancy, nuances of indexing and embedding metrics, and generation metrics aimed at preventing hallucinations in AI responses. It discusses the significance of the FaithfulnessEvaluator tool for checking the alignment of AI responses with retrieved context and introduces RAGAS and the Golden Context Dataset for system evaluation. Additionally, real-world setups for assessing and improving RAG systems are explored through examples of community-based tools, including comprehensive evaluation of retrieval metrics, holistic approach evaluations, and the custom RAG pipeline evaluation.

**LangSmith and LangChain Fundamentals for LLM Applications**

In this lesson, you will learn about the fundamentals of the LangChain framework and the newly introduced LangSmith platform to build and test LLM-powered applications. We will review LangChain components, such as Models, Vector Stores, and Chains, as well as principles of the LangChain Hub, including prompt exploration and versioning for collaborative prompt development. The lesson will guide you through setting up the LangSmith environment, creating an API key, and basics of prompt versioning and tracing. You will also learn how to use LangServe to deploy applications as a REST API and go through the process of reading and processing data from a webpage, storing it into Deep Lake vector store, and using prompts from the LangChain Hub to build a QuestionAnswering Chain application.

# RAG - Metrics & Evaluation

# ****Introduction****

So far, we've observed that LLMs' response generation ability is improved by incorporating context from a vector database, a typical design approach used in RAG systems for chatbots and question-answering systems. RAG applications strive to produce factually grounded outputs supported by the context they retrieve. Creating an evaluation pipeline for these systems is crucial, as it allows for measuring the effectiveness of the added techniques. The evaluation process should focus on ensuring that the output meaningfully incorporates the context, avoiding mere repetition, and aiming to create responses that are comprehensive, non-repetitive, and devoid of redundancy.

# ****RAG Metrics****

A holistic approach in RAG system evaluation is to present a detailed assessment of individual components and the system as a whole. Setting the baseline values for elements like chunking logic and embedding models and then examining each part independently and in an end-to-end manner is key for understanding the impact of modifications on the system’s overall performance. The holistic modules don't always need ground-truth labels, as they can be evaluated based on the query, context, response, and LLM interpretations.

Here are five metrics commonly used to evaluate RAG systems:

* [**Correctness**](https://docs.llamaindex.ai/en/latest/examples/evaluation/correctness_eval.html): Checks if the answer generated matches the reference answer for the given query (labels required). The accuracy of the generated answer is verified by comparing it directly to a reference answer provided for the question.
* [**Faithfulness**](https://docs.llamaindex.ai/en/latest/examples/evaluation/faithfulness_eval.html): Determines if the answer is accurate and doesn't contain fabrication, relative to the retrieved contexts. The faithfulness metric evaluates the integrity of the answer, ensuring it faithfully represents the information in the retrieved context by checking that the answer is accurate and free from distortions or fabrications that could misrepresent the source material.
* [**Context Relevancy**](https://docs.llamaindex.ai/en/latest/examples/evaluation/relevancy_eval.html): Measures the relevance of the retrieved context and the resulting answer to the original query. It does so by ensuring that the system only retrieves information in a way that is pertinent to the user's request.
* [**Guideline Adherence**](https://docs.llamaindex.ai/en/latest/examples/evaluation/guideline_eval.html): Determines if the predicted answer follows a set of guidelines and whether the response meets predefined criteria, encompassing stylistic, factual, and ethical standards, so the answer responds to the query while also aligning with specific established norms.
* [**Embedding Semantic Similarity**](https://docs.llamaindex.ai/en/latest/examples/evaluation/semantic_similarity_eval.html)**:** Calculates the similarity score between embeddings of the generated answer and the reference answer (reference labels required).

The analysis starts broadly, focusing on the overarching goal of RAG applications to produce helpful outputs supported by contextually relevant facts from the retrievers. It is then narrowed down to specific evaluation metrics, including faithfulness, answer relevancy, and the [Sensibleness and Specificity Average (SSA)](https://arxiv.org/abs/2001.09977), focusing on avoiding hallucination in responses. Google's SSA metric evaluates open-domain chatbot responses for sensibleness (contextual coherence) and specificity (detailed and direct responses). Initially involving human evaluators, this approach aims to ensure outputs are comprehensive yet not overly vague.

A high faithfulness score does not guarantee high relevance. For example, an answer that accurately reflects the context but lacks direct relevance to the question would score lower in answer relevance, especially if it includes incomplete or redundant information.

### Faithfulness Evaluator

Avoiding vague responses is essential, but preventing LLMs from “hallucinating” is equally crucial. Hallucination refers to generating responses not grounded in factual content or context. LlamaIndex's FaithfulnessEvaluator assesses responses based on their alignment with the retrieved context, measuring this aspect. Faithfulness evaluation considers whether the response matches the retrieved context, aligns with the query, and adheres to the reference answer or guidelines. The result returns a boolean value indicating whether the response passed the accuracy and faithfulness checks.

To execute the following codes, we must initially install the necessary libraries. This can be done using the Python package manager. Afterward, set the API keys for both OpenAI and the Activeloop service, as we will access indexes from a dataset hosted on Deep Lake. Remember to replace the placeholders with your API keys.

The code for this lesson is also available through a [Colab notebook](https://colab.research.google.com/drive/1XcCcoUi5ZbUNmTCLYXL51zeD-6mXON5T?usp=sharing), where you can follow along.

Copy

pip install -q llama-index==0.9.14.post3 deeplake==3.8.12 openai==1.3.8 cohere==4.37

Shell command to install the required libraries.

Copy

import os

os.environ["OPENAI\_API\_KEY"] = "<YOUR\_OPENAI\_KEY>"

os.environ["ACTIVELOOP\_TOKEN"] = "<YOUR\_ACTIVELOOP\_TOKEN>"

Code to set the environment variables we need.

Here's an example illustrating how to evaluate a single response for faithfulness.

Copy

from llama\_index import ServiceContext

from llama\_index.llms import OpenAI

# build service context

llm = OpenAI(model="gpt-4", temperature=0.0)

service\_context = ServiceContext.from\_defaults(llm=llm)

from llama\_index.vector\_stores import DeepLakeVectorStore

from llama\_index.storage.storage\_context import StorageContext

from llama\_index import VectorStoreIndex

vector\_store = DeepLakeVectorStore(dataset\_path="hub://genai360/LlamaIndex\_paulgraham\_essay", overwrite=False)

storage\_context = StorageContext.from\_defaults(vector\_store=vector\_store)

index = VectorStoreIndex.from\_vector\_store(

vector\_store, storage\_context=storage\_context

)

from llama\_index.evaluation import FaithfulnessEvaluator

# define evaluator

evaluator = FaithfulnessEvaluator(service\_context=service\_context)

# query index

query\_engine = index.as\_query\_engine()

response = query\_engine.query(

"What does Paul Graham do?"

