WELCOME

This notebook will guide you through two increasingly significant applications in the realm of Generative AI: RAG (Retrieval Augmented Generation) chatbots and text summarization for big text.

Through two distinct projects, you will explore these technologies and enhance your skills. Detailed descriptions of the projects are provided below.

Project 1: Building a Chatbot with a PDF Document (RAG)

In this project, you will develop a chatbot using a provided PDF document from web page. You will utilize the Langchain framework along with a large language model (LLM) such as GPT or Gemini. The chatbot will leverage the Retrieval Augmented Generation (RAG) technique to comprehend the document's content and respond to user queries effectively.

Project Steps:

- **1.PDF Document Upload:** Upload the provided PDF document from web page (https://aclanthology.org/N19-1423.pdf) (BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding).
- **2.Chunking:** Divide the uploaded PDF document into smaller segments (chunks). This facilitates more efficient information processing by the LLM.
- **3.ChromaDB Setup:**
  - Save ChromaDB to your Google Drive.
  - Retrieve ChromaDB from your Drive to begin using it in your project.
  - ChromaDB serves as a vector database to store embedding vectors generated from your document.
- **4.Embedding Vectors Creation:**
  - Convert the chunked document into embedding vectors. You can use either GPT or Gemini embedding models for this purpose.
  - If you choose the Gemini embedding model, set "task_type" to "retrieval_document" when converting the chunked document.
- **5.Chatbot Development:**
  - Utilize the **load_qa_chain** function from the Langchain library to build the chatbot.
  - This function will interpret user queries, retrieve relevant information from **ChromaDB**, and generate responses accordingly.

Install Libraries

```
!pip install -qU langchain-community # access the tools and components (like DocumentLoade

!pip install -qU langchain-openai # utilize OpenAI's models for the "Generation" part

# !pip install -qU langchain-google-community        # Google's LLMs for the "Generation" |


!pip install -qU langchain-chroma # the "Store" step of the "Indexing" pipeline

!pip install -qU pypdfium2 #  the "Load" step of the "Indexing" pipeline, DocumentLoader t
```

```
#inspect what exactly is going on inside your chain or agent. The best way to do this is w

# import getpass
# import os

# os.environ["LANGSMITH_TRACING"] = "true"
# os.environ["LANGSMITH_API_KEY"] = getpass.getpass()

# https://smith.langchain.com/o/b249c21a-9672-4a8d-8cc6-c968dbdc2fef/projects/p/4e4bb92c-e
```

```
.........
```

Access Google Drive

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

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_re
```

Entering Your OpenAI or Google Gemini API Key.

```
import os
from google.colab import userdata
os.environ['OPENAI_API_KEY']=userdata.get('OPENAI_API_KEY')
```

Loading PDF Document

```

# PDF reader function
# It reads PDFs page by page and outputs Document objects (LangChain’s standard format). R
a List[Document] → perfect, because the next step (chunking with RecursiveCharacterTextSpl
expects this format
from langchain_community.document_loaders import PyPDFium2Loader

def read_doc(file_path: str):
    """
    Reads a PDF file and returns its pages as LangChain Document objects.
    Args:
        file_path (str): Path to the PDF file.
    Returns:
        List[Document]: A list of Document objects, one per page.
    """

    loader = PyPDFium2Loader(file_path)
    pdf_documents = loader.load()          # Reads PDF page by page
    return pdf_documents
```

```
pdf = read_doc('/content/drive/MyDrive/nlp/bert_article.pdf')

print(f"-" * 55)

print(f"The pdf file has {len(pdf)} pages.")


```

```
-------------------------------------------------------
The pdf file has 16 pages.
```

```
pdf
```

Show hidden output

```
pdf[0].page_content   # text of first page

```

```
'Proceedings of NAACL-HLT 2019, pages 4171–4186\nMinneapolis, Minnesota, June 2 - June 7, 2019. \nc 2019 Associatio
n for Computational Linguistics\n4171\nBERT: Pre-training of Deep Bidirectional Transformers for\nLanguage Understa
nding\nJacob Devlin Ming-Wei Chang Kenton Lee Kristina Toutanova\nGoogle AI Language\n{jacobdevlin,mingweichang,ken
tonl,kristout}@google.com\nAbstract\nWe introduce a new language representa\x02tion model called BERT, which stands
for\nBidirectional Encoder Representations from\nTransformers. Unlike recent language repre\x02sentation models (Pe
ters et al., 2018a; Rad\x02ford et al., 2018), BERT is designed to pre\x02train deep bidirectional representations
from\nunlabeled text by jointly conditioning on both\nleft and right context in all layers. As a re\x02sult, the pr
e-trained BERT model can be fine\x02tuned with just one additional output layer\nto create state-of-the-art models
```

