Member-only story

Leverage LLMs Like GPT to Analyze Your Documents or Transcripts

Use prompt engineering to analyze your documents with langchain and openai in a ChatGPT-like way

Konstantin Rink · Follow

Published in Towards Data Science · 6 min read · Mar 30

(Original) photo by Laura Rivera on Unsplash.

ChatGPT is definitely one of the most popular Large Language Models (LLMs). Since the release of its beta version at the end of 2022, everyone can use the convenient chat function to ask questions or interact with the language model.

But what if we would like to ask ChatGPT questions about our own documents or about a podcast we just listened to?

The goal of this article is to show you how to leverage LLMs like GPT to analyze our documents or transcripts and then ask questions and receive answers in a ChatGPT way about the content in the documents.

tl;dr

- This article uses OpenAI's ChatGPT `gpt-3.5-turbo` model, which requires an API key.
- The `langchain` package, a framework built around LLMs, is used to load and process our documents (Prompt Engineering) and to interact with the model.
- A colab notebook containing the whole code of the article can be found here.

Prerequisites

Before writing all the code, we have to make sure that all the necessary packages are installed, API keys are created, and configurations set.

API key

To make use of ChatGPT one needs to create an OpenAI API key first. The key can be created under this link and then by clicking on the `+ Create new secret key` button.

> ***Nothing is free:*** *Generally OpenAI charges you for every 1,000 tokens. Tokens are the result of processed texts and can be words or chunks of characters. The prices per 1,000 tokens vary per model (e.g., $0.002 / 1K tokens for gpt-3.5-turbo). More details about the pricing options can be found here.*
>
> *The good thing is that OpenAI grants you a free trial usage of $18 without requiring any payment information. An overview of your current usage can be seen in your account.*

Installing the OpenAI package

We have to also install the official OpenAI package by running the following command

```
pip install openai
```

Since OpenAI needs a (valid) API key, we will also have to set the key as a environment variable:

```
import os
os.environ["OPENAI_API_KEY"] = "<YOUR-KEY>"
```

Installing the langchain package

With the tremendous rise of interest in Large Language Models (LLMs) in late 2022 (release of Chat-GPT), a package named LangChain appeared around the same time.

LangChain is a framework built around LLMs like ChatGPT. The aim of this package is to assist in the development of applications that combine LLMs with other sources of computation or knowledge. It covers the application areas like *Question Answering over specific documents* (**goal of this article**), *Chatbots*, and *Agents*. More information can be found in the documentation.

The package can be installed with the following command:

```
pip install langchain
```

Prompt Engineering

You might be wondering what *Prompt Engineering* is. It is possible to fine-

like to analyze. However, besides costs for training we would also need a lot of high-quality examples, ideally vetted by human experts (according to the documentation).

This would be overkill for just analyzing our documents or transcripts. So instead of training or fine-tuning a model, we pass the text (commonly referred to as prompt) that we would like to analyze to it. Producing or creating such high quality prompts is called *Prompt Engineering*.

> ***Note**: A good article for further reading about Prompt Engineering can be found here*

Loading the data

Depending on your use case, `langchain` offers you several "loaders" like `Facebook Chat`, `PDF`, or `DirectoryLoader` to load or read your (unstructured) text (files). The package also comes with a `YoutubeLoader` to transcribe youtube videos.

The following examples focus on the `DirectoryLoader` and `YoutubeLoader`.

Read text files with DirectoryLoader

```
from langchain.document_loaders import DirectoryLoader

loader = DirectoryLoader("", glob="*.txt")
docs = loader.load_and_split()
```

The `DirectoryLoader` takes as a first argument the **path** and as a second a **pattern** to find the documents or document types we are looking for. In our case we would load all text files (*.txt*) in the same directory as the script. The `load_and_split` function then initiates the loading.