)

eval\_result = evaluator.evaluate\_response(response=response)

print( "> response:", response )

print( "> evaluator result:", eval\_result.passing )

The sample code.

Copy

> response: Paul Graham is involved in various activities. He is a writer and has given talks on topics such as starting a startup. He has also worked on software development, including creating software for generating websites and building online stores. Additionally, he has been a studio assistant for a beloved teacher who is a painter.

> evaluator result: True

The output.

Most of the previously mentioned code should be recognizable, as it involves generating an index from the Deep Lake vector store, using this index to query the LLM, and conducting the evaluation procedure. The query engine processes the question, and its response is then forwarded to the evaluator for analysis. Let’s focus on the evaluation process, which starts by setting up an evaluator to assess the accuracy of responses based on the service context.

* The code initializes a FaithfulnessEvaluator, a tool designed to assess the accuracy of responses generated by the language model (GPT-4 in this case).
* The evaluator uses the service\_context created earlier, which includes the configured GPT-4 model. This context provides the necessary environment and parameters for the language model to function.
* The primary role of the FaithfulnessEvaluator is to determine how closely the language model's responses adhere to accurate and reliable information. It uses a set of criteria or algorithms to compare the generated responses against known factual data or expected outputs.

The evaluator then checks the response for its faithfulness to factual information. This means it evaluates whether the response accurately and reliably reflects historical facts about the queried topic. The result of this evaluation (eval\_result) is then checked to see if it meets the standards of accuracy set by the evaluator, indicated by eval\_result.passing. The result returns a boolean value indicating whether the response passed the accuracy and faithfulness checks.

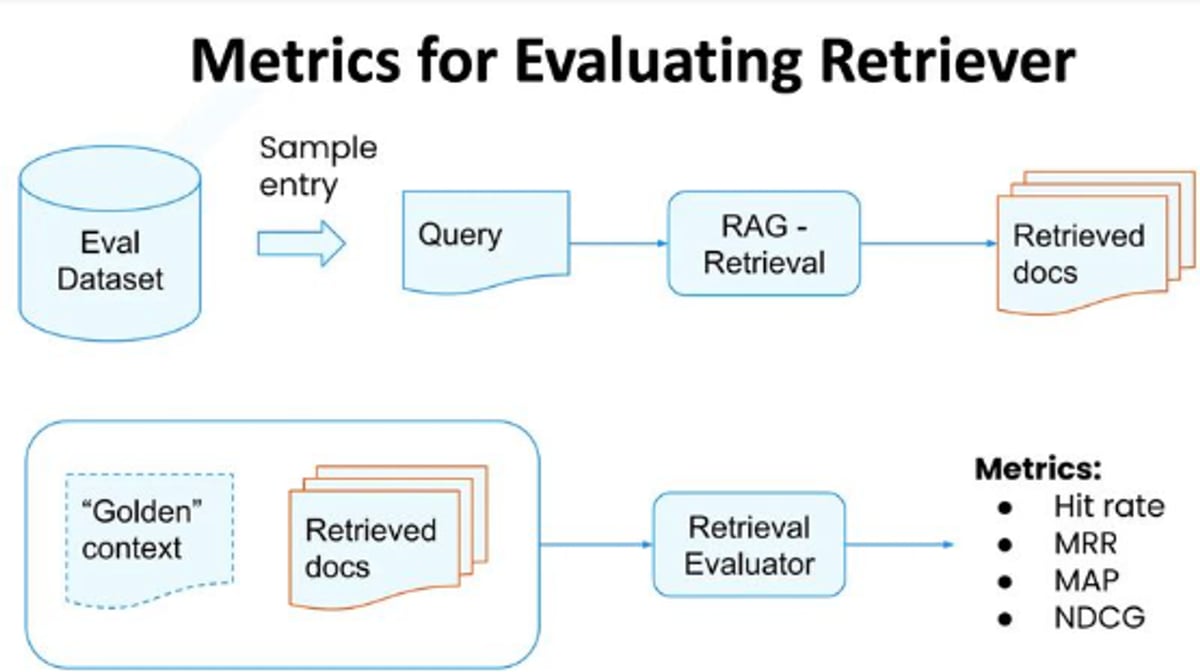
# ****Retrieval Evaluation Metrics****

The evaluation of retrieval in RAG systems involves determining the relevance of documents to specific queries.

✴️

RAG system outputs need to encompass three critical aspects: factual accuracy, direct relevance to the query, and inclusion of essential contextual information.

In information retrieval, the main goal is to identify unstructured data that meets a specific information requirement within a database.

the evaluation metrics of retrieval in RAG systems.

Metrics for evaluating a retriever include Mean Reciprocal Rank (MRR), Hit Rate, MAP and NDCG.

* [**MRR**](https://en.wikipedia.org/wiki/Mean_reciprocal_rank)measures the retrieval system's ability to return the best result as high up in the ranking as possible.
* **Hit Rate** evaluates the presence of relevant items within the top results returned, which is crucial where users only consider the first few results.
* [**MAP (Mean Average Precision)**](https://en.wikipedia.org/wiki/Evaluation_measures_(information_retrieval)#Mean_average_precision): is a measure of ranking quality across multiple queries. MAP calculates the mean of the average precisions for each query, where the average precision is computed as the mean of the precision scores after each relevant document is retrieved.
* [**NDCG (Normalized Discounted Cumulative Gain)**](https://en.wikipedia.org/wiki/Discounted_cumulative_gain): This metric evaluates the ranking of documents based on their relevance, giving more importance to relevant documents that appear higher in the ranking. It is normalized so that the perfect ranking's score is 1, allowing for comparison across different sets of queries.

The RetrieverEvaluator is a more advanced technique from LlamaIndex, which can calculate metrics such as Mean Reciprocal Rank (MRR) and Hit Rate. It is designed to evaluate the efficacy of a retrieval system, which retrieves data pertinent to user queries from a database or index. This class assesses the retriever's performance in relation to specific questions and expected results, providing benchmarks for evaluation.

Inferred from the evaluator's definition, it's necessary to compile an evaluation dataset comprising the contents, a collection of questions, and corresponding nodes that serve as references for answering these questions. The generate\_question\_context\_pairs function in LlamaIndex can take care of generating an evaluation dataset. The rest of the process involves passing a query and using the dataset as a reference to ensure the chatbot retrieves the right documents. Please read the [following tutorial](https://docs.llamaindex.ai/en/stable/examples/evaluation/retrieval/retriever_eval.html) for an example.

💡

While the single-query evaluation is presented, the practical implementation in real-world scenarios often requires batch evaluations. A large set of queries and expected results are extracted from the retriever to determine its overall reliability. In batch testing, the retriever undergoes an assessment involving many queries and their corresponding expected results. Systematically querying the retriever with these varied inputs and evaluating its outputs against predefined correct answers is the method employed to measure its consistency accurately.

