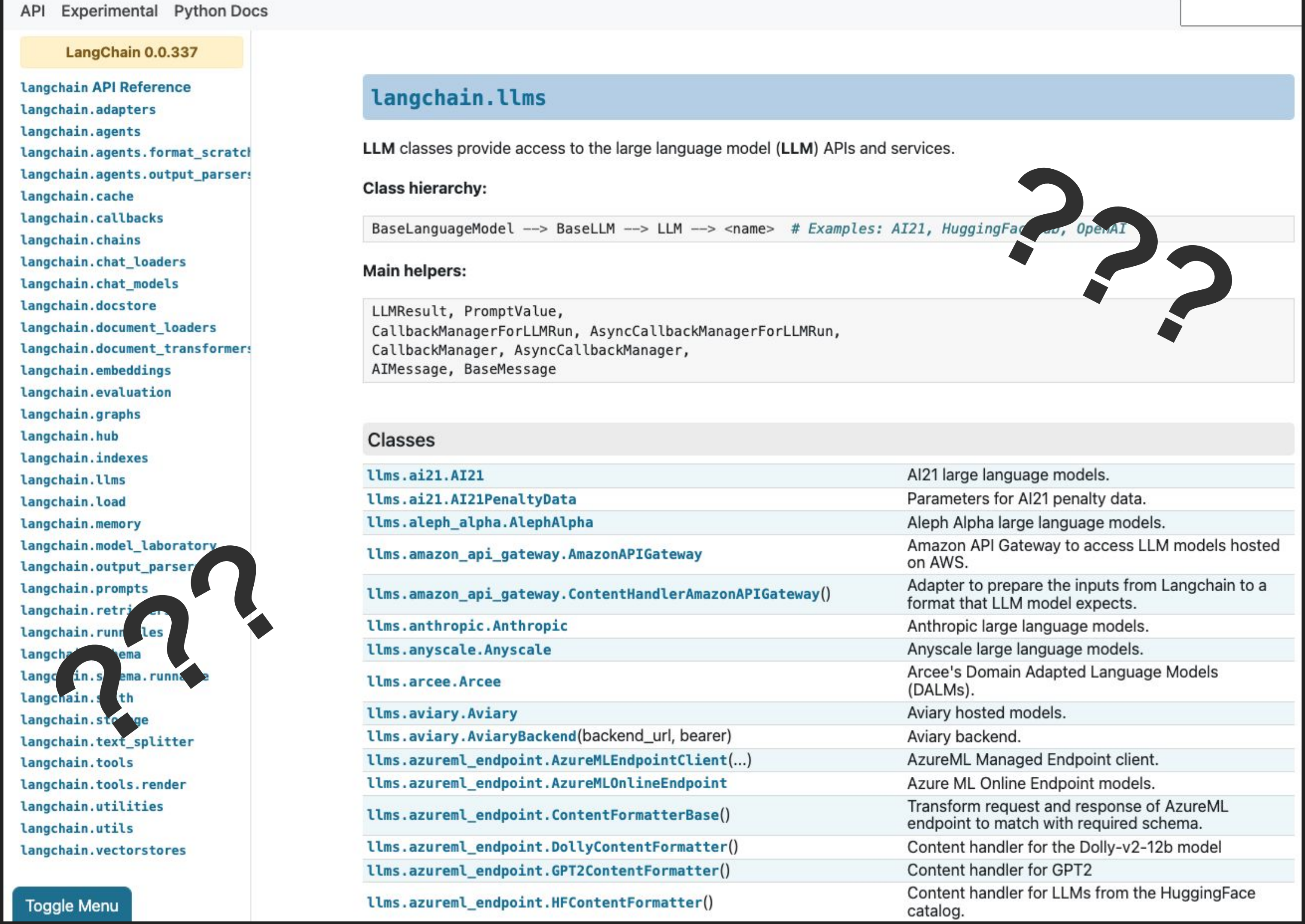
Cohere Coral

New Grads

PyLibrarian

Chat with an LLM about a Python library the moment it is released.

