Custom Knowledge ChatGPT with LangChain - Chat with PDFs

By Liam Ottley: YouTube

0. Installs, Imports and API Keys
1. Loading PDFs and chunking with LangChain
2. Embedding text and storing embeddings
3. Creating retrieval function
4. Creating chatbot with chat memory (OPTIONAL)

0. Installs, Imports and API Keys

```
# RUN THIS CELL FIRST!
!pip install -q langchain==0.0.150 pypdf pandas matplotlib tiktoken textract transformers openai faiss-cpu
```

```
import os
import pandas as pd
import matplotlib.pyplot as plt
from transformers import GPT2TokenizerFast
from langchain.document_loaders import PyPDFLoader
from langchain.text_splitter import RecursiveCharacterTextSplitter
from langchain.embeddings import OpenAIEmbeddings
from langchain.vectorstores import FAISS
from langchain.chains.question_answering import load_qa_chain
from langchain.llms import OpenAI
from langchain.chains import ConversationalRetrievalChain
```

```
os.environ["OPENAI_API_KEY"] = "{YOURAPIKEY}"
```

1. Loading PDFs and chunking with LangChain

```
# You MUST add your PDF to local files in this notebook (folder icon on left hand side of screen)

# Simple method - Split by pages
loader = PyPDFLoader("./attention_is_all_you_need.pdf")
pages = loader.load_and_split()
print(pages[0])

# SKIP TO STEP 2 IF YOU'RE USING THIS METHOD
chunks = pages
```

```
page_content='Attention Is All You Need\nAshish Vaswani\x03\nGoogle Brain\navaswani@google.comNoam Shazeer\x03\nGoogle Brain\nnoam@
```

```
# Advanced method - Split by chunk

# Step 1: Convert PDF to text
import textract
doc = textract.process("./attention_is_all_you_need.pdf")

# Step 2: Save to .txt and reopen (helps prevent issues)
with open('attention_is_all_you_need.txt', 'w') as f:
    f.write(doc.decode('utf-8'))

with open('attention_is_all_you_need.txt', 'r') as f:
    text = f.read()

# Step 3: Create function to count tokens
tokenizer = GPT2TokenizerFast.from_pretrained("gpt2")

def count_tokens(text: str) -> int:
    return len(tokenizer.encode(text))

# Step 4: Split text into chunks
text_splitter = RecursiveCharacterTextSplitter(
    # Set a really small chunk size, just to show.
    chunk_size = 512,
    chunk_overlap  = 24,
    length_function = count_tokens,
)

chunks = text_splitter.create_documents([text])
```

```
# Result is many LangChain 'Documents' around 500 tokens or less (Recursive splitter sometimes allows more tokens to retain context)
type(chunks[0])
```

```
langchain.schema.Document
```

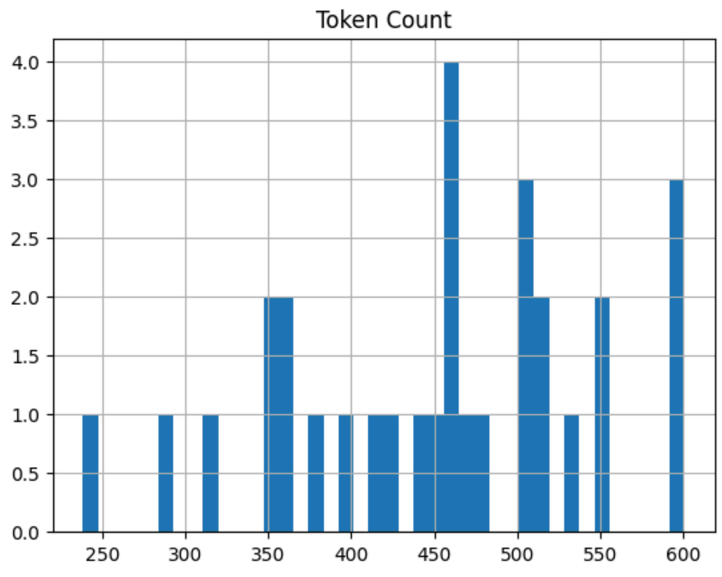
```
# Quick data visualization to ensure chunking was successful

# Create a list of token counts
token_counts = [count_tokens(chunk.page_content) for chunk in chunks]

# Create a DataFrame from the token counts
df = pd.DataFrame({'Token Count': token_counts})

# Create a histogram of the token count distribution
df.hist(bins=40, )

# Show the plot
plt.show()
```