# ****Golden Context Dataset****

The [Golden Context dataset](https://github.com/microsoft/promptflow-resource-hub/blob/main/sample_gallery/golden_dataset/copilot-golden-dataset-creation-guidance.md) would consist of carefully selected queries paired with an ideally matched set of sources that contain the answers. Optionally, it could also include the perfect answers that are expected to be generated by the LLM. For our purposes, 177 representative user queries have been manually curated. For each query, the most relevant source within our documentation has been diligently identified so that these sources directly address the queries in question. The Golden Context Dataset serves as our benchmark for precision evaluation. The dataset is structured around 'question' and 'source' pairings.

To create a Golden Dataset, gather a set of realistic customer questions and pair them with expert answers, then use this dataset to compare against responses from a language model for quality assurance, ensuring the LLM's answers align closely with the expert ones for accuracy and relevance. If you want to learn more about the dataset's creation, follow [the link in the resources](https://github.com/microsoft/promptflow-resource-hub/blob/main/sample_gallery/golden_dataset/copilot-golden-dataset-creation-guidance.md) section.

Once the golden dataset is ready, the next step is to use it to measure the quality of LLM responses. After each evaluation, metrics like the following will be available to quantify the user experience. For example:

|  |  |  |  |
| --- | --- | --- | --- |
| Similarity | Relevance | Coherence | Grounded-ness |
| 3.7 | 77 | 88 | 69 |

⚠️

Note: While synthetic question generation has benefits, it's not recommended for evaluating metrics related to real-world user experiences. Questions used in such evaluations should ideally mirror what actual users might ask, something not typically achieved with questions generated by an LLM. Hand-crafting questions with a focus on the users’ perspective is advised for a more accurate reflection.

# ****Community-Based Evaluation Tools****

LlamaIndex incorporates various evaluation tools designed to foster community engagement and collaborative endeavors. The devices are structured to support a shared process of assessing and enhancing the system, empowering users and developers to play an active role in the evaluation. Through the use of tools shaped by community input, LlamaIndex creates a collaborative environment where constant feedback is smoothly incorporated, contributing to continual development.

Notable tools in this ecosystem include:

* [Ragas](https://github.com/explodinggradients/ragas/blob/main/docs/howtos/integrations/llamaindex.ipynb): Another key tool that provides a framework for evaluating and integrating with LlamaIndex, offering detailed metrics.
* [DeepEval](https://docs.llamaindex.ai/en/latest/module_guides/evaluating/root.html#../../../community/integrations/deepeval.md): A tool designed for in-depth evaluation, facilitating comprehensive assessments of various aspects of the system.

## ****Evaluating with Ragas****

The evaluation process involves importing specific metrics from Ragas, such as faithfulness, answer relevancy, context precision, context recall, and harmfulness.

When evaluating using Ragas, the following elements are essential:

* **Query Engine**: This is the primary component and acts as the core of the evaluation process, where its performance is assessed.
* **Metrics**: Ragas provides a range of metrics specifically designed to evaluate a nuanced assessment of the engine's capabilities.
* **Questions**: A curated set of questions is required, which are used to probe the engine's ability to retrieve and generate accurate responses.

We must first set up a query engine to demonstrate how to use the Ragas library, which involves loading a document. For this, we will use the contents of the "New York City" Wikipedia page as our source document. Additionally, installing two more libraries is necessary: one for processing the webpage content and the other for the evaluation library itself.

Copy

pip install html2text==2020.1.16 ragas==0.0.22

The sample code.

At this point, we can use the SimpleWebPageReader class by providing a URL as its argument to load the content. These documents can then create the index and the query engine. Then, it is possible to ask questions about the document!

Copy

from llama\_index.readers.web import SimpleWebPageReader

from llama\_index import VectorStoreIndex, ServiceContext

documents = SimpleWebPageReader(html\_to\_text=True).load\_data( ["https://en.wikipedia.org/wiki/New\_York\_City"] )

vector\_index = VectorStoreIndex.from\_documents(

documents, service\_context=ServiceContext.from\_defaults(chunk\_size=512)

)

query\_engine = vector\_index.as\_query\_engine()

response\_vector = query\_engine.query("How did New York City get its name?")

print(response\_vector)

The sample code.

Copy

New York City got its name in honor of the Duke of York, who later became King James II of England. The Duke of York was appointed as the proprietor of the former territory of New Netherland, including the city of New Amsterdam, when England seized it from Dutch control.

The output.

Returning to our goal of evaluating the models, the next step involves composing a series of questions, ideally derived from the original document, to ensure a more accurate performance assessment.

Copy

eval\_questions = [

"What is the population of New York City as of 2020?",

"Which borough of New York City has the highest population?",

"What is the economic significance of New York City?",

"How did New York City get its name?",

"What is the significance of the Statue of Liberty in New York City?",

]

eval\_answers = [

"8,804,000", # incorrect answer

"Queens", # incorrect answer

"New York City's economic significance is vast, as it serves as the global financial capital, housing Wall Street and major financial institutions. Its diverse economy spans technology, media, healthcare, education, and more, making it resilient to economic fluctuations. NYC is a hub for international business, attracting global companies, and boasts a large, skilled labor force. Its real estate market, tourism, cultural industries, and educational institutions further fuel its economic prowess. The city's transportation network and global influence amplify its impact on the world stage, solidifying its status as a vital economic player and cultural epicenter.",

"New York City got its name when it came under British control in 1664. King Charles II of England granted the lands to his brother, the Duke of York, who named the city New York in his own honor.",

"The Statue of Liberty in New York City holds great significance as a symbol of the United States and its ideals of liberty and peace. It greeted millions of immigrants who arrived in the U.S. by ship in the late 19th and early 20th centuries, representing hope and freedom for those seeking a better life. It has since become an iconic landmark and a global symbol of cultural diversity and freedom.",

]

eval\_answers = [[a] for a in eval\_answers]

The sample code.

This stage is the setup phase of the evaluation process. QueryEngine's proficiency is assessed based on how effectively it processes and responds to these specific questions, utilizing the answers as a standard for measuring performance. We need to import the metrics from the Ragas library.

Copy

from ragas.metrics import (

faithfulness,

answer\_relevancy,

context\_precision,

context\_recall,

)

from ragas.metrics.critique import harmfulness

metrics = [

faithfulness,

answer\_relevancy,

context\_precision,

context\_recall,

harmfulness,

]

The metrics list compiles the metrics into a collection, which can then be used in the evaluation process to assess various aspects of the QueryEngine's performance. The results, which include scores for each metric, can be further analyzed.