You



- langchain API Reference
- langchain.adapters
- langchain.agents
- langchain.agents.format_scratc
- langchain.agents.output_parser
- langchain.cache
- langchain.callbacks
- langchain.chains
- langchain.chat_loaders
- langchain.chat_models
- langchain.docstore
- langchain.document_loaders
- langchain.document_transformer
- langchain.embeddings
- langchain.evaluation
- langchain.graphs
- langchain.hub
- langchain.indexes
- langchain.llms
- langchain.load
- langchain.memory
- langchain.model_laboratory
- langchain.output_parser
- langchain.prompts
- langchain.retri ers
- langchain.runn les
- langcha ema
- langc in.s ema.runna e
- langchain.s th
- langchain.st ge
- langchain.text_splitter
- langchain.tools
- langchain.tools.render
- langchain.utilities
- langchain.utils
- langchain.vectorstores

langchain.llms

LLM classes provide access to the large language model (**LLM**) APIs and services.

Class hierarchy:

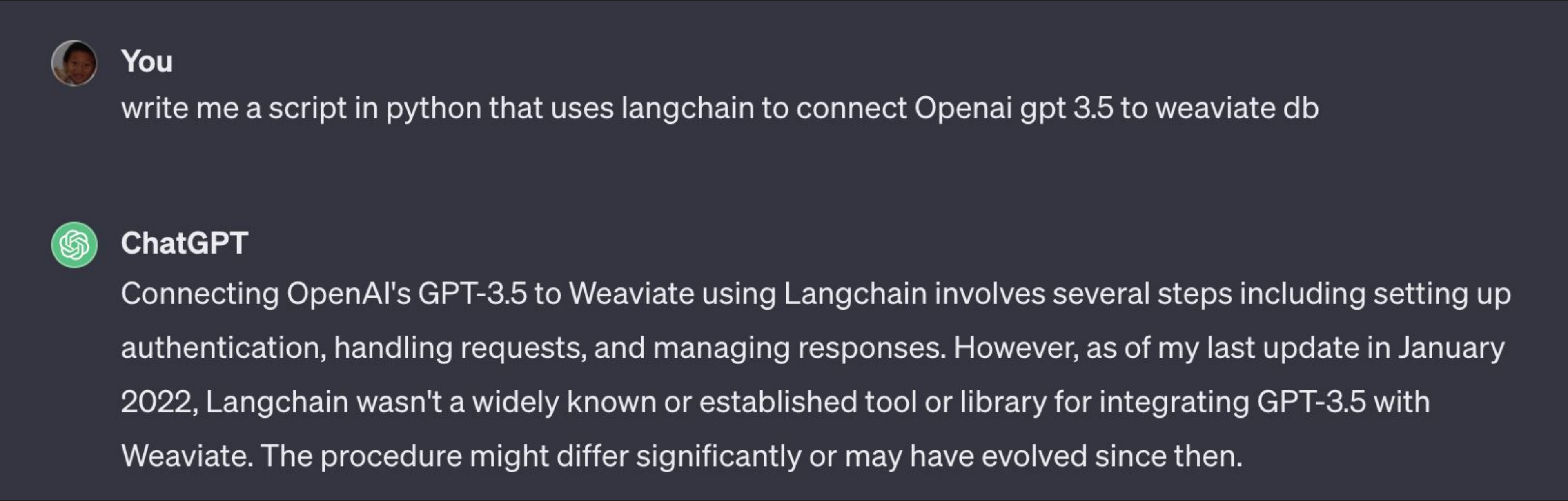
```
BaseLanguageModel --> BaseLLM --> LLM --> <name>  # Examples: AI21, HuggingFa  ab, Ope AI
```

Main helpers:

```
LLMResult, PromptValue,
CallbackManagerForLLMRun, AsyncCallbackManagerForLLMRun,
CallbackManager, AsyncCallbackManager,
AIMessage, BaseMessage
```