```
print(pdf[0].page_content[:1250])
```

```
Proceedings of NAACL-HLT 2019, pages 4171–4186
Minneapolis, Minnesota, June 2 - June 7, 2019.
c 2019 Association for Computational Linguistics
```

```
4171
BERT: Pre-training of Deep Bidirectional Transformers for
Language Understanding
Jacob Devlin Ming-Wei Chang Kenton Lee Kristina Toutanova
Google AI Language
{jacobdevlin,mingweichang,kentonl,kristout}@google.com
Abstract
We introduce a new language representa tion model called BERT, which stands for
Bidirectional Encoder Representations from
Transformers. Unlike recent language repre sentation models (Peters et al., 2018a; Rad ford et al., 2018), BERT is d
unlabeled text by jointly conditioning on both
left and right context in all layers. As a re sult, the pre-trained BERT model can be fine tuned with just one addit
to create state-of-the-art models for a wide
range of tasks, such as question answering and
language inference, without substantial task specific architecture modifications.
BERT is conceptually simple and empirically
powerful. It obtains new state-of-the-art re sults on eleven natural language processing
tasks, including pushing the GLUE score to
80.5% (7.7% point absolute improvement),
MultiN
```

```
1 from IPython.display import Markdown
2
3 Markdown(pdf[2].page_content)
```

4173 BERT BERT E[CLS] E1 E[SEP] ... EN E1' ... EM' C T1 T[SEP] ... TN T1' ... TM' [CLS] Tok 1 [SEP] ... Tok N Tok 1 ... TokM Question Paragraph Start/End Span BERT E[CLS] E1 E[SEP] ... EN E1' ... EM' C T1 T[SEP] ... TN T1' ... TM' [CLS] Tok 1 [SEP] ... Tok N Tok 1 ... TokM Masked Sentence A Masked Sentence B Pre-training Fine-Tuning NSP Mask LM Mask LM Unlabeled Sentence A and B Pair SQuAD Question Answer Pair MNLI NER Figure 1: Overall pre-training and fine-tuning procedures for BERT. Apart from output layers, the same architec tures are used in both pre-training and fine-tuning. The same pre-trained model parameters are used to initialize models for different down-stream tasks. During fine-tuning, all parameters are fine-tuned. [CLS] is a special symbol added in front of every input example, and [SEP] is a special separator token (e.g. separating ques tions/answers). ing and auto-encoder objectives have been used for pre-training such models (Howard and Ruder, 2018; Radford et al., 2018; Dai and Le, 2015). 2.3 Transfer Learning from Supervised Data There has also been work showing effective trans fer from supervised tasks with large datasets, such as natural language inference (Conneau et al., 2017) and machine translation (McCann et al., 2017). Computer vision research has also demon strated the importance of transfer learning from large pre-trained models, where an effective recipe is to fine-tune models pre-trained with Ima geNet (Deng et al., 2009; Yosinski et al., 2014). 3 BERT We introduce BERT and its detailed implementa tion in this section. There are two steps in our framework: pre-training and fine-tuning. Dur ing pre-training, the model is trained on unlabeled data over different pre-training tasks. For fine tuning, the BERT model is first initialized with the pre-trained parameters, and all of the param eters are fine-tuned using labeled data from the downstream tasks. Each downstream task has sep arate fine-tuned models, even though they are ini tialized with the same pre-trained parameters. The question-answering example in Figure 1 will serve as a running example for this section. A distinctive feature of BERT is its unified ar chitecture across different tasks. There is mini mal difference between the pre-trained architec ture and the final downstream architecture. Model Architecture BERT's model architec ture is a multi-layer bidirectional Transformer en coder based on the original implementation de scribed in Vaswani et al. (2017) and released in the tensor2tensor library.1 Because the use of Transformers has become common and our im plementation is almost identical to the original, we will omit an exhaustive background descrip tion of the model architecture and refer readers to Vaswani et al. (2017) as well as excellent guides such as "The Annotated Transformer."2 In this work, we denote the number of layers (i.e., Transformer blocks) as L, the hidden size as H, and the number of self-attention heads as A. 3 We primarily report results on two model sizes: BERTBASE (L=12, H=768, A=12, Total Param eters=110M) and BERTLARGE (L=24, H=1024, A=16, Total Parameters=340M). BERTBASE was chosen to have the same model size as OpenAI GPT for comparison purposes. Critically, however, the BERT Transformer uses bidirectional self-attention, while the GPT Trans former uses constrained self-attention where every token can only attend to context to its left.4 1 https://github.com/tensorflow/tensor2tensor 2 http://nlp.seas.harvard.edu/2018/04/03/attention.html 3 In all cases we set the feed-forward/filter size to be 4H, i.e., 3072 for the H = 768 and 4096 for the H

