

Venelin Valkov

Get SH*T Done with Prompt Engineering and LangChain



Build AI Applications with ChatGPT in Python

Get SH*T Done with Prompt Engineering and LangChain

Build AI Applications with ChatGPT in Python

Venelin Valkov

This book is for sale at

<http://leanpub.com/Get-Shit-Done-with-Prompt-Engineering-and-LangChain>

This version was published on 2023-06-04



This is a [Leanpub](#) book. Leanpub empowers authors and publishers with the Lean Publishing process. [Lean Publishing](#) is the act of publishing an in-progress ebook using lightweight tools and many iterations to get reader feedback, pivot until you have the right book and build traction once you do.

© 2023 Venelin Valkov

Table of Contents

Prompt Engineering	1
Why is it important?	2
What is a Prompt?	3
Prompt Structure	4
Example Prompts	5
Input and Output Format	6
 Large Language Models	 11
Tokens	12
Techniques for optimizing LLM performance	14
 Use Cases	 15
Outline a Topic	16
Write Email	17
Write and Explain Code	18
Social Media Marketing	19
Summarization	20
 LangChain	 21
LangChain Quickstart	22
Load Custom Data with Loaders	29

TABLE OF CONTENTS

Add AI with Models	31
Make LLMs Useful with Chains	33
Complex Tasks with Agents	35
Projects	37
Private GPT4All: Chat with PDF Files Using Free LLM	38
CryptoGPT: Crypto Twitter Sentiment Analysis	47

Prompt Engineering

Welcome to Prompt Engineering with Large Language Models (LLMs)! In recent years, advances in natural language processing (NLP) have led to the development of powerful language models like ChatGPT and GPT-4, which are capable of generating human-like text based on input prompts. However, the effectiveness of these models depends heavily on the quality of the prompts provided.

This is where Prompt Engineering comes in. By carefully crafting and optimizing prompts, we can use LLMs to generate text that is more accurate, relevant, and engaging. Whether you're a data scientist, a content creator, or just someone interested in exploring the capabilities of NLP, learning how to use Prompt Engineering can help you achieve your goals.

In this tutorial, we'll start by getting you set up with the tools and environment you need to use GPT-4 and the OpenAI API. Then, we'll dive into the theory and practice of Prompt Engineering, covering topics like how prompts work, how to craft effective prompts, and common mistakes to avoid.

Throughout the tutorial, I'll use easy-to-understand language and provide plenty of examples and analogies to help you understand the concepts and techniques involved. I'll also be direct and blunt in pointing out common mistakes and potential pitfalls, to help you avoid them and get the most out of your use of GPT-4 with Prompt Engineering.

So, let's get started and explore the exciting world of Prompt Engineering!

Why is it important?

Without Prompt Engineering, the output of LLMs can be unpredictable and often irrelevant. For example, without carefully crafted prompts, an LLM might generate text that is grammatically correct but semantically nonsensical, or it might fail to capture the nuances of a specific topic or domain. This can lead to wasted time and resources as users are forced to sift through irrelevant or inaccurate output.

Effective Prompt Engineering can also help us to optimize the performance of LLMs. By understanding how LLMs work and how to craft effective prompts, we can minimize common errors and pitfalls that can impact the accuracy and efficiency of LLM-generated text. This can help us achieve faster turnaround times and higher quality output, while minimizing the need for manual review and correction.

Investing in Prompt Engineering is a key step for anyone looking to use LLMs effectively, whether for content creation, conversational AI, or other applications. By understanding the principles and techniques of Prompt Engineering, we can unlock the full potential of these powerful tools and achieve better results in less time.

What is a Prompt?

A prompt is a specific instruction or request given to a computer program or language model to perform a particular task or generate a specific output. In the context of natural language processing and machine learning, prompts are often used to guide language models like ChatGPT and GPT-4 in generating text.

Here's a very simple prompt:

- 1 Write a short paragraph about the top 3 attractions to visit in Paris.

In this example, the prompt provides a clear task for the language model to generate text. Can we improve on that?

Yes! We will utilize the following elements to enhance our prompt creation process:

- **ChatGPT's conversational training:** The model was trained to provide responses in a conversational format and then evaluated based on the quality of those responses. By utilizing this information, we can create a comprehensive, repeatable framework for our prompts.
- **Role and context-specific API:** The ChatGPT API can be tailored to meet specific needs by providing it with a specific role and context. This allows for even more precise language generation that aligns with your objectives.

Next, we'll look at how to structure an effective prompt!

Prompt Structure

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/Get-Shit-Done-with-Prompt-Engineering-and-LangChain>

Components of an Effective Prompt

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/Get-Shit-Done-with-Prompt-Engineering-and-LangChain>

Template for an Effective Prompt

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/Get-Shit-Done-with-Prompt-Engineering-and-LangChain>

Example Prompt

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/Get-Shit-Done-with-Prompt-Engineering-and-LangChain>

Example Prompts

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/Get-Shit-Done-with-Prompt-Engineering-and-LangChain>

Travel Blog Post

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/Get-Shit-Done-with-Prompt-Engineering-and-LangChain>

Data Science Report

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/Get-Shit-Done-with-Prompt-Engineering-and-LangChain>

Biology Lesson

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/Get-Shit-Done-with-Prompt-Engineering-and-LangChain>

Copywrtng for SaaS Product

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/Get-Shit-Done-with-Prompt-Engineering-and-LangChain>

Climate Change Article

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/Get-Shit-Done-with-Prompt-Engineering-and-LangChain>

Input and Output Format

LLMs excel at processing various data formats beyond just plain text. They can effectively handle lists, markdown tables, and JSON structures. In this tutorial, we'll delve into the process of incorporating data in these formats into LLMs and specifying the desired response formats accordingly.

Plain Text

The easiest method to provide information for your LLM is by using plain text. Simply enclose the supplied text within triple backticks to indicate its inclusion:

```
1 Write a 2-3 sentence summary of the text:
2
3 ```
4 The Office is an American mockumentary sitcom television series that depicts
5 the everyday work lives of office employees at the Scranton, Pennsylvania,
6 branch of the fictional Dunder Mifflin Paper Company. It aired on NBC from
7 March 24, 2005, to May 16, 2013, spanning a total of nine seasons.[1] Based on
8 the 2001-2003 BBC series of the same name created by Ricky Gervais and Stephen
9 Merchant, it was adapted for American television by Greg Daniels, a veteran
10 writer for Saturday Night Live, King of the Hill, and The Simpsons. It was
11 co-produced by Daniels' Deedle-Dee Productions and Reveille Productions (later
12 Shine America), in association with Universal Television. The original
13 executive producers were Daniels, Gervais, Merchant, Howard Klein, and Ben
14 Silverman, with numerous others being promoted in later seasons.
15 ```
```

ChatGPT response:

“The Office” is a popular American sitcom that aired on NBC from 2005 to 2013, depicting the humorous and often mundane lives of office employees at the Scranton branch of the fictional Dunder Mifflin Paper Company. Based on a British series, the show was adapted for American television by Greg Daniels and gained a dedicated fanbase over its nine-season run.

Lists

Another straightforward approach is to input a list of data:

```
1 I have the following ingredients in my fridge:
2
3 - cheese
4 - eggs
5 - milk
6 - peanuts
7 - salami
8 - lavash bread
9 - chicken breasts
10 - tomatoes
11
12 Suggest 3 different dishes I can make adding only 1-2 more ingredients not on the list
13
```

ChatGPT response:

1. Cheesy Scrambled Eggs: Make a delicious and creamy scrambled eggs dish by adding the cheese and eggs from your fridge. You can enhance the flavors by adding some chopped tomatoes or diced salami.
2. Chicken Wrap: Utilize the chicken breasts and lavash bread to create a tasty chicken wrap. Grill or sauté the chicken, and then wrap it in the lavash bread along with some fresh tomatoes, lettuce, and a sauce of your choice (such as mayonnaise or yogurt-based dressing).
3. Peanut Noodle Stir-Fry: Prepare a flavorful peanut noodle stir-fry by cooking the chicken breasts and combining them with boiled noodles, sliced tomatoes, and peanuts from your fridge. Add some soy sauce, garlic, and ginger for additional seasoning.