Classes

`llms.ai21.AI21`	AI21 large language models.
`llms.ai21.AI21PenaltyData`	Parameters for AI21 penalty data.
`llms.aleph_alpha.AlephAlpha`	Aleph Alpha large language models.
`llms.amazon_api_gateway.AmazonAPIGateway`	Amazon API Gateway to access LLM models hosted on AWS.
`llms.amazon_api_gateway.ContentHandlerAmazonAPIGateway`()	Adapter to prepare the inputs from Langchain to a format that LLM model expects.
`llms.anthropic.Anthropic`	Anthropic large language models.
`llms.anyscale.Anyscale`	Anyscale large language models.
`llms.arcee.Arcee`	Arcee's Domain Adapted Language Models (DALMs).
`llms.aviary.Aviary`	Aviary hosted models.
`llms.aviary.AviaryBackend`(backend_url, bearer)	Aviary backend.
`llms.azureml_endpoint.AzureMLEndpointClient`(...)	AzureML Managed Endpoint client.
`llms.azureml_endpoint.AzureMLOnlineEndpoint`	Azure ML Online Endpoint models.
`llms.azureml_endpoint.ContentFormatterBase`()	Transform request and response of AzureML endpoint to match with required schema.
`llms.azureml_endpoint.DollyContentFormatter`()	Content handler for the Dolly-v2-12b model
`llms.azureml_endpoint.GPT2ContentFormatter`()	Content handler for GPT2
`llms.azureml_endpoint.HFContentFormatter`()	Content handler for LLMs from the HuggingFace catalog.



You

write me a script in python that uses langchain to connect Openai gpt 3.5 to weaviate db

ChatGPT

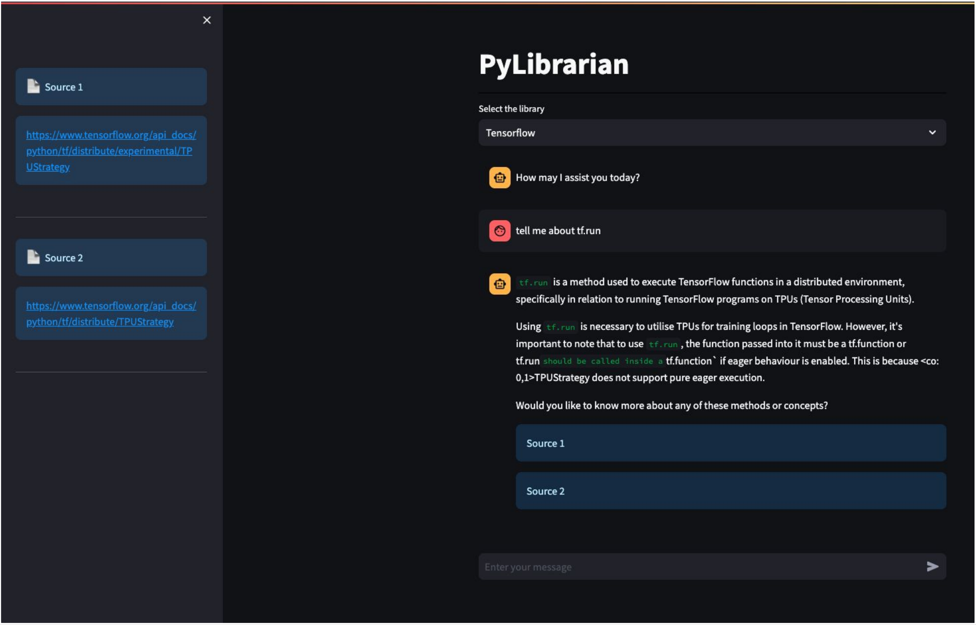
Connecting OpenAI's GPT-3.5 to Weaviate using Langchain involves several steps including setting up authentication, handling requests, and managing responses. However, as of my last update in January 2022, Langchain wasn't a widely known or established tool or library for integrating GPT-3.5 with Weaviate. The procedure might differ significantly or may have evolved since then.

Our Solution:

Expert Python chatbot that

- writes code and answers questions about different libraries
- cites ands displays its sources

- **Differentiator**: having a complete collection of documentation pages + embeddings via data-wrangling dirty-work



https://www.tensorflow.org/api_docs/python/tf/distribute/experimental/TPUStrategy