Finally, let's run the evaluation:

Copy

from ragas.llama\_index import evaluate

result = evaluate(query\_engine, metrics, eval\_questions, eval\_answers)

# print the final scores

print(result)

The sample code.

Copy

evaluating with [faithfulness]

100%|██████████| 1/1 [00:16<00:00, 16.95s/it]

evaluating with [answer\_relevancy]

100%|██████████| 1/1 [00:03<00:00, 3.54s/it]

evaluating with [context\_precision]

100%|██████████| 1/1 [00:02<00:00, 2.73s/it]

evaluating with [context\_recall]

100%|██████████| 1/1 [00:07<00:00, 7.06s/it]

evaluating with [harmfulness]

100%|██████████| 1/1 [00:02<00:00, 2.16s/it]

{'faithfulness': 0.8000, 'answer\_relevancy': 0.7634, 'context\_precision': 0.6000, 'context\_recall': 0.8667, 'harmfulness': 0.0000}

The output.

The metrics analysis quantifies different aspects of the RAG system's performance:

1. faithfulness: **0.8000**
   * measures how accurately the system's responses adhere to the factual content of the source material. A score of 0.7 indicates relatively high faithfulness, meaning the responses are mostly accurate and true to the source.
2. answer\_relevancy: **0.7634**
   * measures how relevant the system's responses are to the given queries. A high score of 0.955 suggests that the majority of the system's responses are closely aligned with the queries' intent.
3. context\_precision: **0.6000**
   * evaluates the precision of the context used by the system to generate responses. A lower score of 0.2335 indicates that the context used often includes irrelevant information.
4. context\_recall: **0.8667**
   * measures the recall rate of relevant context determined by the system. A high score of 0.98 suggests that the system is very effective in retrieving most of the relevant context.
5. harmfulness: **0.0000**
   * measures the system for harmful or inappropriate content generation. A score of 0 implies that no harmful content was generated in the evaluated responses.

## ****The Custom RAG Pipeline Evaluation****

For a practical evaluation of a custom RAG system, it is essential to employ a range of assessment benchmarks instrumental in assessing various facets of the RAG system, such as its effectiveness and reliability. The variety of measures guarantees a detailed evaluation and in-depth insight into the system's overall capabilities. This section involves developing a tailored evaluation pipeline, beginning with loading a dataset, forming an evaluation dataset from its contents, and then calculating the metrics we have previously discussed. Initially, we download the text file to serve as the dataset.

Copy

wget 'https://raw.githubusercontent.com/idontcalculate/data-repo/main/venus\_transmission.txt'

The sample code.

Copy

--2023-12-18 20:21:53-- https://raw.githubusercontent.com/idontcalculate/data-repo/main/venus\_transmission.txt

Resolving raw.githubusercontent.com (raw.githubusercontent.com)... 185.199.108.133, 185.199.109.133, 185.199.110.133, ...

Connecting to raw.githubusercontent.com (raw.githubusercontent.com)|185.199.108.133|:443... connected.

HTTP request sent, awaiting response... 200 OK

Length: 19241 (19K) [text/plain]

Saving to: ‘venus\_transmission.txt’

venus\_transmission. 100%[===================>] 18.79K --.-KB/s in 0.001s

2023-12-18 20:21:53 (32.6 MB/s) - ‘venus\_transmission.txt’ saved [19241/19241]

The output.

Then, the text file can be loaded as a Document object, identifiable by LlamaIndex.

Copy

from llama\_index import SimpleDirectoryReader

reader = SimpleDirectoryReader(input\_files=["/content/venus\_transmission.txt"])

docs = reader.load\_data()

print(f"Loaded {len(docs)} docs")

The sample code.

Copy

Loaded 1 docs

The output.

The SimpleNodeParser, in this context, converts documents into a structured format known as nodes and serves for customization in parsing documents, specifically in terms of defining the chunk size, managing overlap, and incorporating metadata. Each chunk of the document is treated as a node. In this case, the parser is set with a chunk\_size of 512, meaning each node will consist of 512 characters from the original document. These chunks can then be used to generate the indexes.

Copy

from llama\_index.node\_parser import SimpleNodeParser

from llama\_index import VectorStoreIndex

# Build index with a chunk\_size of 512

node\_parser = SimpleNodeParser.from\_defaults(chunk\_size=512)

nodes = node\_parser.get\_nodes\_from\_documents(docs)

vector\_index = VectorStoreIndex(nodes)

The sample code.

The indexes can now be used as a query engine to query a specific question concerning the source document.

Copy

query\_engine = vector\_index.as\_query\_engine()

response\_vector = query\_engine.query("What was The first beings to inhabit the planet?")

print( response\_vector.response )

The sample code.

Copy

The first beings to inhabit the planet were a dinoid and reptoid race from two different systems outside our solar system.

The output.

The response generated by the query engine is stored in response\_vector. So, the document is processed into nodes, indexed, and then queried using a language model. To investigate the response further, we can use the .source\_nodes key to access the retrieved document from the index used to answer the question.

Copy

# First retrieved node

response\_vector.source\_nodes[0].get\_text()

The sample code.

Copy

They had heard of this beautiful new planet. At this time, Earth had two moons to harmonize the weather conditions and control the tides of the large bodies of water.

The first beings to inhabit the planet were a dinoid and reptoid race from two different systems outside our solar system. They were intelligent and walked on two legs like humans and were war-like considering themselves to be superior to all other life forms. In the past, the four races of humans had conflicts with them before they outgrew such behavior. They arrived on Earth to rob it of its minerals and valuable gems. Soon they had created a terrible war. They were joined by re-

1

enforcements from their home planets. One set up its base on one of the Earth's moons, the other on Earth. It was a terrible war with advanced nuclear and laser weapons like you see in your science fiction movies. It lasted very long. Most of the life forms lay in singed waste and the one moon was destroyed. No longer interested in Earth, they went back to their planets leaving their wounded behind, they had no use for them.

The four races sent a few forces to see if they could help the wounded dinoids and reptilians and to see what they could do to repair the Earth. They soon found that due to the nuclear radiation it was too dangerous on Earth before it was cleared. Even they had to remain so as not to contaminate their own planets.

Due to the radiation, the survivors of the dinoids and reptoids mutated into the Dinosaurs and giant reptilians you know of in your history. The humans that were trapped there mutated into what you call Neanderthals.

The Earth remained a devastated ruin, covered by a huge dark nuclear cloud and what vegetation was left was being devoured by the giant beings, also humans and animals by some. It was this way for hundreds of years before a giant comet crashed into one of the oceans and created another huge cloud. This created such darkness that the radiating heat of the Sun could not interact with Earth's gravitational field and an ice age was created. This destroyed the mutated life forms and gave the four races the chance to cleanse and heal the Earth with technology and their energy.

Once again, they brought various forms of life to the Earth, creating again a paradise, except for extreme weather conditions and extreme tidal activities.