```
1
2 # all pages
3
4
5
6
7
8
9
10 from IPython.display import Markdown, display
11
12 for i, d in enumerate(pdf, start=1):
13     display(Markdown(f"### Page {i}\n\n{d.page_content}"))
14
15
16
```

Show hidden output

Document Splitter

```
1 from langchain.text_splitter import RecursiveCharacterTextSplitter #CharacterTextSplitter
2
3
4 def chunk_data(docs, chunk_size=1000, chunk_overlap=200):
5     text_splitter = RecursiveCharacterTextSplitter(chunk_size=chunk_size,
6                                                    chunk_overlap=chunk_overlap)
```

```
    # each chunk will be a maximum of 1000 characters long# each chunk will share a 200-cl
    pdf = text_splitter.split_documents(docs)
    return pdf



# This code splits documents into chunks using the RecursiveCharacterTextSplitter class f

# A function named chunk_data is defined, which takes a document or a collection of docum
# chunk_size and chunk_overlap. chunk_size specifies the maximum number of characters in
# overlap between consecutive chunks.

# The function divides the documents into chunks based on these parameters using the Recu

# As a result, the documents are segmented into chunks of specified sizes, and these chun

# The chunk_overlap parameter is used to specify the sharing of characters between consec
# the end of one chunk reappear at the beginning of the next chunk. This prevents the los
```

```
pdf_doc = chunk_data(docs=pdf)

len(pdf_doc)
```

```
83
```

```
pdf_doc[0:2]
```

```
[Document(metadata={'producer': 'pdfTeX-1.40.18', 'creator': 'LaTeX with hyperref package', 'creationdate': '2019-
04-29T17:36:03+00:00', 'title': 'BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding',
'author': 'Jacob Devlin ; Ming-Wei Chang ; Kenton Lee ; Kristina Toutanova', 'subject': 'N19-1 2019', 'keywords':
'', 'moddate': '2019-04-29T17:36:03+00:00', 'source': '/content/drive/MyDrive/nlp/bert_article.pdf', 'total_pages':
16, 'page': 0}, page_content='Proceedings of NAACL-HLT 2019, pages 4171–4186\nMinneapolis, Minnesota, June 2 - June
7, 2019. \nc 2019 Association for Computational Linguistics\n4171\nBERT: Pre-training of Deep Bidirectional
Transformers for\nLanguage Understanding\nJacob Devlin Ming-Wei Chang Kenton Lee Kristina Toutanova\nGoogle AI
Language\n{jacobdevlin,mingweichang,kentonl,kristout}@google.com\nAbstract\nWe introduce a new language
representa\x02tion model called BERT, which stands for\nBidirectional Encoder Representations from\nTransformers.
Unlike recent language repre\x02sentation models (Peters et al., 2018a; Rad\x02ford et al., 2018), BERT is designed
to pre\x02train deep bidirectional representations from\nunlabeled text by jointly conditioning on both\nleft and
right context in all layers. As a re\x02sult, the pre-trained BERT model can be fine\x02tuned with just one
additional output layer\nto create state-of-the-art models for a wide\nrange of tasks, such as question answering
and'),
 Document(metadata={'producer': 'pdfTeX-1.40.18', 'creator': 'LaTeX with hyperref package', 'creationdate': '2019-
04-29T17:36:03+00:00', 'title': 'BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding',
'author': 'Jacob Devlin ; Ming-Wei Chang ; Kenton Lee ; Kristina Toutanova', 'subject': 'N19-1 2019', 'keywords':
'', 'moddate': '2019-04-29T17:36:03+00:00', 'source': '/content/drive/MyDrive/nlp/bert_article.pdf', 'total_pages':
16, 'page': 0}, page_content='to create state-of-the-art models for a wide\nrange of tasks, such as question
answering and\nlanguage inference, without substantial task\x02specific architecture modifications.\nBERT is
conceptually simple and empirically\npowerful. It obtains new state-of-the-art re\x02sults on eleven natural
language processing\ntasks, including pushing the GLUE score to\n80.5% (7.7% point absolute improvement),\nMultiNLI
accuracy to 86.7% (4.6% absolute\nimprovement), SQuAD v1.1 question answer\x02ing Test F1 to 93.2 (1.5 point
absolute im\x02provement) and SQuAD v2.0 Test F1 to 83.1\n(5.1 point absolute improvement).\n1
Introduction\nLanguage model pre-training has been shown to\nbe effective for improving many natural
language\nprocessing tasks (Dai and Le, 2015; Peters et al.,\n2018a; Radford et al., 2018; Howard and
Ruder,\n2018). These include sentence-level tasks such as\nnatural language inference (Bowman et al.,
2015;\nWilliams et al., 2018) and paraphrasing (Dolan')]
```