JSON

Public resources often utilize structured data, and it is likely that ChatGPT and other LLMs have been trained on extensive JSON data (considering the abundance of JSON data on GitHub). These models handle this type of data effectively:

```
1 Use the JSON list to find the hex value of the color cyan:
2
3 [
4   {
5     "color": "red",
6     "value": "#f00"
7   },
8   {
9     "color": "cyan",
10    "value": "#0ff"
11  },
12  {
13    "color": "green",
14    "value": "#0f0"
15  },
16  {
17    "color": "blue",
18    "value": "#00f"
19  },
20 ]
```

ChatGPT response:

The hex value of the color cyan in the given JSON list is "#0ff".

Markdown

My preferred format for feeding large amounts of structured data to an LLM is using a Markdown table. LLMs have token limitations¹, and their processing time depends on the input size. Markdown tables with a high number of rows are more compact compared to JSON files. Additionally, the markdown format is widely used in the public domain.

Let's look into an example:

¹Space efficiency in Prompt Engineering²

```

1 Which Large Language Model has the longest context length? Extract it from the markd\
2 own table:
3
4 | Language Model          | Params (B) | Context Length          \
5 |                          |            |                          \
6 | -----                | -----   | -----                 \
7 | -----                |            |                          \
8 | Open Assistant (Pythia family) | 12         | 2048                    \
9 |                          |            |                          \
10 | Dolly                   | 3, 7, 12   | [2048](https://github.com/databricks\
11 labs/dolly#dolly)        |            |                          \
12 | GPT-J-6B                | 6           | [2048](https://github.com/kingoflolz\
13 /mesh-transformer-jax/#gpt-j-6b) |            |                          \
14 | GPT-NeoX-20B            | 20          | [2048](https://huggingface.co/Eleuth\
15 erAI/gpt-neox-20b)       |            |                          \
16 | StableLM-Alpha          | 3 - 65     | [4096](https://github.com/Stability-\
17 AI/StableLM#stablelm-alpha) |            |                          \
18 | MPT-7B                  | 7           | [84k (ALiBi)](https://huggingface.co\
19 /mosaicml/mpt-7b#how-is-this-model-different) |            |                          \
20 | OpenLLaMA               | 3, 7        | [2048](https://huggingface.co/h2oai)\
21 |                          |            |                          \

```

ChatGPT response:

The language model with the longest context length is MPT-7B, which has a context length of 84k (ALiBi).

Request Output Format

You can instruct the model to generate output in a format other than plain text. For instance, you can request a JSON response. However, it's important to provide at least one example of the expected response to guide the model's output:

```

1 Which Large Language Model has the longest context length? Extract it from the markd\
2 own table:
3
4 | Language Model          | Params (B) | Context Length          \
5 |                         |            |                         \
6 | ----- | ----- | ----- \
7 |-----|-----|-----|
8 | Open Assistant (Pythia family) | 12          | 2048          \
9 |                               |            |              \
10 | Dolly                        | 3, 7, 12    | [2048](https://github.com/databricks\
11 | labs/dolly#dolly)           |            |              \
12 | GPT-J-6B                     | 6           | [2048](https://github.com/kingoflolz\
13 | /mesh-transformer-jax/#gpt-j-6b) |            |              \
14 | GPT-NeoX-20B                 | 20          | [2048](https://huggingface.co/Eleuth\
15 | erAI/gpt-neox-20b)           |            |              \
16 | StableLM-Alpha               | 3 - 65      | [4096](https://github.com/Stability-\
17 | AI/StableLM#stablelm-alpha)  |            |              \
18 | MPT-7B                       | 7           | [84k (ALiBi)](https://huggingface.co\
19 | /mosaicml/mpt-7b#how-is-this-model-different) |            |              \
20 | OpenLLaMA                    | 3, 7        | [2048](https://huggingface.co/h2oai)\
21 |                               |            |              \
22 |-----|-----|-----|
23 Return your output as JSON. The format should be:
24
25 ```
26 {
27     model: "The name of the model",
28     parameters: 4,
29     maximumTokens: 918
30 }
31 ```

```

ChatGPT response:

```

1 {
2   "model": "MPT-7B",
3   "parameters": 7,
4   "maximumTokens": 84000
5 }

```

Large Language Models

Large Language Models (LLMs) are a type of artificial intelligence that have the ability to process and generate human-like responses to natural language queries. These models are typically based on deep learning algorithms, which means that they are trained on large amounts of text data to identify patterns and relationships between words, phrases, and concepts.

At their core, LLMs work by breaking down a query or prompt into smaller components, such as individual words or phrases, and then using statistical models to predict the most likely response based on the patterns and relationships it has learned during training. This process involves feeding massive amounts of text data into the model, which allows it to learn to recognize patterns and relationships between words, phrases, and concepts.

Once trained, LLMs can be used to perform a wide range of natural language processing tasks, including language translation, sentiment analysis, question answering, and more. They are particularly useful for tasks that require a high level of language understanding and context, such as text summarization or generating coherent and natural-sounding responses to user queries.

Overall, LLMs are a powerful tool for natural language processing that have the potential to transform the way we interact with computers and machines. As research and development in this area continues, we can expect to see even more advanced and sophisticated LLMs that are capable of understanding and responding to natural language queries with increasing accuracy and relevance.

Tokens

Large Language Models (LLMs) don't truly comprehend raw text. Instead, text is transformed into numerical representations known as tokens. These tokens are then fed into the model for processing.

Here's an example of tokenizing the text "Dec 31, 1993. Things are getting weird." using the OpenAI playground¹:

Tokens	Characters
10	39

A screenshot of the OpenAI playground interface showing the text "Dec 31, 1993. Things are getting weird." with 10 tokens and 39 characters. The text is displayed in a light blue box with a dark blue border. The tokens are represented by colored squares: "Dec" (blue), "31," (orange), "1993." (green), "Things" (purple), "are" (pink), "getting" (light blue), and "weird." (light green).

Tokenization Example

The same words converted to tokens using GPT-3 tokenizer:

Tokens	Characters
10	39

A screenshot of the GPT-3 tokenizer output showing the token IDs for the text "Dec 31, 1993. Things are getting weird." The output is a list of integers: [10707, 3261, 11, 9656, 13, 11597, 389, 1972, 7650, 13].

Tokenization Ids Example

Tokens are the fundamental units of text in large language models (LLMs) such as GPT-3 or ChatGPT. In simple terms, a token can be thought of as a single word, character, or subword unit. LLMs process text by breaking it down into tokens and then analyzing and generating responses based on these tokens.

However, working with tokens in LLMs comes with some common gotchas. One important consideration is the maximum token limit imposed by the model. LLMs have a maximum token capacity, and if the input exceeds this limit, it needs to be truncated or shortened. This can lead to the loss of important context or information from the text.

Here's a table of the maximum token limits for some popular LLMs:

¹Space efficiency in Prompt Engineering²

Language Model	Token Limit
T5	512
Open Assistant (Pythia family)	2048
Dolly	2048
GPT-J-6B	2048
GPT-NeoX-20B	2048
ChatGPT (GPT-3.5-Turbo)	4096
StableLM-Alpha	4096
OpenLLaMA	2048
GPT4	32K
MPT-7B	84k
Claude	100k

Another potential issue is that tokens can have different sizes and meanings depending on the model's vocabulary and tokenization scheme. For example, words like “running” and “ran” may be represented by different tokens, which can affect the model's understanding of tense or verb forms.

Furthermore, certain characters or symbols may be treated as separate tokens, impacting the overall token count. This can be relevant when dealing with limited token budgets or when calculating costs based on token usage. One example of this is the space character, which is treated as a separate token in ChatGPT.

One way to better understand how tokens are processed is to use the OpenAI playground³ for models like GPT-3 or ChatGPT. You can also calculate the number of tokens in a text using the tiktoken⁵ library.

A helpful rule of thumb is that one token generally corresponds to ~4 characters of text for common English text. This translates to roughly $\frac{3}{4}$ of a word (so 100 tokens \approx 75 words).

Being mindful of these token-related considerations is crucial when working with LLMs to ensure effective text processing and generate accurate and meaningful responses.

³Space efficiency in Prompt Engineering⁴

⁵tiktoken library⁶

Techniques for optimizing LLM performance

Optimizing the performance of Large Language Models (LLMs) is a critical step in leveraging their full potential for natural language processing tasks. There are two popular techniques that can be used to optimize the performance of LLMs and ensure that they are generating accurate and relevant responses.