> *Even though we might only load one text document, it makes sense to do a splitting in case we have a large file and to avoid a*

> *`NotEnoughElementsException` (minimum four documents are needed). More Information can be found here.*

Transcribe youtube videos with YoutubeLoader

LangChain comes with a YoutubeLoader module, which makes use of the `youtube_transcript_api` package. This module gathers the (generated) subtitles for a given video.

> *Not every video comes with its own subtitles. In these cases auto-generated subtitles are available. However, in some cases they have a bad quality. In these cases the usage of Whisper to transcribe audio files could be an alternative.*

The code below takes the **video id** and a **language** (default: en) as parameters.

```
from langchain.document_loaders import YoutubeLoader

loader = YoutubeLoader(video_id="XYZ", language="en")
docs = loader.load_and_split()
```

Before we continue...

In case you decide to go with **transcribed youtube videos**, consider a **proper cleaning** of, e.g., Latin1 characters (*\xa0*) **first**. I experienced in the *Question-Answering* part differences in the answers depending on which format of the same source I used.

Processing the data

LLMs like GPT can only handle a certain amount of tokens. These limitations are important when working with large(r) documents. In general, there are three ways of dealing with these limitations. One is to make use of embeddings or `vector space engine`. A second way is to try out different chaining methods like `map-reduce` or `refine`. And a third one is a combination of both.

> *A great article that provides more details about the different chaining methods and the use of a vector space engine can be found here. Also keep in mind: The more tokens you use, the more you get charged.*

In the following we combine `embeddings` with the chaining method `stuff` which "stuffs" all documents in one single prompt.

First we ingest our transcript (`docs`) into a vector space by using `OpenAIEmbeddings`. The embeddings are then stored in an in-memory

embeddings database called Chroma.

```
from langchain.embeddings.openai import OpenAIEmbeddings
from langchain.vectorstores import Chroma

embeddings = OpenAIEmbeddings()
docsearch = Chroma.from_documents(docs, embeddings)
```

After that, we define the **model_name** we would like to use to analyze our data. In this case we choose `gpt-3.5-turbo`. A full list of available models can be found here. The **temperature** parameter defines the sampling temperature. Higher values lead to more random outputs, while lower values will make the answers more focused and deterministic.

Last but not least we use the `RetrievalQA` (Question/Answer) Retriever and set the respective parameters (`llm`, `chain_type`, `retriever`).

```
from langchain.chains import RetrievalQA
from langchain.chat_models import ChatOpenAI

llm = ChatOpenAI(model_name="gpt-3.5-turbo", temperature=0.2)

qa = RetrievalQA.from_chain_type(llm=llm,
```

```
                                    chain_type="stuff",
                                    retriever=docsearch.as_retriever())
```

Asking questions

Now we are ready to ask the model questions about our documents. The code below shows how to define the query.

```
query = "What are the three most important points in the text?"
qa.run(query)
```

What do to with incomplete answers?

In some cases you might experience incomplete answers. The answer text just stops after a few words.

The reason for an incomplete answer is most likely the token limitation. If the provided prompt is quite long, the model does not have that many tokens left to give an (full) answer. One way of handling this could be to switch to a different **chain-type** like `refine`.

```
llm = ChatOpenAI(model_name="gpt-3.5-turbo", temperature=0.2)

qa = RetrievalQA.from_chain_type(llm=llm,
                                 chain_type="refine",
                                 retriever=docsearch.as_retriever())
```

However, I experienced that when using a different `chain_type` than *stuff* , I get less concrete results. Another way of handling these issues is to rephrase the question and make it more concrete.

Conclusion

Thanks to the LangChain package, one needs only a few lines of code to analyze LLMs like GPT text documents or transcripts. Since the package is relatively new, I expect many updates and code changes soon. That might affect the provided code snippets in this article.

In case you think about using LLM in your daily work or for a larger private project, you should focus on cleaning the data properly, optimizing the number of used tokens, and using best practices like setting budget limits or alarms.

I hope you enjoyed reading this article. A colab notebook with the source code can be found here.