```
Markdown(pdf_doc[82].page_content)
```

MASK means that we replace the target token with the [MASK] symbol for MLM; SAME means that we keep the target token as is; RND means that we replace the target token with another random token. The numbers in the left part of the table repre sent the probabilities of the specific strategies used during MLM pre-training (BERT uses 80%, 10%, 10%). The right part of the paper represents the Dev set results. For the feature-based approach, we concatenate the last 4 layers of BERT as the features, which was shown to be the best approach in Section 5.3. From the table it can be seen that fine-tuning is surprisingly robust to different masking strategies. However, as expected, using only the MASK strat egy was problematic when applying the feature based approach to NER. Interestingly, using only the RND strategy performs much worse than our strategy as well.

1. Creating A Embedding Model

2. Convert the Each Chunk of The Split Document to Embedding Vectors

3. Storing of The Embedding Vectors to Vectorstore

4. Save the Vectorstore to Your Drive

```
from langchain_openai import OpenAIEmbeddings
# from langchain.embeddings import OpenAIEmbeddings


embeddings = OpenAIEmbeddings(model="text-embedding-3-large",
                              dimensions=3072)  # dimensions=256, 1024, 3072
embeddings

# As the embedding model, we use Openai's latest introduced text-embedding-3-large model.
# dimensions of text-embedding-3-large are 256, 1024 and 3072
```

```
OpenAIEmbeddings(client=<openai.resources.embeddings.Embeddings object at 0x7e260d54f800>, async_client=
<openai.resources.embeddings.AsyncEmbeddings object at 0x7e260908dfd0>, model='text-embedding-3-large',
dimensions=3072, deployment='text-embedding-ada-002', openai_api_version=None, openai_api_base=None,
openai_api_type=None, openai_proxy=None, embedding_ctx_length=8191, openai_api_key=SecretStr('**********'),
openai_organization=None, allowed_special=None, disallowed_special=None, chunk_size=1000, max_retries=2,
request_timeout=None, headers=None, tiktoken_enabled=True, tiktoken_model_name=None, show_progress_bar=False,
model_kwargs={}, skip_empty=False, default_headers=None, default_query=None, retry_min_seconds=4,
retry_max_seconds=20, http_client=None, http_async_client=None, check_embedding_ctx_length=True)
```

```
# the "Embed" step of the RAG (Retrieval Augmented Generation) pipeline.
# This is the third stage in the indexing process, after "Load" and "Split".
```

```
# store the embedding vectors.
```

```
from langchain_chroma import Chroma

# VectorStore object
# Embeds each Document in pdf_doc using the embeddings function. Stores vectors + metadata
index = Chroma.from_documents(documents=pdf_doc,
                              embedding=embeddings,
                              persist_directory="./chroma_store_bert_z")  # persist_direct

# Create a retriever object from our Chroma vector store (index). Wraps the vector store a
retriever = index.as_retriever()  # for using agents #index.as_retriever(search_kwargs={"k"



```

```
# refinement of the "Retrieval" step in a RAG system.
# to retrieve the four most relevant pieces of information from the indexed PDF