One effective technique for optimizing LLM performance is to fine-tune the model on a specific dataset or task. This involves taking a pre-trained LLM and further training it on a specific set of data to improve its performance on that task. By fine-tuning the model on a specific dataset or task, developers can improve the accuracy and relevance of the model's responses for that particular use case.

Another technique for optimizing LLM performance is to use prompt engineering (what we'll mostly focus on here). This involves crafting high-quality prompts or questions that are tailored to the specific task at hand. By providing the model with clear and relevant prompts, it can more easily generate accurate and relevant responses. Prompt engineering can involve techniques such as adding context to the prompt, using multiple prompts to provide more diverse input to the model, and incorporating additional constraints or requirements into the prompt.

Overall, optimizing LLM performance requires a combination of techniques and approaches that are tailored to the specific task and use case. By leveraging the power of these techniques, developers and data scientists can ensure that their LLMs are generating accurate and relevant responses that meet the needs of their users and stakeholders.

Use Cases

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/Get-Shit-Done-with-Prompt-Engineering-and-LangChain>

Outline a Topic

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/Get-Shit-Done-with-Prompt-Engineering-and-LangChain>

Write Email

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/Get-Shit-Done-with-Prompt-Engineering-and-LangChain>

Write and Explain Code

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/Get-Shit-Done-with-Prompt-Engineering-and-LangChain>

Social Media Marketing

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/Get-Shit-Done-with-Prompt-Engineering-and-LangChain>

Generate Ideas

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/Get-Shit-Done-with-Prompt-Engineering-and-LangChain>

Tweets

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/Get-Shit-Done-with-Prompt-Engineering-and-LangChain>

Email

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/Get-Shit-Done-with-Prompt-Engineering-and-LangChain>

LinkedIn

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/Get-Shit-Done-with-Prompt-Engineering-and-LangChain>

Summarization

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/Get-Shit-Done-with-Prompt-Engineering-and-LangChain>

Detect Sentiment

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/Get-Shit-Done-with-Prompt-Engineering-and-LangChain>

Summarize and Write Content

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/Get-Shit-Done-with-Prompt-Engineering-and-LangChain>

Summarize a Web Page

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/Get-Shit-Done-with-Prompt-Engineering-and-LangChain>

References

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/Get-Shit-Done-with-Prompt-Engineering-and-LangChain>

LangChain

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/Get-Shit-Done-with-Prompt-Engineering-and-LangChain>

LangChain Quickstart

Time to start with LangChain quickly! In this part, you'll learn the basics of LangChain and how to get started with building powerful apps using OpenAI and ChatGPT. We'll start by setting up a Google Colab notebook and running a simple OpenAI model. Then, we'll dive deeper by loading an external webpage and using LangChain to ask questions using OpenAI embeddings and ChatGPT.

Finally, we'll explore prompt templates and chains, and combine all the components we've learned to create a powerful retrieval-based question answering system. By the end of this part, you'll have a solid understanding of LangChain and how to use it to build your own language models. So let's get started!

Setup

To begin using LangChain, let's include some necessary imports:

```
1 import os
2 import textwrap
3
4 import chromadb
5 import langchain
6 import openai
7 from langchain.chains import RetrievalQA
8 from langchain.chat_models import ChatOpenAI
9 from langchain.document_loaders import WebBaseLoader
10 from langchain.embeddings import OpenAIEmbeddings
11 from langchain.indexes import VectorstoreIndexCreator
12 from langchain.llms import OpenAI
13 from langchain.prompts import PromptTemplate
14 from langchain.vectorstores import Chroma
```

Before we dive into LangChain, let's create a simple function that formats text outputs:

```
1 def print_response(response: str):
2     print("\n".join(textwrap.wrap(response, width=100)))
```

The function uses the `textwrap` module to wrap the text in response so that each line is no more than 100 characters wide. Finally, the wrapped text is printed with the `"\n".join` method to concatenate the wrapped lines with newlines between them.

Next, we will set up your OpenAI API key:

```
1 os.environ["OPENAI_API_KEY"] = "YOUR OPENAI KEY"
```

You can get yours from [OpenAI website](https://platform.openai.com/account/api-keys)¹.

Using the OpenAI Model

We will now use the basic model and pass in your first prompt:

```
1 model = OpenAI(temperature=0)
2
3 print(
4     model(
5         """
6         You're Dwight K. Schrute from the Office.
7         Suggest 5 places to visit in Scranton that are connected to the TV show.
8         """
9     )
10 )
11 )
```

```
1 1. The Dunder Mifflin Paper Company - Visit the office building where the show was f\
2 ilmed and take a tour of the set.
3
4 2. Poor Richard's Pub - Enjoy a drink at the bar where the cast often hung out.
5
6 3. Steamtown National Historic Site - Take a ride on the historic train that was fea\
7 tured in the show.
8
9 4. The Scranton Cultural Center - Attend a show at the theater where the cast perfor\
10 med a play in the episode "The Duel".
11
12 5. The Mall at Steamtown - Shop at the mall where the cast went on a shopping spree \
13 in the episode "The Coup".
```

¹<https://platform.openai.com/account/api-keys>

We start by initializing an instance of the `OpenAI` class (which uses `text-davinci-003` by default) with a temperature value of 0. The temperature controls the “creativity” of the generated text, with higher values leading to more varied and surprising output. In this case, a temperature of 0 will result in the most “safe” and predictable output.

Q&A Over Webpage

LangChain’s coolest feature is its ability to integrate external sources with ChatGPT (and similar models) to incorporate knowledge from them.

Thankfully, the process is remarkably straightforward. All we need to do is load the data with a loader, create an index, and query it.

Let’s load the blog post about the [recommendation algorithm used by Twitter](https://blog.twitter.com/engineering/en_us/topics/open-source/2023/twitter-recommendation-algorithm)²:

```
1 loader = WebBaseLoader(
2     "https://blog.twitter.com/engineering/en_us/topics/open-source/2023/twitter-reco\
3 mmendation-algorithm"
4 )
```

The [WebBaseLoader](#)³ enables the loading of all text data from a specified webpage.

Let’s load the specified documents:

```
1 documents = loader.load()
2 len(documents)
```

```
1 1
```

Let’s take a look at the loaded document:

```
1 document = documents[0]
2 document.__dict__.keys()

1 dict_keys(['page_content', 'metadata'])
```

The properties are easy to understand, so let’s dive into both of them:

²https://blog.twitter.com/engineering/en_us/topics/open-source/2023/twitter-recommendation-algorithm

³https://python.langchain.com/en/latest/modules/indexes/document_loaders/examples/web_base.html

```
1 document.page_content[:100]

1 Twitter's Recommendation Algorithm
2
3 Engineering
4
5 Bac
```

The `page_content` field contains the webpage's main content with the HTML tags removed.

```
1 document.metadata

1 {
2     'source': 'https://blog.twitter.com/engineering/en_us/topics/open-source/2023/tw\
3     iter-recommendation-algorithm'
4 }
```

Our specific metadata in this case contains the URL of the webpage.

Now that we have loaded the document, let's convert into a format that LangChain can understand:

```
1 index = VectorstoreIndexCreator().from_loaders([loader])
```

`VectorstoreIndexCreator` is a component used to create an index that enables querying text documents. It splits the documents into smaller chunks, generates embeddings (i.e., numerical representations) for each document, and stores the documents and their embeddings in a vectorstore, which can then be queried to retrieve relevant documents.

With the webpage data indexed, we can now utilize the query method with a prompt that can operate on:

```
1 query = """
2 You're Dwight K. Schrute from the Office.
3 Explain the Twitter recommendation algorithm in 5 sentences
4 using analogies from the Office.
5 """
6 print_response(index.query(query))
```

```

1 The Twitter recommendation algorithm is like Dwight K. Schrute's job at Dunder Mifflin.
2
3
4 It takes the 500 million Tweets posted daily and distills them down to a handful of \
5 top Tweets that show up on your timeline, just like Dwight distills the vast amount
6 of paper at Dunder Mifflin into a few
7 important documents.
8
9 The algorithm uses a set of core models and features to extract latent
10 information from Tweet, user, and engagement data, just like Dwight uses his keen eye
11 to spot the
12 important details in the documents. It then uses a logistic regression model to rank \
13 the Tweets,
14 similar to how Dwight ranks the documents in order of importance. Finally, it traverses \
15 the graph of
16 engagements and follows to answer questions about what Tweets and Users are similar \
17 to your
18 interests, just like Dwight uses his knowledge of the office to answer questions about \
19 the people
20 and documents in the office.