Sources

LangChain: Introduction and Getting Started | Pinecone

arge Language M odels (LLMs) entered the world stage with the release of OpenAI's GPT-3 in 2020 [1]. Since then...

www.pinecone.io

Build a GitHub Support Bot with GPT3, LangChain, and Python | Dagster Blog

January 9, 2023 * 13 minute read * ChatGPT, ever heard of it? ChatGPT came out a few months ago and blew everyones'...

dagster.io

Prompt Engineering

Here, we discuss a few principles and techniques for writing prompts (inputs for our models) that will help you get the...

docs.cohere.ai

Retrieval

TL;DR: We are adjusting our abstractions to make it easy for other retrieval methods besides the LangChain VectorDB...

blog.langchain.dev

Data Science | NLP | Large Language Models | Machine Learning | Gpt

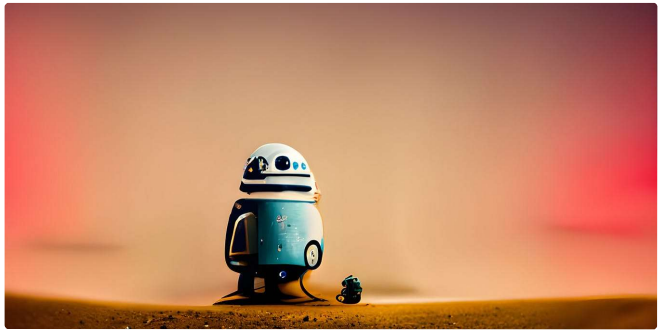
Written by Konstantin Rink

Follow

1K Followers · Writer for Towards Data Science

Team Lead Data Science @ Fintech | https://www.linkedin.com/in/konstantin-rink

More from Konstantin Rink and Towards Data Science



Konstantin Rink in Towards Data Science

Build a Transparent Question-Answering Bot for Your Documen...

Guide to developing an informative QA bot with displayed sources used

✦ · 11 min read · Jul 21

164 5

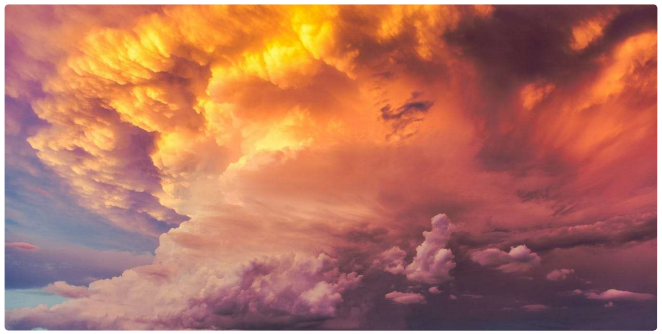
Bex T. in Towards Data Science

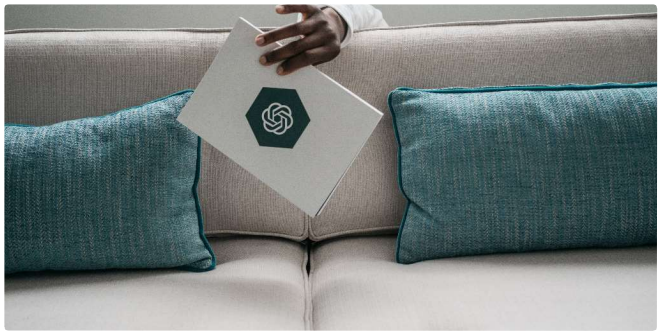
130 ML Tricks And Resources Curated Carefully From 3 Years...

Each one is worth your time

✦ · 48 min read · Aug 1

2.4K 10





 Kenneth Leung in Towards Data Science

Running Llama 2 on CPU Inference Locally for Document Q&A

Clearly explained guide for running quantized open-source LLM applications on CPUs usin...

✦ · 11 min read · Jul 18

 2K  27 

 Konstantin Rink in Towards Data Science

Run Interactive Sessions With ChatGPT In Jupyter Notebook

Use LangChain and IPyWidgets to run conversational sessions with ChatGPT about...