retriever = index.as_retriever(search_kwargs={"k": 4})

```

```
retriever.invoke("Describe BERT model architecture")
```

```
[Document(id='10ffc67d-9f78-408d-9d50-9acf62d59031', metadata={'subject': 'N19-1 2019', 'producer': 'pdfTeX-
1.40.18', 'creationdate': '2019-04-29T17:36:03+00:00', 'page': 2, 'author': 'Jacob Devlin ; Ming-Wei Chang ; Kento
Lee ; Kristina Toutanova', 'source': '/content/drive/MyDrive/nlp/bert_article.pdf', 'keywords': '', 'moddate':
'2019-04-29T17:36:03+00:00', 'total_pages': 16, 'creator': 'LaTeX with hyperref package', 'title': 'BERT: Pre-
training of Deep Bidirectional Transformers for Language Understanding'}, page_content='framework: pre-training an
fine-tuning. Dur\x02ing pre-training, the model is trained on unlabeled\ndata over different pre-training tasks.
For fine\x02tuning, the BERT model is first initialized with\nthe pre-trained parameters, and all of the
param\x02eters are fine-tuned using labeled data from the\ndownstream tasks. Each downstream task has sep\x02arate
fine-tuned models, even though they are ini\x02tialized with the same pre-trained parameters. The\nquestion-
answering example in Figure 1 will serve\nas a running example for this section.\nA distinctive feature of BERT is
its unified ar\x02chitecture across different tasks. There is mini\x02mal difference between the pre-trained
architec\x02ture and the final downstream architecture.\nModel Architecture BERT’s model architec\x02ture is a
multi-layer bidirectional Transformer en\x02coder based on the original implementation de\x02scribed in Vaswani et
al. (2017) and released in\nthe tensor2tensor library.1 Because the use'),
```

```
 Document(id='db0bae0a-9e10-424b-b65e-c29e35d64163', metadata={'creationdate': '2019-04-29T17:36:03+00:00',
'keywords': '', 'producer': 'pdfTeX-1.40.18', 'moddate': '2019-04-29T17:36:03+00:00', 'creator': 'LaTeX with
hyperref package', 'title': 'BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding',
'author': 'Jacob Devlin ; Ming-Wei Chang ; Kenton Lee ; Kristina Toutanova', 'source':
'/content/drive/MyDrive/nlp/bert_article.pdf', 'subject': 'N19-1 2019', 'page': 2, 'total_pages': 16},
page_content='4173\nBERT BERT\nE[CLS] E1\n E[SEP] ... EN\nE1’ ... EM’\nC T1 T[SEP] ... TN\nT1’ ... TM’\n[CLS] Tok
[SEP] ... Tok N Tok 1 ... TokM\nQuestion Paragraph\nStart/End Span\nBERT\nE[CLS] E1\n E[SEP] ... EN\nE1’ ... EM’\n
T1 T[SEP] ... TN\nT1’ ... TM’\n[CLS] Tok 1 [SEP] ... Tok N Tok 1 ... TokM\nMasked Sentence A Masked Sentence
B\nPre-training Fine-Tuning\nNSP Mask LM Mask LM\nUnlabeled Sentence A and B Pair \nSQuAD\nQuestion Answer
Pair\nMNLI NER\nFigure 1: Overall pre-training and fine-tuning procedures for BERT. Apart from output layers, the
same architec\x02tures are used in both pre-training and fine-tuning. The same pre-trained model parameters are
used to initialize\nmodels for different down-stream tasks. During fine-tuning, all parameters are fine-tuned.
[CLS] is a special\nsymbol added in front of every input example, and [SEP] is a special separator token (e.g.
separating ques\x02tions/answers).\ning and auto-encoder objectives have been used\nfor pre-training such models
(Howard and Ruder,'),
 Document(id='b94e7da0-751b-47e0-ab26-94e0e1289db6', metadata={'moddate': '2019-04-29T17:36:03+00:00', 'source':
'/content/drive/MyDrive/nlp/bert_article.pdf', 'total_pages': 16, 'author': 'Jacob Devlin ; Ming-Wei Chang ; Kento
Lee ; Kristina Toutanova', 'keywords': '', 'creationdate': '2019-04-29T17:36:03+00:00', 'creator': 'LaTeX with
hyperref package', 'subject': 'N19-1 2019', 'title': 'BERT: Pre-training of Deep Bidirectional Transformers for
Language Understanding', 'producer': 'pdfTeX-1.40.18', 'page': 0}, page_content='to create state-of-the-art models
for a wide\nrange of tasks, such as question answering and\nlanguage inference, without substantial
task\x02specific architecture modifications.\nBERT is conceptually simple and empirically\npowerful. It obtains ne
state-of-the-art re\x02sults on eleven natural language processing\ntasks, including pushing the GLUE score
to\n80.5% (7.7% point absolute improvement),\nMultiNLI accuracy to 86.7% (4.6% absolute\nimprovement), SQuAD v1.1
question answer\x02ing Test F1 to 93.2 (1.5 point absolute im\x02provement) and SQuAD v2.0 Test F1 to 83.1\n(5.1
point absolute improvement).\n1 Introduction\nLanguage model pre-training has been shown to\nbe effective for
improving many natural language\nprocessing tasks (Dai and Le, 2015; Peters et al.,\n2018a; Radford et al., 2018;
Howard and Ruder,\n2018). These include sentence-level tasks such as\nnatural language inference (Bowman et al.,
2015;\nWilliams et al., 2018) and paraphrasing (Dolan'),
 Document(id='01826491-be9a-4ad6-8e8b-d51e7bd6fa6f', metadata={'total_pages': 16, 'moddate': '2019-04-
29T17:36:03+00:00', 'creationdate': '2019-04-29T17:36:03+00:00', 'keywords': '', 'source':
'/content/drive/MyDrive/nlp/bert_article.pdf', 'page': 0, 'subject': 'N19-1 2019', 'author': 'Jacob Devlin ; Ming-
Wei Chang ; Kenton Lee ; Kristina Toutanova', 'producer': 'pdfTeX-1.40.18', 'creator': 'LaTeX with hyperref
package', 'title': 'BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding'},
page_content='example, in OpenAI GPT, the authors use a left-to\x02right architecture, where every token can only
at\x02tend to previous tokens in the self-attention layers\nof the Transformer (Vaswani et al., 2017). Such
re\x02strictions are sub-optimal for sentence-level tasks,\nand could be very harmful when applying fine\x02tuning
based approaches to token-level tasks such\nas question answering, where it is crucial to incor\x02porate context
from both directions.\nIn this paper, we improve the fine-tuning based\napproaches by proposing BERT:
Bidirectional\nEncoder Representations from Transformers.\nBERT alleviates the previously mentioned
unidi\x02rectionality constraint by using a “masked lan\x02guage model” (MLM) pre-training objective, in\x02spired
by the Cloze task (Taylor, 1953). The\nmasked language model randomly masks some of\nthe tokens from the input, an
the objective is to\npredict the original vocabulary id of the masked')]
```

Load Vectorstore(index) From Your Drive