https://www.tensorflow.org/api_docs/python/tf/distribute/TPUStrategy

PyLibrarian

Select the library: Tensorflow

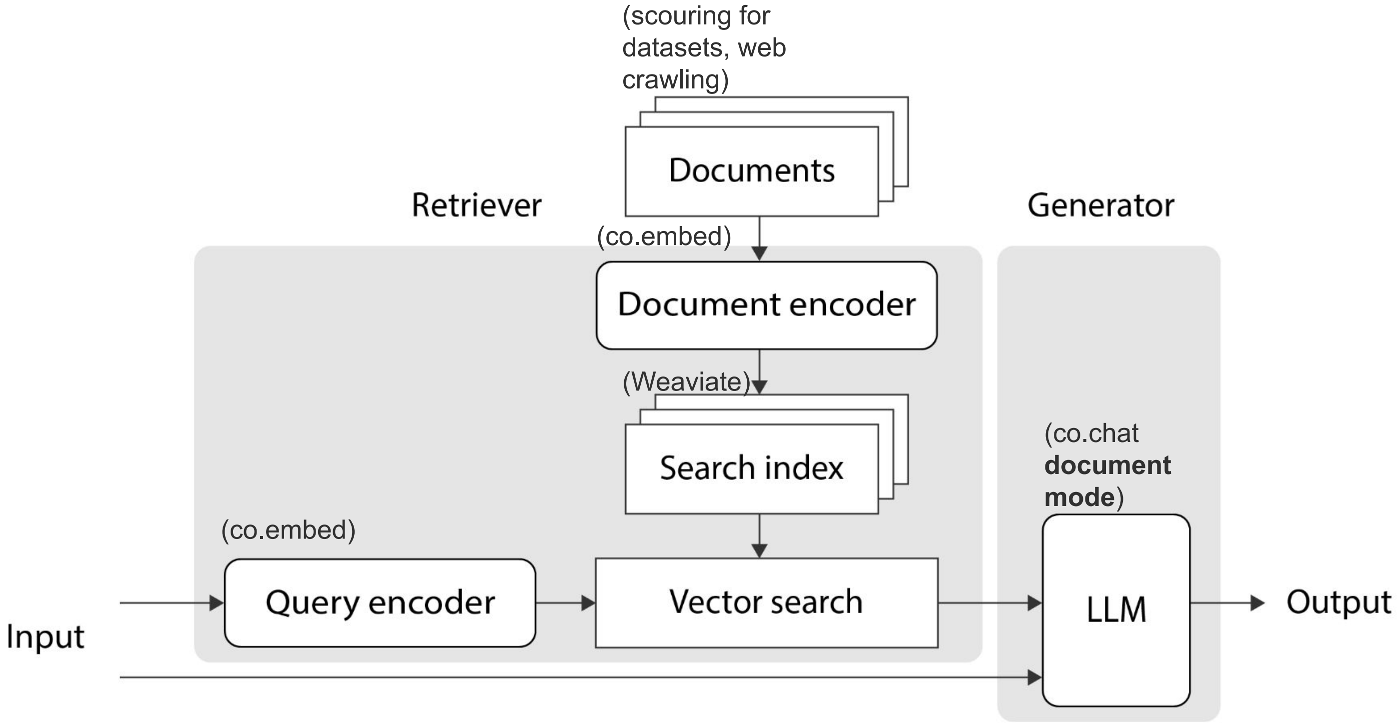
How may I assist you today?

tell me about tf.run

`tf.run` is a method used to execute TensorFlow functions in a distributed environment, specifically in relation to running TensorFlow programs on TPUs (Tensor Processing Units).

Using `tf.run` is necessary to utilise TPUs for training loops in TensorFlow. However, it's important to note that to use `tf.run`, the function passed into it must be a tf.function or tf.run `should be called inside a` tf.function` if eager behaviour is enabled. This is because <co: 0,1>TPUStrategy does not support pure eager execution.

Would you like to know more about any of these methods or concepts?



(scouring for
datasets, web
crawling)

Documents

Retriever

Generator

(co.embed)

Document encoder

(Weaviate)

Search index

(co.chat
**document
mode**)

(co.embed)

Query encoder

Vector search

LLM

Output

Input

Next Steps

- Upgrade existing UI + functionality
  - Make use of citation indices
  - Expand supported libraries, enrich data/embeddings
  - Support more ‘discovery’
- Serve more complex queries (i.e. multi-hop QA)
  - Use Cohere Query mode
- Web search to find source repos, code samples
  - Use Cohere Connector mode
  - Metaphor

Business Value

- Limited meaningful competition
- Subscription based specific LLM tools- proven value
- Scaling opportunities
  - Different languages/libraries
  - Different domains