The output.

We can index the second item on the list to view the content of the second node that contributed to the generation of the content.

Copy

# Second retrieved node

response\_vector.source\_nodes[1].get\_text()

The sample code.

Copy

Due to the radiation, the survivors of the dinoids and reptoids mutated into the Dinosaurs and giant reptilians you know of in your history. The humans that were trapped there mutated into what you call Neanderthals.

The Earth remained a devastated ruin, covered by a huge dark nuclear cloud and what vegetation was left was being devoured by the giant beings, also humans and animals by some. It was this way for hundreds of years before a giant comet crashed into one of the oceans and created another huge cloud. This created such darkness that the radiating heat of the Sun could not interact with Earth's gravitational field and an ice age was created. This destroyed the mutated life forms and gave the four races the chance to cleanse and heal the Earth with technology and their energy.

Once again, they brought various forms of life to the Earth, creating again a paradise, except for extreme weather conditions and extreme tidal activities.

During this time they realized that their planets were going into a natural dormant stage that they would not be able to support physical life. So they decided to colonize the Earth with their own people. They were concerned about the one moon, because it is creating earthquakes and tidal waves and storms and other difficulties for the structure of the Earth. They knew how to drink fluids to protect and balance themselves. These were the first colonies like Atlantis and Lemuria.

The rest of the people stayed on their planets to await their destiny. They knew that they would perish and die. They had made the decision only to bring the younger generation with some spiritual teachers and elders to the Earth. The planet was too small for all of them. But they had no fear of death.

They had once again created a paradise. They were instructed to build special temples here as doorways to the other dimensions. Because of the aggressive beings, the temples were hidden for future times when they will be important. There they could do their meditations and the higher beings.

They were informed to build two shields around the Earth out of ice particles to balance the influence of the one moon. They created a tropical climate for the Earth. There were no deserts at that time. They have special crystals for these doorways and they were able to lower their vibration to enter through these doorways. The news spread of the beautiful planet.

The output.

You can view the textual information from the second node that the query engine found relevant, providing additional context or information in response to the query. This helps understand the breadth of knowledge the query engine pulls from and how different parts of the indexed documents contribute to the overall response.

As previously mentioned, our task involves creating an evaluation dataset. This essentially entails generating a series of questions and their respective answers, all in relation to the content we have loaded. The generate\_question\_context\_pairs class leverages the LLM to create questions based on the content of each node: For each node, two questions will be created, resulting in a dataset where each item consists of a context (the node's text) and a corresponding set of questions. The Q&A dataset will serve us to evaluate the capabilities of an RAG system in question generation and context understanding tasks. You can see the first ten questions in the output.

Copy

from llama\_index.llms import OpenAI

from llama\_index.evaluation import generate\_question\_context\_pairs

# Define an LLM

llm = OpenAI(model="gpt-3.5-turbo")

qa\_dataset = generate\_question\_context\_pairs(

nodes,

llm=llm,

num\_questions\_per\_chunk=2

)

queries = list(qa\_dataset.queries.values())

print( queries[0:10] )

The sample code.

Copy

100%|██████████| 13/13 [00:31<00:00, 2.46s/it]

['Explain the role of different alien races in the history of our solar system according to the information provided. How did these races contribute to the transformation process and why was Earth considered a special planet?', 'Describe the advanced abilities and technology possessed by the Masters and beings mentioned in the context. How did their understanding of creation and their eternal nature shape their perspective on life and death?', 'How did the four races of humans demonstrate their mastery of creativity and what were the potential consequences of using this power for selfish reasons?', 'Describe the initial state of Earth before it became a planet and how the four races of humans contributed to its transformation into a unique paradise.', 'How did the arrival of the dinoid and reptoid races on Earth lead to a devastating war? Discuss the reasons behind their conflict with the four races of humans and the impact it had on the planet.', "Explain the process of mutation that occurred among the survivors of the dinoids and reptoids, resulting in the emergence of dinosaurs and Neanderthals. Discuss the role of nuclear radiation and its effects on the Earth's environment and living organisms.", 'How did the survivors of the dinoids and reptoids mutate into the dinosaurs and giant reptilians we know of in history? Explain the role of radiation in this process.', 'Describe the events that led to the creation of an ice age on Earth. How did this ice age affect the mutated life forms and provide an opportunity for the four races to cleanse and heal the Earth?', 'Explain the purpose and significance of building special temples as doorways to other dimensions in the context of the given information. How did these temples serve the people and protect them from the dark forces?', 'Discuss the actions taken by the colonies in response to the war declared by another race of humans. How did the colonies ensure the preservation of their knowledge and technology, and what measures did they take to protect themselves from the dark forces?', 'How did the inhabitants of Lemuria and Atlantis ensure that their knowledge and technology would not be misused by the dark forces?', 'What measures were taken by the controlling forces to prevent the people from communicating with other dimensions and remembering their past lives or the hidden temples?', 'How has the manipulation and control of human beings by the rich and powerful impacted society throughout history? Discuss the role of religion, race, and power in perpetuating this control and the potential consequences for humanity.', 'Explain the role of the Galactic Brotherhood and other spiritually evolved beings in the transformation of Earth. How have they worked to change the energy of the planet and its inhabitants? Discuss the potential risks they aim to mitigate, such as genetic manipulation and the use of destructive technologies.', "Explain the role of the Galactic Brotherhood in the transformation of the planet's energy and the introduction of new technologies. How are different beings, such as the Spiritual Hierarchy, Ascended Masters, and nature spirits, cooperating in this process?", 'Discuss the significance of the hidden temples and the space ships in the frequency change of the Earth. How do these elements contribute to the gradual transformation and what effects do they have on the environment?', 'Explain the concept of chakras and their role in the transformation process described in the context information. How do chakras relate to the abilities of mental telepathy, intuition, and past life recollection?', "Discuss the significance of the Earth's future purpose as mentioned in the context information. How does it differ from its past role? How does the concept of yin and yang, as well as the negative and positive energies, tie into this transformation?", 'How does the concept of division into good and bad energies contribute to the perpetuation of negative forces and selfishness among individuals?', 'Discuss the shift in power dynamics from feminine qualities to male energy in societies after genetic manipulation. How does the future vision of equal and balanced male and female powers impact the purpose of Earth for human beings?', 'How has the balance of feminine and masculine energies shifted throughout human history, and what is the envisioned future for this balance on Earth?', 'In the future described in the context information, how will individuals govern themselves and what role will manmade laws play in society?', 'How does the concept of obeying spiritual laws contribute to living in harmony on other planets for millions of years? Provide examples or evidence from the context information to support your answer.', 'According to the context information, what are some key aspects of the future living style and awareness on Earth after the transformation is complete? How do these aspects differ from the current state of existence?', "How does the concept of eternity and the ability to overcome time and aging impact one's perspective on life and the enjoyment of experiences?", 'In what ways can individuals create a balance and harmony within themselves, and why is it important for them to do so?']

