```
!pip install langchain
```

Show hidden output

```
!pip install google-generativeai
```

Show hidden output

```
!pip install sentence-transformers
```

Show hidden output

```
!pip install faiss-cpu
```

Show hidden output

```
!pip install pymupdf
```

Show hidden output

```
!pip install transformers
```

Show hidden output

```
!pip install langchain
```

Show hidden output

```
!pip install -U langchain-community
```

Show hidden output

```
from langchain.text_splitter import RecursiveCharacterTextSplitter
from langchain.document_loaders import PyPDFLoader
```

```
from google.colab import drive
drive.mount("/content/drive")
```

```
Mounted at /content/drive
```

```
import os
```

```
from glob import glob
```

```
folder_path="/content/drive/MyDrive/RAG"
```

```
print(folder_path)
```

```
/content/drive/MyDrive/RAG
```

```
pdf_paths = glob(os.path.join(folder_path, "*.pdf"))
```

```
pdf_filenames = [os.path.basename(path) for path in pdf_paths]
```

```
print("PDF Paths:\n", pdf_paths)
print("\nPDF Filenames:\n", pdf_filenames)
```

```
.03762v7.pdf', '/content/drive/MyDrive/RAG/2005.11401v4.pdf', '/content/drive/MyDrive/RAG/2005.14165v4.

4.pdf', '2005.14165v4.pdf']
```

```
!pip install google-generativeai
```

Show hidden output

```
all_docs = []


pdf_paths = [
    '/content/drive/MyDrive/RAG/1706.03762v7.pdf',
    '/content/drive/MyDrive/RAG/2005.11401v4.pdf',
    '/content/drive/MyDrive/RAG/2005.14165v4.pdf'
]


!pip install pypdf
```

Show hidden output

```
for file in pdf_paths:
  loader = PyPDFLoader(file)
  all_docs.extend(loader.load())


Start coding or generate with AI.


print(all_docs)
```

```
[Document(metadata={'producer': 'pdfTeX-1.40.25', 'creator': 'LaTeX with hyperref', 'creationdate': '20
```

```
splitter = RecursiveCharacterTextSplitter(chunk_size=700, chunk_overlap=200)
chunks = splitter.split_documents(all_docs)


from langchain.vectorstores import FAISS
from langchain.embeddings import HuggingFaceEmbeddings


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

```
<ipython-input-32-323160131>:1: LangChainDeprecationWarning: The class `HuggingFaceEmbeddings` was depr
  embedding = HuggingFaceEmbeddings(model_name="sentence-transformers/all-MiniLM-L6-v2")
/usr/local/lib/python3.11/dist-packages/huggingface_hub/utils/_auth.py:94: UserWarning:
The secret `HF_TOKEN` does not exist in your Colab secrets.
To authenticate with the Hugging Face Hub, create a token in your settings tab (https://huggingface.co/
You will be able to reuse this secret in all of your notebooks.
Please note that authentication is recommended but still optional to access public models or datasets.
  warnings.warn(
modules.json: 100%                                                  349/349 [00:00<00:00, 18.9kB/s]
config_sentence_transformers.json: 100%                             116/116 [00:00<00:00, 6.76kB/s]
README.md: 100%                                                     10.5k/10.5k [00:00<00:00, 531kB/s]
sentence_bert_config.json: 100%                                     53.0/53.0 [00:00<00:00, 3.08kB/s]
config.json: 100%                                                   612/612 [00:00<00:00, 41.4kB/s]
model.safetensors: 100%                                             90.9M/90.9M [00:01<00:00, 66.1MB/s]
tokenizer_config.json: 100%                                         350/350 [00:00<00:00, 20.8kB/s]
vocab.txt: 100%                                                     232k/232k [00:00<00:00, 660kB/s]
tokenizer.json: 100%                                                466k/466k [00:00<00:00, 1.33MB/s]
special_tokens_map.json: 100%                                       112/112 [00:00<00:00, 5.70kB/s]
config.json: 100%                                                   190/190 [00:00<00:00, 13.2kB/s]
```

Start coding or generate with AI.