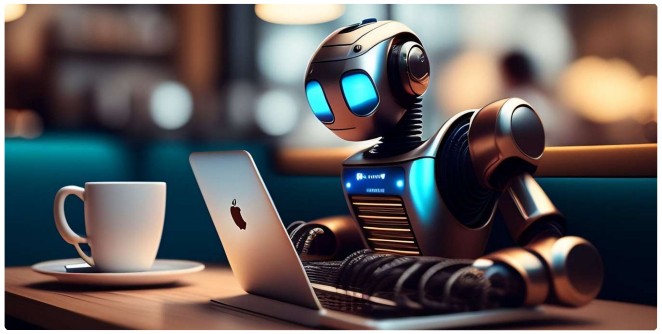
✦ · 6 min read · May 4

 142  3 

See all from Konstantin Rink

See all from Towards Data Science

Recommended from Medium



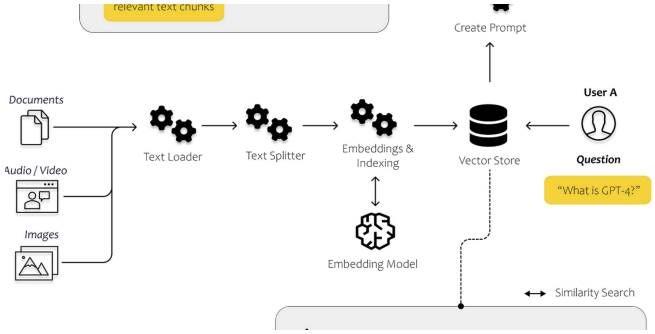
Yvann in Better Programming

Build a Chatbot on Your CSV Data With LangChain and OpenAI

Chat with your CSV file with a memory chatbot 🤖 — Made with Langchain 🦜 ...

5 min read · Jun 2

898 21



Dominik Polzer in Towards Data Science

All You Need to Know to Build Your First LLM App

A step-by-step tutorial to document loaders, embeddings, vector stores and prompt...

✦ · 26 min read · Jun 21

4.3K 40

Lists

Predictive Modeling w/ Python

20 stories · 272 saves

Natural Language Processing

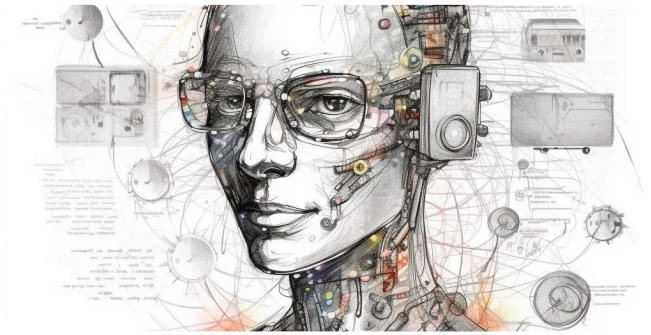
513 stories · 138 saves

Practical Guides to Machine Learning

10 stories · 285 saves

The New Chatbots: ChatGPT, Bard, and Beyond

13 stories · 83 saves



Wei-Meng Lee in Level Up Coding

Training Your Own LLM using privateGPT

Learn how to train your own language model without exposing your private data to the...

✦ · 8 min read · May 19

1.3K 11

Tomaz Bratanic in Neo4j Developer Blog

Knowledge Graphs & LLMs: Fine-Tuning Vs. Retrieval-Augmented...

What are the limitations of LLMs, and how to overcome them

12 min read · Jun 6

582 7

Maximilian Vogel in MLearning.ai

The ChatGPT list of lists: A collection of 3000+ prompts,...

Updated Aug 13, 2023. Added prompt design courses, masterclasses and tutorials.

10 min read · Feb 7

6.8K 72

Mirantha Jayathilaka, PhD

Building a Chat-AI to answer about your own data—Part I

First things first

· 6 min read · Feb 22

85 2

See more recommendations