The output.

The RetrieverEvaluator class can now use this QA dataset to evaluate the retriever's performance. It queries each question using the retriever and evaluates which chunks are returned as the answer. The higher MRR and Hit rate numbers represent the retriever’s ability to identify the chunk with the correct answer.

Copy

from llama\_index.evaluation import RetrieverEvaluator

retriever = vector\_index.as\_retriever(similarity\_top\_k=2)

retriever\_evaluator = RetrieverEvaluator.from\_metric\_names(

["mrr", "hit\_rate"], retriever=retriever

)

# Evaluate

eval\_results = await retriever\_evaluator.aevaluate\_dataset(qa\_dataset)

def display\_results(name, eval\_results):

"""Display results from evaluate."""

metric\_dicts = []

for eval\_result in eval\_results:

metric\_dict = eval\_result.metric\_vals\_dict

metric\_dicts.append(metric\_dict)

full\_df = pd.DataFrame(metric\_dicts)

hit\_rate = full\_df["hit\_rate"].mean()

mrr = full\_df["mrr"].mean()

metric\_df = pd.DataFrame(

{"Retriever Name": [name], "Hit Rate": [hit\_rate], "MRR": [mrr]}

)

return metric\_df

display\_results("OpenAI Embedding Retriever", eval\_results)

The sample code.

Copy

| | Retriever Name | Hit Rate | MRR |

| ---- | -------------------------- | -------- | --------- |

| 0 | OpenAI Embedding Retriever | 0.884615 | 0.730769 |

The output.

We can now enhance our analysis of the application's performance by including additional metrics like faithfulness and relevancy. To achieve this, we utilize a subset of the generated Q&A dataset and create instances of both GPT-3.5 and GPT-4. It's advisable to employ a more advanced model like GPT-4 for evaluation purposes while using the less expensive model for the generation process.

Copy

# gpt-3.5-turbo

gpt35 = OpenAI(temperature=0, model="gpt-3.5-turbo")

service\_context\_gpt35 = ServiceContext.from\_defaults(llm=gpt35)

# gpt-4

gpt4 = OpenAI(temperature=0, model="gpt-4")

service\_context\_gpt4 = ServiceContext.from\_defaults(llm=gpt4)

vector\_index = VectorStoreIndex(nodes, service\_context = service\_context\_gpt35)

query\_engine = vector\_index.as\_query\_engine()

eval\_query = queries[10]

response\_vector = query\_engine.query(eval\_query)

print( "> eval\_query: ", eval\_query )

print( "> response\_vector:", response\_vector )

The sample code.

Copy

> eval\_query: How did the colonies respond to the declaration of war by the dark forces, and what measures did they take to protect their knowledge and technology?

> response\_vector: The colonies did not fight back against the dark forces when they declared war. Instead, they sent most of their people into hiding in order to rebuild the colonies later. They also destroyed everything to ensure that their knowledge and technology would not fall into the hands of the dark forces. Additionally, Lemuria and Atlantis were destroyed by their inhabitants to prevent the misuse of their knowledge and technology by the dark forces.

The output.

Now, we can establish the evaluator classes responsible for measuring each metric. We'll then use a sample response to determine if it meets the test criteria.

Copy

from llama\_index.evaluation import RelevancyEvaluator

from llama\_index.evaluation import FaithfulnessEvaluator

relevancy\_gpt4 = RelevancyEvaluator(service\_context=service\_context\_gpt4)

faithfulness\_gpt4 = FaithfulnessEvaluator(service\_context=service\_context\_gpt4)

# Compute faithfulness evaluation

eval\_result = faithfulness\_gpt4.evaluate\_response(response=response\_vector)

# check passing parameter in eval\_result if it passed the evaluation.

print( eval\_result.passing )

# Relevancy evaluation

eval\_result = relevancy\_gpt4.evaluate\_response(

query=eval\_query, response=response\_vector

)

# You can check passing parameter in eval\_result if it passed the evaluation.

print( eval\_result.passing )

The sample code.

Copy

True

True

The output.

We must perform a for-loop to feed each sample from the evaluation dataset and get the appropriate results. In this situation, we can use the LlamaIndex BatchEvalRunner class, which runs the evaluation process in batches and concurrently. It means the evaluation can be done faster.

Copy

#Batch Evaluator:

#BatchEvalRunner to compute multiple evaluations in batch wise manner.

from llama\_index.evaluation import BatchEvalRunner

# Let's pick top 10 queries to do evaluation

batch\_eval\_queries = queries[:10]

# Initiate BatchEvalRunner to compute FaithFulness and Relevancy Evaluation.

runner = BatchEvalRunner(

{"faithfulness": faithfulness\_gpt4, "relevancy": relevancy\_gpt4},

workers=8,

)

# Compute evaluation

eval\_results = await runner.aevaluate\_queries(

query\_engine, queries=batch\_eval\_queries

)

# get faithfulness score

faithfulness\_score = sum(result.passing for result in eval\_results['faithfulness']) / len(eval\_results['faithfulness'])

# get relevancy score

relevancy\_score = sum(result.passing for result in eval\_results['faithfulness']) / len(eval\_results['relevancy'])

print( "> faithfulness\_score", faithfulness\_score )

print( "> relevancy\_score", relevancy\_score )

The sample code.

Copy

> faithfulness\_score 1.0

> relevancy\_score 1.0

The output.

The batch processing method helps in quickly assessing the system’s performance over a range of different queries. A faithfulness score of 1.0 signifies that the generated answers contain no hallucinations and are entirely based on retrieved context. Additionally, the Relevance score of 1.0 suggests that the answers generated consistently align with the retrieved context and the queries.

# Conclusion

This lesson has guided us through constructing and assessing an RAG pipeline using LlamaIndex, concentrating mainly on evaluating the retrieval system and the responses generated within the pipeline. Assessing LLMs and chatbots presents a challenge due to the subjective nature of their outputs. Perceptions of what constitutes a great response can vary significantly from one person to another. Therefore, examining various facets of an RAG application and evaluating each aspect individually based on specific metrics is prudent.

>> [Notebook](https://colab.research.google.com/drive/1XcCcoUi5ZbUNmTCLYXL51zeD-6mXON5T?usp=sharing).

