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

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 Report[1]. I've selected just two pages from it. Let's download the file:

```
!gdown 1DpFisoGXsQbpQJvijuvxkLW_pg-FUUMF
```

Here are the pages:

Dividends

Our Board of Directors declared the following dividends:

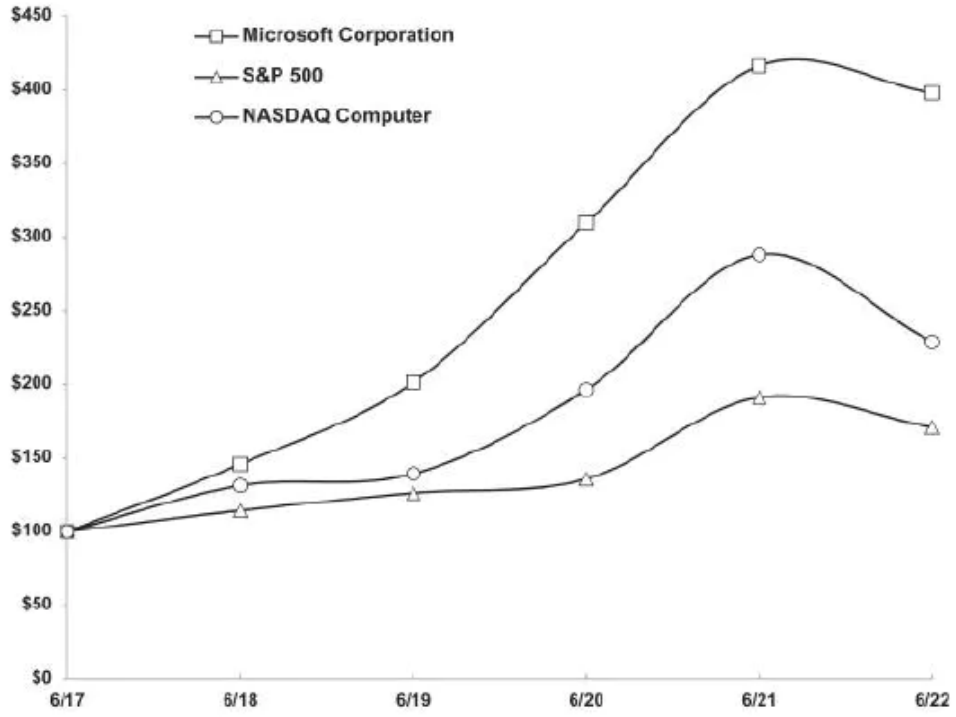
Declaration Date	Record Date	Payment Date	Dividend Per Share	Amount
Fiscal Year 2022				(In millions)
September 14, 2021	**November 18, 2021**	**December 9, 2021**	**$ 0.62**	**$ 4,652**
December 7, 2021	**February 17, 2022**	**March 10, 2022**	**0.62**	**4,645**
March 14, 2022	**May 19, 2022**	**June 9, 2022**	**0.62**	**4,632**
June 14, 2022	**August 18, 2022**	**September 8, 2022**	**0.62**	**4,627**
Total			**$ 2.48**	**$ 18,556**
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			$ 2.24	$ 16,871

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



	6/17	6/18	6/19	6/20	6/21	6/22
Microsoft Corporation	100.00	145.84	201.36	309.69	416.25	**397.90**
S&P 500	100.00	114.37	126.29	135.77	191.15	**170.86**
NASDAQ Computer	100.00	131.27	139.29	196.40	288.13	**228.71**

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

10

The next file we need is the GPT4All checkpoint:

```
!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:

```
from langchain.chains import RetrievalQA
from langchain.document_loaders import PyPDFLoader
from langchain.embeddings import HuggingFaceEmbeddings
from langchain.llms import GPT4All
from langchain.text_splitter import RecursiveCharacterTextSplitter
from langchain.vectorstores import Chroma
from pdf2image import convert_from_path
```

Load PDF

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

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

```
2
```

The `PyPDFLoader` [2] 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:

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

```
9
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
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      $   2.48  $   18,556
Fiscal Year 2021
September  15, 2020   November  19, 2020    December  10, 2020   $ 0.56  $ 4,
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      $ 2.24  $ 16,871
The dividend declared on June  14, 2022 was included in other current liabil
```

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

```
text_splitter = RecursiveCharacterTextSplitter(
    chunk_size=1024,
    chunk_overlap=64
)
texts = text_splitter.split_documents(documents)
len(texts)
```

```
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.

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

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

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

Create Chain

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

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

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:

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

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:

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

```
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:

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

```
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.

References

1. MICROSOFT 2022 ANNUAL REPORT ↩️
2. PyPDFLoader LangChain Example ↩️

I won't send you any spam, ever!