```

Prompt Templates

Using f-strings to write prompts can sometimes be frustrating. Fortunately, LangChain offers a solution in the form of prompt templates:

```

1 template = """You're Dwight K. Schrute from the Office.
2
3 {context}
4
5 Answer with analogies from the Office to the question and the way Dwight speaks.
6
7 Question: {question}
8 Answer: """
9
10 prompt = PromptTemplate(template=template, input_variables=["context", "question"])
11 print(
12     prompt.format(
13         context="Paper sells are declining 10% year over year.",
14         question="How to sell paper?",
15     )
16 )

```

```
1 You're Dwight K. Schrute from the Office.
2
3 Paper sells are declining 10% year over year.
4
5 Answer with analogies from the Office to the question and the way Dwight speaks.
6
7 Question: How to sell paper?
8 Answer:
```

The `format()` method of the prompt template is called with the context and question values provided as arguments to replace the variables in the template string.

Next, we will utilize OpenAI embeddings along with a local vector database (Chroma) to store them. This is a crucial aspect of the `VectorstoreIndexCreator`, it enables the creation of a local database that can be queried to retrieve relevant documents:

```
1 embeddings = OpenAIEmbeddings()
2 db = Chroma.from_documents(documents, embeddings)
```

We can combine all components in a `RetrievalQA` chain:

```
1 chain_type_kwargs = {"prompt": prompt}
2 chain = RetrievalQA.from_chain_type(
3     llm=ChatOpenAI(temperature=0),
4     chain_type="stuff",
5     retriever=db.as_retriever(search_kwargs={"k": 1}),
6     chain_type_kwargs=chain_type_kwargs,
7 )
```

This chain allows us to query the index using the model and the prompt template, making it easier to retrieve the relevant information from a custom dataset.

Now it's time to put our `RetrievalQA` chain to the test by running a query:

```
1 query = "Explain the Twitter recommendation algorithm in 5 sentences"
2 response = chain.run(query)
3 print_response(response)
```

```
1 Well, Twitter's got this fancy algorithm that picks out the best tweets
2 from the millions of tweets posted every day.
3
4 It's like Michael Scott trying to pick the best Dundie award winners
5 from all the employees.
6
7 They use a bunch of models and features to figure out what you might like,
8 like how Dwight uses his knowledge of his coworkers to predict their behavior.
9
10 Then they rank the tweets using a big neural network,
11 kind of like how Jim ranks his pranks on Dwight.
12
13 Finally, they filter out any tweets you don't want to see,
14 like how Angela filters out any fun from the office.
15
16 And voila, you've got your personalized Twitter timeline.
```

Conclusion

You now know the basics of the LangChain library and its capabilities in working with custom data and ChatGPT/GPT-4 models. With the knowledge gained, you can now explore the various features and applications of LangChain to create powerful language-based AI apps. Next, we'll dive a bit deeper into LangChain!

Load Custom Data with Loaders

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/Get-Shit-Done-with-Prompt-Engineering-and-LangChain>

Setup

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/Get-Shit-Done-with-Prompt-Engineering-and-LangChain>

Question a Text File

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/Get-Shit-Done-with-Prompt-Engineering-and-LangChain>

Loaders

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/Get-Shit-Done-with-Prompt-Engineering-and-LangChain>

Text Splitters

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/Get-Shit-Done-with-Prompt-Engineering-and-LangChain>

Embeddings

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/Get-Shit-Done-with-Prompt-Engineering-and-LangChain>

Vectorstores

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/Get-Shit-Done-with-Prompt-Engineering-and-LangChain>

Storing Embeddings

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/Get-Shit-Done-with-Prompt-Engineering-and-LangChain>

Loading Embeddings

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/Get-Shit-Done-with-Prompt-Engineering-and-LangChain>

Question a PDF File

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/Get-Shit-Done-with-Prompt-Engineering-and-LangChain>

Conclusion

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/Get-Shit-Done-with-Prompt-Engineering-and-LangChain>

Add AI with Models

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/Get-Shit-Done-with-Prompt-Engineering-and-LangChain>

Setup

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/Get-Shit-Done-with-Prompt-Engineering-and-LangChain>

Large Language Models

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/Get-Shit-Done-with-Prompt-Engineering-and-LangChain>

Multiple Completions

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/Get-Shit-Done-with-Prompt-Engineering-and-LangChain>

Embeddings

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/Get-Shit-Done-with-Prompt-Engineering-and-LangChain>

Chat Models

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/Get-Shit-Done-with-Prompt-Engineering-and-LangChain>

Use Prompt Templates

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/Get-Shit-Done-with-Prompt-Engineering-and-LangChain>

Streaming

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/Get-Shit-Done-with-Prompt-Engineering-and-LangChain>

Conclusion

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/Get-Shit-Done-with-Prompt-Engineering-and-LangChain>

Make LLMs Useful with Chains

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/Get-Shit-Done-with-Prompt-Engineering-and-LangChain>

Setup

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/Get-Shit-Done-with-Prompt-Engineering-and-LangChain>

LLM Chain

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/Get-Shit-Done-with-Prompt-Engineering-and-LangChain>

Sequential Chain

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/Get-Shit-Done-with-Prompt-Engineering-and-LangChain>

Summarization Chain

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/Get-Shit-Done-with-Prompt-Engineering-and-LangChain>

Question Answering

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/Get-Shit-Done-with-Prompt-Engineering-and-LangChain>

Bash Chain

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/Get-Shit-Done-with-Prompt-Engineering-and-LangChain>

Conclusion

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/Get-Shit-Done-with-Prompt-Engineering-and-LangChain>

Complex Tasks with Agents

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/Get-Shit-Done-with-Prompt-Engineering-and-LangChain>

Setup

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/Get-Shit-Done-with-Prompt-Engineering-and-LangChain>

Data

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/Get-Shit-Done-with-Prompt-Engineering-and-LangChain>

Create Agent

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/Get-Shit-Done-with-Prompt-Engineering-and-LangChain>

Prompt

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/Get-Shit-Done-with-Prompt-Engineering-and-LangChain>

Analyze Bitcoin Data

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/Get-Shit-Done-with-Prompt-Engineering-and-LangChain>

Conclusion

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/Get-Shit-Done-with-Prompt-Engineering-and-LangChain>

Projects

Private GPT4All: Chat with PDF Files Using Free LLM

Have concerns about data privacy while using ChatGPT? Want an alternative to cloud-based language models that is both powerful and free? Look no further than GPT4All.

While ChatGPT offers impressive text generation capabilities, it operates on the cloud, meaning your data is sent to external servers. This raises privacy concerns for individuals and organizations who prefer to keep their data local and under their control. Moreover, using ChatGPT can get pretty expensive.

In this part, we will be using Jupyter Notebook to run the code. If you prefer to follow along, you can find the notebook on GitHub: [GitHub Repository](#)¹

GPT4All

Enter GPT4All, an ecosystem that provides customizable language models running locally on consumer-grade CPUs. With GPT4All, you can leverage the power of language models while maintaining data privacy. By running models locally, you retain full control over your data and ensure sensitive information stays secure within your own infrastructure. Additionally, GPT4All models are freely available, eliminating the need to worry about additional costs.

Currently, GPT4All supports three different model architectures: GPTJ, LLAMA, and MPT. Each architecture has its own unique features and examples that can be explored. With GPT4All, you have access to a range of models to suit your specific needs and leverage their capabilities in various applications.