### **RESOURCES:**

* [Response Evaluation](https://docs.llamaindex.ai/en/latest/module_guides/evaluating/usage_pattern.html)
* [Retrieval](https://docs.llamaindex.ai/en/latest/module_guides/evaluating/usage_pattern_retrieval.html) Evaluation
* **openai-cookbook-eval**

**[github.com](https://github.com/openai/openai-cookbook/blob/main/examples/evaluation/Evaluate_RAG_with_LlamaIndex.ipynb" \t "_blank)**

* **llamaindex**

**[Evaluating - LlamaIndex 🦙 0.9.15](https://docs.llamaindex.ai/en/stable/module_guides/evaluating/root.html" \t "_blank)**

[docs.llamaindex.ai](https://docs.llamaindex.ai/en/stable/module_guides/evaluating/root.html" \t "_blank)

* **golden-dataset**

[github.com](https://github.com/microsoft/promptflow-resource-hub/blob/main/sample_gallery/golden_dataset/copilot-golden-dataset-creation-guidance.md" \t "_blank)

* **RAGAS**

[github.com](https://github.com/explodinggradients/ragas/blob/main/docs/howtos/integrations/llamaindex.ipynb" \t "_blank)

* RagEvaluatorPack

##### **[Downloading a LlamaDataset from LlamaHub - LlamaIndex 🦙 0.9.15.post2](https://docs.llamaindex.ai/en/stable/examples/llama_dataset/downloading_llama_datasets.html" \t "_blank)**

[docs.llamaindex.ai](https://docs.llamaindex.ai/en/stable/examples/llama_dataset/downloading_llama_datasets.html" \t "_blank)

# LangSmith Introduction

In this lesson, we will recap LangChain components, review its essential concepts, and discuss how to use the newly introduced LangSmith platform. Additionally, we will create a basic Large Language Model application to understand its capabilities better.

# LangChain Recap

LangChain is a specialized framework designed for building LLM-powered applications. It streamlines the development of intelligent, responsive LLMs and Libraries for handling chains and agents with integrated components. It also offers [Templates](https://python.langchain.com/docs/templates) for deployable task-specific architectures and [LangSmith](https://python.langchain.com/docs/langsmith) for debugging in a testing environment. The key features of LangChain, like Models, Vector Stores, Chains, etc., have been explained in detail in the previous lesson.

💡

While LangChain is suitable for prototyping, LangSmith enables an environment for debugging, testing, and optimizing LLM apps.

# ****LangChain Hub****

LangChain Hub is a centralized repository for community-sourced prompts tailored to various use cases like classification or summarization. The Hub supports both public contributions and private organizational use, fostering a collaborative development environment. The platform's version control system enables users to track prompt modifications and maintain consistency across applications.

The Hub offers features like **Prompt Exploration**, ideal for fresh interactions with language models or specific prompts to achieve particular objectives. It also simplifies the process of finding and utilizing effective prompts for various models. Additionally, the user can share, modify, and track prompt versions with **Prompt Versioning**. It allows for the easy management of different versions of prompts, a highly relevant feature in real-world projects where reverting to earlier versions may be necessary.

The user-friendly interface of the Hub allows for prompt testing, customization, and iteration in a playground environment.

💡

Discover, version control, and experiment with different prompts for LangChain and LLMs directly in your browser: [docs.smith.langchain.com](https://docs.smith.langchain.com/hub/faq).

# LangSmith

[LangSmith](https://www.langchain.com/langsmith) provides an environment for evaluating and monitoring the quality of LLM outputs. An integral part of its functionality includes metadata monitoring, token usage, and execution time, which are crucial for resource management.

This platform facilitates the refinement of new chains and tools, potentially enhancing their efficiency and performance. Users can create diverse testing environments tailored to specific needs, enabling thorough evaluation under various conditions. Additionally, the service provides visualization tools that can aid in identifying response patterns and trends, thereby supporting a deeper understanding and assessment of performance.

Lastly, the platform supports tracing the runs associated with an active instance and testing and evaluating any prompts or answers generated.

LangSmith is designed with user-friendliness in mind. The platform offers a range of tutorials and documentation to help you get started.

The setup for LangChain requires installing the necessary libraries and configuring the required environment variables, which we will cover in the following section. For certain functionalities like tracing, you need to have a LangSmith account. Please follow the steps outlined below to set up a new account.

* Head over to the [LangSmith](https://www.langchain.com/langsmith) website and sign up for an account. You can use various supported login methods.
* Once your account is set up, go to the settings page. Here, you'll find the option to create an API key.
* Click the 'Generate API Key' button to receive your API key.

## Versioning

You can commit a prompt after implementing and debugging your chain. Add this prompt under your handle's namespace to view it in the Hub.

Copy

from langchain import hub

from langchain.prompts.chat import ChatPromptTemplate

prompt = ChatPromptTemplate.from\_template("tell me a joke about {topic}")

handle = "<YOUR\_USERNAME>"

hub.push(f"{handle}/rag", prompt)Copy

The sample code.

During evaluation, if you come up with a better idea after trying the prompt, you can push the updated prompt to the same key to "commit" a new version of the prompt. For instance, let's add a system message to the prompt.

Copy

# You may try making other changes and saving them in a new commit.

from langchain import schema

prompt.messages.insert(0,

schema.SystemMessage(

content="You are a precise, autoregressive question-answering system."

)

)Copy

The sample code.

With the saved changes, we can analyze how the change reflects the model performance. The newest version of the prompt is saved as the latest version.

Copy

# Pushing to the same prompt "repo" will create a new commit

hub.push(f"{handle}/rag-prompt", prompt)Copy

The sample code.

## Tracing

LangSmith allows users to review the inputs and outputs of each element in the chain by simplifying the process of logging runs for your LLM applications. This feature is useful when debugging your application or understanding the behavior of specific components. The following section will explore the optional environment variables that enable the tracing feature. For more information, you can [visit the documentation](https://docs.smith.langchain.com/tracing#log-runs).

## ****Serving (LangServe)****

[LangServe](https://www.langchain.com/langserve) helps developers deploy LangChain-powered applications and chains as a REST API. It is integrated with FastAPI, which makes the process of creating API endpoints easy and accessible. It is possible to quickly deploy applications by using the langserve package. The deployment process is out of the scope of this course. However, you can learn more about the process from the [Github repository](https://github.com/langchain-ai/langserve?ref=blog.langchain.dev).

# QuestionAnswering Chain & LangChain Hub

The next steps are loading data from a webpage, splitting it into smaller chunks, transforming them into embeddings, storing them on the Deep Lake vector store, and utilizing the prompt templates from the [LangSmith Hub](https://smith.langchain.com/hub).

Before exploring the code, installing the essential libraries from the Python package (pip) manager is necessary.

Copy

pip install -q langchain==0.0.346 openai==1.3.7 tiktoken==0.5.2 cohere==4.37 deeplake==3.8.11 langchainhub==0.1.14Copy

The sample command.

The next step is to set the API keys in the environment for OpenAI, utilized in the embedding generation process, and the Activeloop key, required for storing data in the cloud.