2. Embed text and store embeddings

```
# Get embedding model
embeddings = OpenAIEmbeddings()
```

3. Setup retrieval function

```
# Check similarity search is working
query = "Who created transformers?"
docs = db.similarity_search(query)
docs[0]
```

```
Document(page_content='7 Conclusion\n\nIn this work, we presented the Transformer, the first sequence transduction model based entirely on\nattention, replacing the recurrent layers most commonly used in encoder-decoder architectures with\nmulti-headed self-attention.\nFor translation tasks, the Transformer can be trained significantly faster than architectures based\non recurrent or convolutional layers. On both WMT 2014 English-to-German and WMT 2014\nEnglish-to-French translation tasks, we achieve a new state of the art. In the former task our best\nmodel outperforms even all previously reported ensembles.\nWe are excited about the future of attention-based models and plan to apply them to other tasks. We\nplan to extend the Transformer to problems involving input and output modalities other than text and\nto investigate local, restricted attention mechanisms to efficiently handle large inputs and outputs\nsuch as images, audio and video. Making generation less sequential is another research goals of ours.\nThe code we used to train and evaluate our models is available at https://github.com/\ntensorflow/tensor2tensor.\n\nAcknowledgements We are grateful to Nal Kalchbrenner and Stephan Gouws for their fruitful\ncomments, corrections and inspiration.\n\nReferences\n[1] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. arXiv preprint\n\narXiv:1607.06450, 2016.\n\n[2] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly\n\nlearning to align and translate. CoRR, abs/1409.0473, 2014.\n\n[3] Denny Britz, Anna Goldie, Minh-Thang Luong, and Quoc V. Le. Massive exploration of neural\n\nmachine translation architectures. CoRR, abs/1703.03906, 2017.\n\n[4] Jianpeng Cheng, Li Dong, and Mirella Lapata. Long short-term memory-networks for machine\n\nreading. arXiv preprint arXiv:1601.06733, 2016.', metadata={})
```

```
# Create QA chain to integrate similarity search with user queries (answer query from knowledge base)

chain = load_qa_chain(OpenAI(temperature=0), chain_type="stuff")

query = "Who created transformers?"
docs = db.similarity_search(query)

chain.run(input_documents=docs, question=query)
```

```
' Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin crea
ted transformers.'
```

5. Create chatbot with chat memory (OPTIONAL)

```
from IPython.display import display
import ipywidgets as widgets

# Create conversation chain that uses our vectordb as retriver, this also allows for chat history management
qa = ConversationalRetrievalChain.from_llm(OpenAI(temperature=0.1), db.as_retriever())


chat_history = []

def on_submit(_):
    query = input_box.value
    input_box.value = ""

    if query.lower() == 'exit':
        print("Thank you for using the State of the Union chatbot!")
        return

    result = qa({"question": query, "chat_history": chat_history})
    chat_history.append((query, result['answer']))

    display(widgets.HTML(f'<b>User:</b> {query}'))
    display(widgets.HTML(f'<b><font color="blue">Chatbot:</font></b> {result["answer"]}'))

print("Welcome to the Transformers chatbot! Type 'exit' to stop.")

input_box = widgets.Text(placeholder='Please enter your question:')
input_box.on_submit(on_submit)

display(input_box)
```

```
Welcome to the Transformers chatbot! Type 'exit' to stop.
```

Please enter your question:

User: Who created transformers?

Chatbot: Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin created transformers.

User: Were they smart?

Chatbot: I don't know.