Data

The PDF file we'll be using is from the Microsoft 2022 Annual Report². I've selected just two pages from it. Let's download the file:

¹<https://github.com/curiously/Get-Things-Done-with-Prompt-Engineering-and-LangChain>

²[tiktoken library](#)³

1 !gdown 1DpFisoGXsQbpQJvijuvxkLW_pg-FUUMF

Here are the pages:

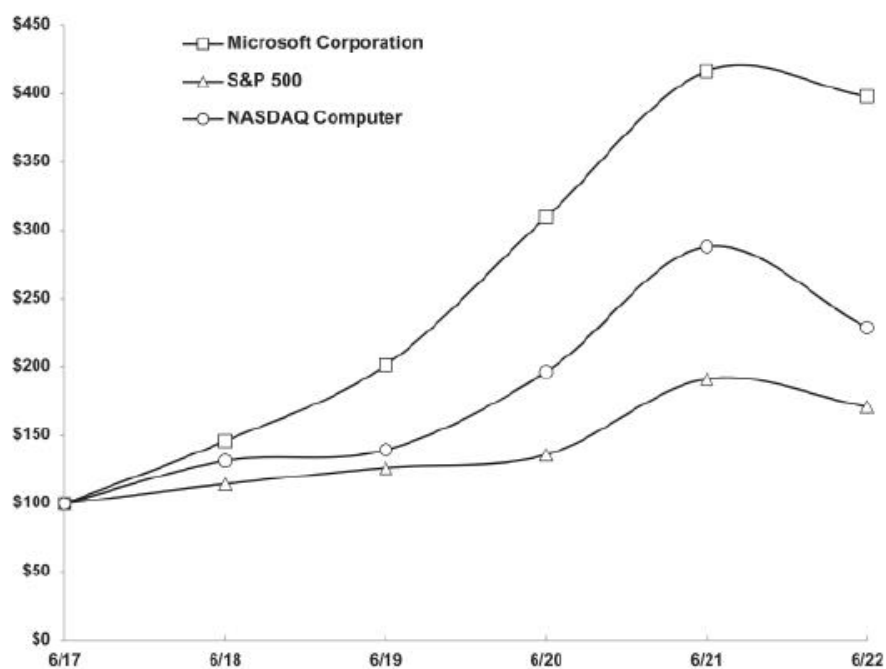
Dividends

Our Board of Directors declared the following dividends:

Declaration Date	Record Date	Payment Date	Dividend Per Share	Amount
Fiscal Year 2022				(In millions)
September 14, 2021	November 18, 2021	December 9, 2021	\$ 0.62	\$ 4,652
December 7, 2021	February 17, 2022	March 10, 2022	0.62	4,645
March 14, 2022	May 19, 2022	June 9, 2022	0.62	4,632
June 14, 2022	August 18, 2022	September 8, 2022	0.62	4,627
Total			<u>\$ 2.48</u>	<u>\$ 18,556</u>
Fiscal Year 2021				
September 15, 2020	November 19, 2020	December 10, 2020	\$ 0.56	\$ 4,230
December 2, 2020	February 18, 2021	March 11, 2021	0.56	4,221
March 16, 2021	May 20, 2021	June 10, 2021	0.56	4,214
June 16, 2021	August 19, 2021	September 9, 2021	0.56	4,206
Total			<u>\$ 2.24</u>	<u>\$ 16,871</u>

The dividend declared on June 14, 2022 was included in other current liabilities as of June 30, 2022.

STOCK PERFORMANCE
COMPARISON OF 5 YEAR CUMULATIVE TOTAL RETURN*
 Among Microsoft Corporation, the S&P 500 Index
 and the NASDAQ Computer Index



* \$100 invested on 6/30/17 in stock or index, including reinvestment of dividends. Fiscal year ending June 30.

The next file we need is the GPT4All checkpoint:

```
1 !wget https://gpt4all.io/models/ggml-gpt4all-j-v1.3-groovy.bin
```

Yes, it's massive, weighing in at over 3.5 GB! The ggml-gpt4all-j-v1.3-groovy checkpoint is the (current) best commercially licensable model, built on the GPT-J architecture, and trained by Nomic AI using the latest curated GPT4All dataset.

Setup

Let's add all the imports we'll need:

```
1 from langchain.chains import RetrievalQA
2 from langchain.document_loaders import PyPDFLoader
3 from langchain.embeddings import HuggingFaceEmbeddings
4 from langchain.llms import GPT4All
5 from langchain.text_splitter import RecursiveCharacterTextSplitter
6 from langchain.vectorstores import Chroma
7 from pdf2image import convert_from_path
```

Load PDF

Let's load the PDF file and split it into pages:

```
1 loader = PyPDFLoader("ms-financial-statement.pdf")
2 documents = loader.load_and_split()
3 len(documents)
```

```
1 2
```

The PyPDFLoader⁴ from LanChain uses the pypdf library to load and split the PDF file into pages. The load_and_split method returns a list of Document objects. Let's take a look at the first page:

```
1 print(documents[0].page_content)
```

⁴Unstructured library⁵

```

1 9
2 Dividends
3 Our Board of Directors declared the following dividends:
4
5 Declaration Date   Record Date   Payment   Date   Dividend
6 Per Share   Amount
7
8 Fiscal Year 2022           (In millions)
9
10 September 14, 2021   November 18, 2021   December 9, 2021   $ 0.62   $ 4,652
11 December 7, 2021   February 17, 2022   March 10, 2022   0.62   4,645
12 March 14, 2022   May 19, 2022   June 9, 2022   0.62   4,632
13 June 14, 2022   August 18, 2022   September 8, 2022   0.62   4,627
14 Total           $ 2.48   $ 18,556
15
16 Fiscal Year 2021
17
18 September 15, 2020   November 19, 2020   December 10, 2020   $ 0.56   $ 4,230
19 December 2, 2020   February 18, 2021   March 11, 2021   0.56   4,221
20 March 16, 2021   May 20, 2021   June 10, 2021   0.56   4,214
21 June 16, 2021   August 19, 2021   September 9, 2021   0.56   4,206
22
23 Total           $ 2.24   $ 16,871
24 The dividend declared on June 14, 2022 was included in other current liabilities as\
25 of June 30, 2022.

```

The extracted text flow is fine. Let's chunk it into smaller parts:

```

1 text_splitter = RecursiveCharacterTextSplitter(
2     chunk_size=1024,
3     chunk_overlap=64
4 )
5 texts = text_splitter.split_documents(documents)
6 len(texts)

```

```

1 3

```

The `RecursiveCharacterTextSplitter` chunks the text in 1024 characters and adds an overlap of 64 characters. The result is a list of Document objects.

Create Embeddings

We'll utilize the `HuggingFaceEmbeddings` functionality from the sentence transformers library to generate embeddings for each text chunk. Although OpenAI embeddings are available, for the sake of keeping this tutorial cost-free, we'll stick with the HuggingFace embeddings.

```
1 embeddings = HuggingFaceEmbeddings(  
2     model_name="sentence-transformers/all-MiniLM-L6-v2"  
3 )
```

We'll save the generated embeddings into a Chroma database for storage and easy retrieval:

```
1 db = Chroma.from_documents(texts, embeddings, persist_directory="db")
```

Create Chain

Loading the model is straightforward as LangChain leverages the GPT4All bindings internally:

```
1 llm = GPT4All(  
2     model="./ggml-gpt4all-j-v1.3-groovy.bin",  
3     n_ctx=1000,  
4     backend="gptj",  
5     verbose=False  
6 )
```

We specify the backend as `gptj` and set the maximum number of tokens to 1000. The verbose flag is set to `False` to avoid printing the model's output.

Next, we'll make use of a standard application of Language Models (LLMs) on texts using LangChain. We'll pass our GPT4All model to a RetrievalQA chain:


```
1 qa = RetrievalQA.from_chain_type(  
2     llm=llm,  
3     chain_type="stuff",  
4     retriever=db.as_retriever(search_kwargs={"k": 3}),  
5     return_source_documents=True,  
6     verbose=False,  
7 )
```

We're using our ChromaDB storage as the retriever for the chain. It's important to note that I've set the maximum number of documents to 3, which corresponds to the number of text chunks we have. The `return_source_documents` flag is set to `True` to return the source documents along with the answer. This is useful for debugging purposes.

Ask Questions

Finally, we're ready to ask questions to our PDF file. Let's start with a simple one:

```
1 res = qa(f"""  
2     How much is the dividend per share during during 2022?  
3     Extract it from the text.  
4 """)  
5 print(res["result"])
```

```
1 The dividend per share during 2022 is $0.62.
```

That's correct! However, the not-so-good news is that it took around 6 minutes for the generation process to complete. Hopefully, in the future, when GPT4All supports GPU inference, it will perform better and be faster.

Let's try somewhat harder question:

```
1 res = qa(f"""  
2     How much is the investment amount in Microsoft on 6/22?  
3     Extract the answer from the text.  
4 """)  
5 print(res["result"])
```

```
1 The investment amount in Microsoft on 6/22 is $309.69.
```

Unfortunately, this time the model failed to provide the correct answer. The expected value is \$397.90, but the provided response does not match the information in the table. Can you improve the prompt to get a better result?