```
%pip install --upgrade --quiet  sentence-transformers
```

```
db = FAISS.from_documents(chunks, embedding)
retriever = db.as_retriever(search_kwargs={"k": 4})
```

```
from langchain.llms import HuggingFacePipeline
from transformers import pipeline
```

```
qa_pipeline = pipeline("text-generation", model="google/flan-t5-base", max_new_tokens=256)
llm = HuggingFacePipeline(pipeline=qa_pipeline)
```

```
config.json: 100%                                                   1.40k/1.40k [00:00<00:00, 20.5kB/s]
model.safetensors: 100%                                             990M/990M [00:32<00:00, 84.5MB/s]
generation_config.json: 100%                                        147/147 [00:00<00:00, 6.52kB/s]
tokenizer_config.json: 100%                                         2.54k/2.54k [00:00<00:00, 101kB/s]
spiece.model: 100%                                                  792k/792k [00:02<00:00, 342kB/s]
tokenizer.json: 100%                                                2.42M/2.42M [00:00<00:00, 2.68MB/s]
special_tokens_map.json: 100%                                       2.20k/2.20k [00:00<00:00, 181kB/s]
Device set to use cpu
The model 'T5ForConditionalGeneration' is not supported for text-generation. Supported models are ['Pef
<ipython-input-36-2289353153>:2: LangChainDeprecationWarning: The class `HuggingFacePipeline` was depre
  llm = HuggingFacePipeline(pipeline=qa_pipeline)
```

```
from langchain.chains import RetrievalQA
```

```
qa_chain = RetrievalQA.from_chain_type(
    llm=llm,
    retriever=retriever,
    return_source_documents=True
)


def askQuestion(query):
    result = qa_chain({"query": query})
    print(" Answer:", result["result"])
    print(" Source")
    for doc in result["source_documents"]:
        print(f"- {doc.metadata.get('source', 'unknown')}")


askQuestion("What is the main contribution of the second paper?")
```

Show hidden output

```
askQuestion("What is Encoder and Decoder Stacks")
```

```
 Answer: Use the following pieces of context to answer the question at the end. If you don't know the a

Decoder: The decoder is also composed of a stack of N = 6identical layers. In addition to the two
sub-layers in each encoder layer, the decoder inserts a third sub-layer, which performs multi-head
attention over the output of the encoder stack. Similar to the encoder, we employ residual connections
around each of the sub-layers, followed by layer normalization. We also modify the self-attention
sub-layer in the decoder stack to prevent positions from attending to subsequent positions. This
masking, combined with fact that the output embeddings are offset by one position, ensures that the
predictions for position i can depend only on the known outputs at positions less than i.
3.2 Attention

Decoder: The decoder is also composed of a stack of N = 6identical layers. In addition to the two
sub-layers in each encoder layer, the decoder inserts a third sub-layer, which performs multi-head
attention over the output of the encoder stack. Similar to the encoder, we employ residual connections
around each of the sub-layers, followed by layer normalization. We also modify the self-attention
sub-layer in the decoder stack to prevent positions from attending to subsequent positions. This
masking, combined with fact that the output embeddings are offset by one position, ensures that the
predictions for position i can depend only on the known outputs at positions less than i.
3.2 Attention

wise fully connected feed-forward network. We employ a residual connection [11] around each of
the two sub-layers, followed by layer normalization [ 1]. That is, the output of each sub-layer is
LayerNorm(x + Sublayer(x)), where Sublayer(x) is the function implemented by the sub-layer
itself. To facilitate these residual connections, all sub-layers in the model, as well as the embedding
layers, produce outputs of dimension dmodel = 512.
Decoder: The decoder is also composed of a stack of N = 6identical layers. In addition to the two
sub-layers in each encoder layer, the decoder inserts a third sub-layer, which performs multi-head

wise fully connected feed-forward network. We employ a residual connection [11] around each of
the two sub-layers, followed by layer normalization [ 1]. That is, the output of each sub-layer is
LayerNorm(x + Sublayer(x)), where Sublayer(x) is the function implemented by the sub-layer
itself. To facilitate these residual connections, all sub-layers in the model, as well as the embedding
layers, produce outputs of dimension dmodel = 512.
Decoder: The decoder is also composed of a stack of N = 6identical layers. In addition to the two
sub-layers in each encoder layer, the decoder inserts a third sub-layer, which performs multi-head

Question: What is Encoder and Decoder Stacks
Helpful Answer:
 Source
- /content/drive/MyDrive/RAG/1706.03762v7.pdf
- /content/drive/MyDrive/RAG/1706.03762v7.pdf
- /content/drive/MyDrive/RAG/1706.03762v7.pdf
- /content/drive/MyDrive/RAG/1706.03762v7.pdf
```

Start coding or generate with AI.