Copy

import os

os.environ["OPENAI\_API\_KEY"] = "<YOUR\_OPENAI\_API\_KEY>"

os.environ["ACTIVELOOP\_TOKEN"] = "<YOUR\_ACTIVELOOP\_API\_KEY>"Copy

The sample code.

You can optionally use the following environment variables to keep track of the runs in the LangSmith dashboard under the projects section.

Copy

os.environ["LANGCHAIN\_TRACING\_V2"]=True

os.environ["LANGCHAIN\_ENDPOINT"]="https://api.smith.langchain.com"

os.environ["LANGCHAIN\_API\_KEY"]="<YOUR\_LANGSMITH\_API\_KEY>"

os.environ["LANGCHAIN\_PROJECT"]="langsmith-intro" # if not specified, defaults to "default"Copy

The sample code.

Now, we can read the content of a webpage using the WebBaseLoader class. It will return a single instance of the Document class containing all the textual information from the mentioned address. Subsequently, the lengthy text is divided into smaller segments of 500 characters each, with no overlap, resulting in 130 chunks.

Copy

from langchain.document\_loaders import WebBaseLoader

from langchain.text\_splitter import RecursiveCharacterTextSplitter

# Loading

loader = WebBaseLoader("https://lilianweng.github.io/posts/2023-06-23-agent/")

data = loader.load()

print(len(data))

# Split

text\_splitter = RecursiveCharacterTextSplitter(chunk\_size=500, chunk\_overlap=0)

all\_splits = text\_splitter.split\_documents(data)

print(len(all\_splits))Copy

The sample code.

Copy

1

130

The output.

Chunks can be saved to the Deep Lake vector store through LangChain integration. The DeepLake class handles converting texts into embeddings via OpenAI's API and then stores these results in the cloud. The dataset can be loaded from the GenAI360 course organization, or you can use your organization name (which defaults to your username) to create the dataset. Note that this task incurs the associated costs of using OpenAI endpoints.

Copy

from langchain.embeddings import OpenAIEmbeddings

from langchain.vectorstores import DeepLake

vectorstore = DeepLake.from\_documents(

all\_splits,

dataset\_path="hub://genai360/langsmith\_intro",

embedding=OpenAIEmbeddings(), overwrite=False)Copy

The sample code.

Copy

Your Deep Lake dataset has been successfully created!

Creating 130 embeddings in 1 batches of size 130:: 100%|██████████| 1/1 [00:05<00:00, 5.81s/it] dataset (path='hub://genai360/langsmith\_intro', tensors=['text', 'metadata', 'embedding', 'id'])

tensor htype shape dtype compression

------- ------- ------- ------- -------

text text (130, 1) str None

metadata json (130, 1) str None

embedding embedding (130, 1536) float32 None

id text (130, 1) str NoneCopy

The output.

Once the data is processed, we can retrieve a prompt from the LangChain hub, which provides a ChatPromptTemplate instance. This eliminates the need for designing a prompt through trial and error, allowing us to build upon already tested implementations. The following code tagged a specific prompt version so future changes would not impact the active deployment version.

Copy

from langchain import hub

prompt = hub.pull("rlm/rag-prompt:50442af1")

print(prompt)Copy

The sample code.

Copy

ChatPromptTemplate(input\_variables=['context', 'question'], messages=[HumanMessagePromptTemplate(prompt=PromptTemplate(input\_variables=['context', 'question'], template="You are an assistant for question-answering tasks. Use the following pieces of retrieved context to answer the question. If you don't know the answer, just say that you don't know. Use three sentences maximum and keep the answer concise.\nQuestion: {question} \nContext: {context} \nAnswer:"))])

The output.

Finally, we can employ the RetrievalQA chain to fetch related documents from the database and utilize the ChatOpenAI model to use these documents to generate our final response.

Copy

# LLM

from langchain.chains import RetrievalQA

from langchain.chat\_models import ChatOpenAI

llm = ChatOpenAI(model\_name="gpt-3.5-turbo", temperature=0)

# RetrievalQA

qa\_chain = RetrievalQA.from\_chain\_type(

llm,

retriever=vectorstore.as\_retriever(),

chain\_type\_kwargs={"prompt": prompt}

)

question = "What are the approaches to Task Decomposition?"

result = qa\_chain({"query": question})

result["result"]Copy

The sample code.

Copy

The approaches to task decomposition include using LLM with simple prompting, task-specific instructions, and human inputs.

The output.

Prompt versioning supports ongoing experimentation and collaboration, effectively preventing the accidental deployment of chain components that haven't been sufficiently validated.

# Conclusion

In this lesson, we discussed how to use the LangChain [Hub](https://colab.research.google.com/corgiredirector?site=https%3A%2F%2Fsmith.langchain.com%2Fhub%3Fpage%3D1) to store and share prompts for a retrieval QA chain. The Hub is a centralized location to manage, version, and share prompts. LangSmith excels in diagnosing errors, comparing prompt effectiveness, assessing output quality, and tracking key metadata like token usage and execution time for optimizing LLM applications.

The platform also provides a detailed analysis of how different prompts affected the LLM performance. The intuitive UI and the valuable insights the platform offers make the iterative process of refining LLMs more transparent and manageable. It's evident that the LangSmith platform, even in its beta phase, has the potential to be a significant tool for developers aiming to leverage the full potential of LLMs.

LangSmith provides an immediate functionality to sift through your runs and presents metrics. These metrics are essential for quickly assessing latency and the total token count throughout your application.

>> [Notebook](https://colab.research.google.com/drive/1scwUUFOqgjqCE0CRFe19Llv86GcBt7Sh?usp=sharing).

### **RESOURCES**:

* hub-examples: LangSmith cookbook

##### **[github.com](https://github.com/langchain-ai/langsmith-cookbook/blob/main/hub-examples/retrieval-qa-chain/retrieval-qa.ipynb" \t "_blank)**

[github.com](https://github.com/langchain-ai/langsmith-cookbook/blob/main/hub-examples/retrieval-qa-chain/retrieval-qa.ipynb" \t "_blank)

* the art of LangSmith article

##### **[The Art of LangSmithing](https://betterprogramming.pub/the-art-of-langsmithing-42dcd191a220" \t "_blank)**

* [A guide to testing, evaluating, and monitoring LLM calls for production using LangSmith.](https://betterprogramming.pub/the-art-of-langsmithing-42dcd191a220" \t "_blank)
* [betterprogramming.pub](https://betterprogramming.pub/the-art-of-langsmithing-42dcd191a220" \t "_blank)

### Building Fullstack Applications with LlamaIndex: LlamaPacks & more

<https://youtu.be/wV4ajKStYSc>

### Technical Report: Boosting Cosine Similarity & Retrieval Accuracy with Intel CPUs & Deep Memory

<https://github.com/activeloopai/deeplake>