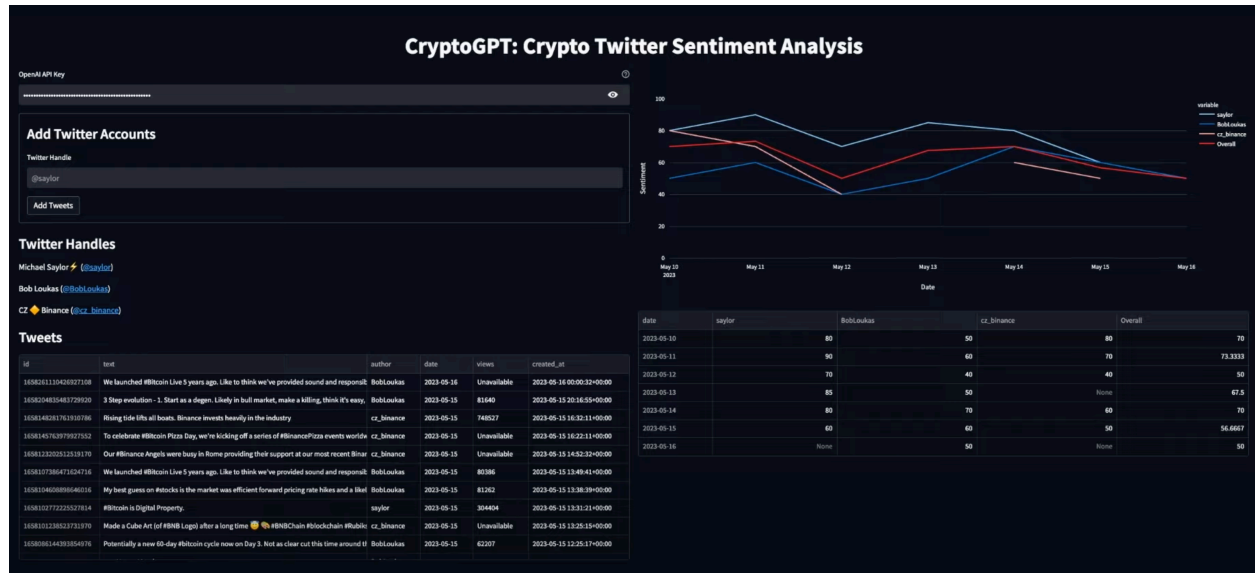
Conclusion

In conclusion, we have explored the fascinating capabilities of GPT4All in the context of interacting with a PDF file. Through this tutorial, we have seen how GPT4All can be leveraged to extract text from a PDF. While the results were not always perfect, it showcased the potential of using GPT4All for document-based conversations.

We learned how to preprocess the PDF, split it into chunks, and store the embeddings in a Chroma database for efficient retrieval. By employing the RetrievalQA chain with the GPT4All model as the backbone, we were able to pose questions and receive answers based on the document content.

It's worth noting that the performance of GPT4All can vary depending on the complexity of the PDF and the quality of the prompt. Experimenting with different prompts and refining the input can lead to more accurate and relevant responses.

CryptoGPT: Crypto Twitter Sentiment Analysis



CryptGPT Preview

Welcome to the CryptoGPT! In this tutorial, we'll dive into a fascinating project that combines Streamlit, ChatGPT, and LangChain to analyze the sentiment of tweets related to cryptocurrencies. By utilizing Streamlit, we'll create a user-friendly interface that allows us to interact with our sentiment analysis application effortlessly.

By constructing a well-crafted prompt and utilizing ChatGPT's capabilities, we'll be able to generate a sentiment score for each tweet. Each sentiment score will be between 0 (bearish) and 100 (bullish). Let's start building!

The project is hosted on Streamlit Cloud. Try it out: [CryptoGPT¹](https://cryptogpt-crypto-twitter-sentiment.streamlit.app/)

Project Setup

We'll use Python 3.11.3 for this project, and the directory structure will be as follows:

¹<https://cryptogpt-crypto-twitter-sentiment.streamlit.app/>

```
1 .
2 - .flake8
3 - .gitignore
4 - .python-version
5 - .vscode
6   - settings.json
7 - main.py
8 - requirements.txt
9 - sentiment_analyzer.py
```

Libraries

Let's install all of the libraries we'll need for this project:

```
1 pip install -U pip
2 pip install black isort langchain openai pandas plotly tweety-ns
```

Config

We'll use black and isort for formatting and import sorting. Additionally, we'll configure VSCode for the project:

```
1 {
2   "python.formatting.provider": "black",
3   "[python]": {
4     "editor.formatOnSave": true,
5     "editor.codeActionsOnSave": {
6       "source.organizeImports": true
7     }
8   },
9   "isort.args": ["--profile", "black"]
10 }
```

```
1 [flake8]
2 max-line-length = 120
```

Streamlit

Streamlit² is an open-source Python library designed for building custom web applications with ease. It allows us to create interactive and visually appealing data-driven applications

²[Space efficiency in Prompt Engineering³](#)

using Python. With Streamlit, we can quickly transform our data analysis code into shareable web applications, making it ideal for our sentiment analysis project. Let's leverage the power of Streamlit to create a seamless and user-friendly interface for analyzing the sentiment of cryptocurrency tweets.

Get Tweets

To fetch tweets for our analysis, we'll make use of the `tweetpy`⁴ library. This library interacts with Twitter's frontend API to retrieve the desired tweets:

```
1 from tweetpy.bot import Twitter
2
3 twitter_client = Twitter()
```

Now, let's fetch some tweets from Elon Musk's Twitter account:

```
1 tweets = twitter_client.get_tweets("elonmusk")
2 for tweet in tweets:
3     print(tweet.text)
4     print()
```

```
1 Spaces interview with @davidfaber starting now
2
3 Tesla shareholder meeting underway
4
5 As more satellites & ground stations are added, latency & jitter will improv\
6 e. Goal is <20ms latency.
7
8 Soros reminds me of Magneto
9
10 Tesla Powerwall does the seem for individual homes (if you have the backup switch in\
11 stalled)
12 https://t.co/mY2WHe1KE1
```

We can remove unnecessary elements like URLs, new lines, and multiple spaces from the tweets, as they are not relevant for our sentiment analysis and will save tokens for ChatGPT:

⁴[tiktoken library](#)⁵

```
1 import re
2
3 def clean_tweet(text: str) -> str:
4     text = re.sub(r"http\S+", "", text)
5     text = re.sub(r"www.\S+", "", text)
6     return re.sub(r"\s+", " ", text)
```

The first two lines use `re.sub()` to remove any URLs starting with “http://” or “www” from the text. The third line replaces consecutive whitespace characters (such as spaces, tabs, and new lines) with a single space.

We’ll use a dataframe to organize and easily visualize the tweets:

```
1 from datetime import datetime
2 from typing import Dict, List
3
4 import pandas as pd
5 from tweepy.types import Tweet
6
7 def create_dataframe_from_tweets(tweets: List[Tweet]) -> pd.DataFrame:
8     rows = []
9     for tweet in tweets:
10         clean_text = clean_tweet(tweet.text)
11         if len(clean_text) == 0:
12             continue
13         rows.append(
14             {
15                 "id": tweet.id,
16                 "text": clean_text,
17                 "author": tweet.author.username,
18                 "date": str(tweet.date.date()),
19                 "created_at": tweet.date,
20                 "views": tweet.views,
21             }
22         )
23
24     df = pd.DataFrame(
25         rows,
26         columns=["id", "text", "author", "date", "views", "created_at"]
27     )
28     df.set_index("id", inplace=True)
29     if df.empty:
30         return df
```

```
31     today = datetime.now().date()
32     df = df[
33         df.created_at.dt.date > today - pd.to_timedelta("7day")
34     ]
35     return df.sort_values(by="created_at", ascending=False)
```

This `create_dataframe_from_tweets` function iterates over each tweet, cleans the text using the `clean_tweet` function, and adds relevant information such as tweet ID, text, author, date, views, and creation timestamp to a dictionary. These dictionaries are used to create a DataFrame with tweets from the past 7 days.

Let's try it out:

```
1 df = create_dataframe_from_tweets(tweets)
2 df.head()
```

text	author	date	views	created_at
Tesla shareholder meeting underway	elonmusk	2023-05-16	8244078	2023-05-16 20:06:31+00:00
As more satellites & ground stations are added, latency & jitter will improve. Goal is <20ms latency.	elonmusk	2023-05-16	8907366	2023-05-16 17:32:32+00:00
Soros reminds me of Magneto	elonmusk	2023-05-16	39937506	2023-05-16 02:02:31+00:00
Tesla Powerwall does the seem for individual homes (if you have the backup switch installed)	elonmusk	2023-05-16	17370816	2023-05-16 01:31:51+00:00

Tweet Data UI

Our UI will have a straightforward design, with a split-screen layout consisting of two columns. The left column will be dedicated to loading the data:

```
1 col1, col2 = st.columns(2)
```

We require two pieces of information from the user - the OpenAI API key and the Twitter handles:

```
1 with col1:
2     st.text_input(
3         "OpenAI API Key",
4         type="password",
5         key="api_key",
6         placeholder="sk-...4242",
7         help="Get your API key: https://platform.openai.com/account/api-keys",
8     )
9
10    with st.form(key="twitter_handle_form", clear_on_submit=True):
11        st.subheader("Add Twitter Accounts", anchor=False)
12        st.text_input(
13            "Twitter Handle", value="", key="twitter_handle", placeholder="@saylor"
14        )
15        submit = st.form_submit_button(label="Add Tweets", on_click=on_add_author)
16
17    if st.session_state.twitter_handles:
18        st.subheader("Twitter Handles", anchor=False)
19        for handle, name in st.session_state.twitter_handles.items():
20            handle = "@" + handle
21            st.markdown(f"{name} ([{handle}])(https://twitter.com/{handle}))")
22
23    st.subheader("Tweets", anchor=False)
24
25    st.dataframe(
26        create_dataframe_from_tweets(st.session_state.tweets),
27        use_container_width=True
28    )
```

We have a password input field for the user to enter their OpenAI API key, and a form to add Twitter handles. The form has an input field where the user can enter a Twitter

handle, and a button to add the handle and retrieve tweets. We also display tweet authors in a list. Finally, there is a section displaying the tweets in a dataframe format using the `create_dataframe_from_tweets` function (defined previously).

Let's take a look at how we add tweets:

```
1 def on_add_author():
2     twitter_handle = st.session_state.twitter_handle
3     if twitter_handle.startswith("@"):
4         twitter_handle = twitter_handle[1:]
5     if twitter_handle in st.session_state.twitter_handles:
6         return
7     all_tweets = twitter_client.get_tweets(twitter_handle)
8     if len(all_tweets) == 0:
9         return
10    st.session_state.twitter_handles[twitter_handle] = all_tweets[0].author.name
11    st.session_state.tweets.extend(all_tweets)
12    st.session_state.author_sentiment[twitter_handle] = analyze_sentiment(
13        twitter_handle, st.session_state.tweets
14    )
```

The `on_add_author` function is triggered when the user clicks the “Add Tweets” button after entering a Twitter handle. It removes the “@” symbol from the handle if present, checks if the handle is already added, fetches all the tweets for that handle, and adds the data to the session state.

Finally, analyzes the sentiment of the tweets using the `analyze_sentiment` function and stores it in the session state.

Sentiment Analysis with ChatGPT

To analyze crypto sentiment using ChatGPT, we will provide it with the following prompt:

```
1 PROMPT_TEMPLATE = """
2 You're a cryptocurrency trader with 10+ years of experience. You always follow
3 the trend and follow and deeply understand crypto experts on Twitter. You
4 always consider the historical predictions for each expert on Twitter.
5
6 You're given tweets and their view count from @{twitter_handle} for specific dates:
7
8 {tweets}
9
```

```

10 Tell how bullish or bearish the tweets for each date are. Use numbers between 0
11 and 100, where 0 is extremely bearish and 100 is extremely bullish.
12
13 Use a JSON using the format:
14
15 date: sentiment
16
17 Each record of the JSON should give the aggregate sentiment for that date.
18 Return just the JSON. Do not explain.
19 """

```

The prompt sets the context of ChatGPT as an experienced cryptocurrency trader who relies on Twitter experts and considers historical predictions. It provides a variable {twitter_handle} for the Twitter handle and {tweets} for the tweet data with view counts.

The task is to analyze the sentiment of the tweets for each date and provide a JSON output containing the aggregate sentiment for each date. The sentiment values should range from 0 (extremely bearish) to 100 (extremely bullish). We require that the model doesn't provide any other output.

Let's use the prompt:

```

1 def analyze_sentiment(twitter_handle: str, tweets: List[Tweet]) -> Dict[str, int]:
2     chat_gpt = ChatOpenAI(temperature=0, model_name="gpt-3.5-turbo")
3     prompt = PromptTemplate(
4         input_variables=["twitter_handle", "tweets"], template=PROMPT_TEMPLATE
5     )
6
7     sentiment_chain = LLMChain(llm=chat_gpt, prompt=prompt)
8     response = sentiment_chain(
9         {
10             "twitter_handle": twitter_handle,
11             "tweets": create_tweet_list_for_prompt(tweets, twitter_handle),
12         }
13     )
14     return json.loads(response["text"])

```

The function `analyze_sentiment` takes a Twitter handle and a list of tweets as inputs. It creates an instance of the `ChatOpenAI` class, specifying the model as ChatGPT. It also creates a prompt and provides the variables `twitter_handle` and `tweets`.

We send a request to ChatGPT (via the `LLMChain` from `LangChain`) by passing the Twitter handle and the processed tweet list as input variables. Finally, the function returns the

parsed JSON object representing the sentiment analysis of the tweets, with each date mapped to an integer sentiment value.

The final part is this helper function:

```

1 def create_tweet_list_for_prompt(tweets: List[Tweet], twitter_handle: str) -> str:
2     df = create_dataframe_from_tweets(tweets)
3     user_tweets = df[df.author == twitter_handle]
4     if user_tweets.empty:
5         return ""
6     if len(user_tweets) > 100:
7         user_tweets = user_tweets.sample(n=100)
8
9     text = ""
10
11     for tweets_date, tweets in user_tweets.groupby("date"):
12         text += f"{tweets_date}:"
13         for tweet in tweets.itertuples():
14             text += f"\n{tweet.views} - {tweet.text}"
15     return text

```

The function creates a dataframe from the tweets using the `create_dataframe_from_tweets` function. It then keeps only the tweets authored by the given Twitter handle and limits them to 100.

The function then appends tweet texts and view counts grouped by date to a text variable.

Visualize Sentiment

We'll utilize Plotly to visualize the sentiment. We can generate a line chart to visualize the sentiment trends. Additionally, we'll display a dataframe that contains the sentiment data:

```

1 with col2:
2     sentiment_df = create_sentiment_dataframe(st.session_state.author_sentiment)
3     if not sentiment_df.empty:
4         fig = px.line(
5             sentiment_df,
6             x=sentiment_df.index,
7             y=sentiment_df.columns,
8             labels={"date": "Date", "value": "Sentiment"},
9         )
10     fig.update_layout(yaxis_range=[0, 100])

```

```
11         st.plotly_chart(fig, theme="streamlit", use_container_width=True)
12
13         st.dataframe(sentiment_df, use_container_width=True)
```

Note that we specify the y axis range as [0, 100] to ensure that the sentiment values are scaled properly.

Let's create the data frame for the sentiment chart:

```
1 def create_sentiment_dataframe(sentiment_data: Dict[str, int]) -> pd.DataFrame:
2     date_list = pd.date_range(
3         datetime.now().date() - timedelta(days=6), periods=7, freq="D"
4     )
5     dates = [str(date) for date in date_list.date]
6     chart_data = {"date": dates}
7
8     for author, sentiment_data in sentiment_data.items():
9         author_sentiment = []
10        for date in dates:
11            if date in sentiment_data:
12                author_sentiment.append(sentiment_data[date])
13            else:
14                author_sentiment.append(None)
15        chart_data[author] = author_sentiment
16
17    sentiment_df = pd.DataFrame(chart_data)
18    sentiment_df.set_index("date", inplace=True)
19
20    if not sentiment_df.empty:
21        sentiment_df["Overall"] = sentiment_df.mean(skipna=True, axis=1)
22    return sentiment_df
```

Our function generates a list of dates for the past 7 days and initializes the DataFrame with the dates as the index. Then, it populates it with sentiment values for each author, filling in missing values with None. Finally, it calculates the overall sentiment by taking the mean of the sentiment values for each date (row) as a new column.

Conclusion

In this tutorial, we covered the process of sentiment analysis on cryptocurrency tweets using LangChain and ChatGPT. We learned how to download and preprocess tweets,

visualize sentiment data using Plotly, and create a Streamlit application to interact with the sentiment analysis pipeline.

The integration of Streamlit allows us to create an interactive and intuitive interface for users to input Twitter handles, view sentiment analysis results, and visualize the sentiment trends over time.

Complete Code

```
1  import json
2  import re
3  from datetime import datetime
4  from typing import Dict, List
5
6  import pandas as pd
7  import streamlit as st
8  from langchain.chains import LLMChain
9  from langchain.chat_models import ChatOpenAI
10 from langchain.prompts import PromptTemplate
11 from tweety.types import Tweet
12
13 PROMPT_TEMPLATE = """
14 You're a cryptocurrency trader with 10+ years of experience. You always follow the t\
15 rend
16 and follow and deeply understand crypto experts on Twitter. You always consider the \
17 historical predictions for each expert on Twitter.
18
19 You're given tweets and their view count from @{twitter_handle} for specific dates:
20
21 {tweets}
22
23 Tell how bullish or bearish the tweets for each date are. Use numbers between 0 and \
24 100, where 0 is extremely bearish and 100 is extremely bullish.
25 Use a JSON using the format:
26
27 date: sentiment
28
29 Each record of the JSON should give the aggregate sentiment for that date. Return ju\
30 st the JSON. Do not explain.
31 """
32
33
```

```
34 def clean_tweet(text: str) -> str:
35     text = re.sub(r"http\S+", "", text)
36     text = re.sub(r"www.\S+", "", text)
37     return re.sub(r"\s+", " ", text)
38
39
40 def create_dataframe_from_tweets(tweets: List[Tweet]) -> pd.DataFrame:
41     rows = []
42     for tweet in tweets:
43         clean_text = clean_tweet(tweet.text)
44         if len(clean_text) == 0:
45             continue
46         rows.append(
47             {
48                 "id": tweet.id,
49                 "text": clean_text,
50                 "author": tweet.author.username,
51                 "date": str(tweet.date.date()),
52                 "created_at": tweet.date,
53                 "views": tweet.views,
54             }
55         )
56
57     df = pd.DataFrame(
58         rows, columns=["id", "text", "author", "date", "views", "created_at"]
59     )
60     df.set_index("id", inplace=True)
61     if df.empty:
62         return df
63     df = df[df.created_at.dt.date > datetime.now().date() - pd.to_timedelta("7day")]
64     return df.sort_values(by="created_at", ascending=False)
65
66
67 def create_tweet_list_for_prompt(tweets: List[Tweet], twitter_handle: str) -> str:
68     df = create_dataframe_from_tweets(tweets)
69     user_tweets = df[df.author == twitter_handle]
70     if user_tweets.empty:
71         return ""
72     if len(user_tweets) > 100:
73         user_tweets = user_tweets.sample(n=100)
74
75     text = ""
76
```

```

77     for tweets_date, tweets in user_tweets.groupby("date"):
78         text += f"{tweets_date}:"
79         for tweet in tweets.itertuples():
80             text += f"\n{tweet.views} - {tweet.text}"
81     return text
82
83
84 def analyze_sentiment(twitter_handle: str, tweets: List[Tweet]) -> Dict[str, int]:
85     chat_gpt = ChatOpenAI(temperature=0, model_name="gpt-3.5-turbo")
86     prompt = PromptTemplate(
87         input_variables=["twitter_handle", "tweets"], template=PROMPT_TEMPLATE
88     )
89
90     sentiment_chain = LLMChain(llm=chat_gpt, prompt=prompt)
91     response = sentiment_chain(
92         {
93             "twitter_handle": twitter_handle,
94             "tweets": create_tweet_list_for_prompt(tweets, twitter_handle),
95         }
96     )
97     return json.loads(response["text"])

```



```

1  import os
2  from datetime import datetime, timedelta
3  from typing import Dict
4
5  import pandas as pd
6  import plotly.express as px
7  import streamlit as st
8  from tweety.bot import Twitter
9
10 from sentiment_analyzer import analyze_sentiment, create_dataframe_from_tweets
11
12 twitter_client = Twitter()
13
14
15 def on_add_author():
16     twitter_handle = st.session_state.twitter_handle
17     if twitter_handle.startswith("@"):
18         twitter_handle = twitter_handle[1:]
19     if twitter_handle in st.session_state.twitter_handles:
20         return

```

```
21     all_tweets = twitter_client.get_tweets(twitter_handle)
22     if len(all_tweets) == 0:
23         return
24     st.session_state.twitter_handles[twitter_handle] = all_tweets[0].author.name
25     st.session_state.tweets.extend(all_tweets)
26     st.session_state.author_sentiment[twitter_handle] = analyze_sentiment(
27         twitter_handle, st.session_state.tweets
28     )
29
30
31 def create_sentiment_dataframe(sentiment_data: Dict[str, int]) -> pd.DataFrame:
32     date_list = pd.date_range(
33         datetime.now().date() - timedelta(days=6), periods=7, freq="D"
34     )
35     dates = [str(date) for date in date_list.date]
36     chart_data = {"date": dates}
37
38     for author, sentiment_data in sentiment_data.items():
39         author_sentiment = []
40         for date in dates:
41             if date in sentiment_data:
42                 author_sentiment.append(sentiment_data[date])
43             else:
44                 author_sentiment.append(None)
45         chart_data[author] = author_sentiment
46
47     sentiment_df = pd.DataFrame(chart_data)
48     sentiment_df.set_index("date", inplace=True)
49
50     if not sentiment_df.empty:
51         sentiment_df["Overall"] = sentiment_df.mean(skipna=True, axis=1)
52     return sentiment_df
53
54
55 st.set_page_config(
56     layout="wide",
57     page_title="CryptoGPT: Crypto Twitter Sentiment Analysis",
58     page_icon="https://cdn.jsdelivr.net/gh/twitter/twemoji@14.0.2/assets/72x72/1f4c8\
59 .png",
60 )
61
62
63 st.markdown(
```



```

64     "<h1 style='text-align: center'>CryptoGPT: Crypto Twitter Sentiment Analysis</h1\
65 >",
66     unsafe_allow_html=True,
67 )
68
69
70 if not "tweets" in st.session_state:
71     st.session_state.tweets = []
72     st.session_state.twitter_handles = {}
73     st.session_state.api_key = ""
74     st.session_state.author_sentiment = {}
75
76 os.environ["OPENAI_API_KEY"] = st.session_state.api_key
77
78 col1, col2 = st.columns(2)
79
80 with col1:
81     st.text_input(
82         "OpenAI API Key",
83         type="password",
84         key="api_key",
85         placeholder="sk-...4242",
86         help="Get your API key: https://platform.openai.com/account/api-keys",
87     )
88
89     with st.form(key="twitter_handle_form", clear_on_submit=True):
90         st.subheader("Add Twitter Accounts", anchor=False)
91         st.text_input(
92             "Twitter Handle", value="", key="twitter_handle", placeholder="@saylor"
93         )
94         submit = st.form_submit_button(label="Add Tweets", on_click=on_add_author)
95
96     if st.session_state.twitter_handles:
97         st.subheader("Twitter Handles", anchor=False)
98         for handle, name in st.session_state.twitter_handles.items():
99             handle = "@" + handle
100             st.markdown(f"{name} ({handle})(https://twitter.com/{handle})")
101
102     st.subheader("Tweets", anchor=False)
103
104     st.dataframe(
105         create_dataframe_from_tweets(st.session_state.tweets), use_container_width=True
106     )

```

```
107     )
108
109     with col2:
110         sentiment_df = create_sentiment_dataframe(st.session_state.author_sentiment)
111         if not sentiment_df.empty:
112             fig = px.line(
113                 sentiment_df,
114                 x=sentiment_df.index,
115                 y=sentiment_df.columns,
116                 labels={"date": "Date", "value": "Sentiment"},
117             )
118             fig.update_layout(yaxis_range=[0, 100])
119             st.plotly_chart(fig, theme="streamlit", use_container_width=True)
120
121             st.dataframe(sentiment_df, use_container_width=True)
```