```
# Initiates the Chroma class, which serves as our vector database.

loaded_from_drive_index = Chroma(persist_directory="/content/drive/MyDrive/nlp/bert_articl
                        embedding_function=embeddings)
#/chroma_store_bert_z
```

```
load_retriver = loaded_from_drive_index.as_retriever(search_kwargs={"k": 5})
load_retriver
```

```
VectorStoreRetriever(tags=['Chroma', 'OpenAIEmbeddings'], vectorstore=<langchain_chroma.vectorstores.Chroma object
at 0x7e260cfa34d0>, search_kwargs={'k': 5})
```

```

# bonus ... shorter versions of above


def retrieve_query(query, k=4):
    retriever = index.as_retriever(search_kwargs={"k": k})  # loaded_index
    return retriever.invoke(query)

# The query we ask is first converted into an embedding vector. Then, using the cosine sim
```

```
our_query = "Describe BERT model architecture"

doc_search = retrieve_query(our_query, k=5)  # first 5 most similar texts are returned
doc_search
```

```
[Document(id='10ffc67d-9f78-408d-9d50-9acf62d59031', metadata={'keywords': '', 'creator': 'LaTeX with hyperref
package', 'moddate': '2019-04-29T17:36:03+00:00', 'subject': 'N19-1 2019', 'total_pages': 16, 'creationdate':
'2019-04-29T17:36:03+00:00', 'producer': 'pdfTeX-1.40.18', 'page': 2, 'source':
```

```
'/content/drive/MyDrive/nlp/bert_article.pdf', 'author': 'Jacob Devlin ; Ming-Wei Chang ; Kenton Lee ; Kristina
Toutanova', 'title': 'BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding'},
page_content='framework: pre-training and fine-tuning. Dur\x02ing pre-training, the model is trained on
unlabeled\ndata over different pre-training tasks. For fine\x02tuning, the BERT model is first initialized
with\nthe pre-trained parameters, and all of the param\x02eters are fine-tuned using labeled data from
the\ndownstream tasks. Each downstream task has sep\x02arate fine-tuned models, even though they are
ini\x02tialized with the same pre-trained parameters. The\nquestion-answering example in Figure 1 will serve\nas
a running example for this section.\nA distinctive feature of BERT is its unified ar\x02chitecture across
different tasks. There is mini\x02mal difference between the pre-trained architec\x02ture and the final
downstream architecture.\nModel Architecture BERT's model architec\x02ture is a multi-layer bidirectional
Transformer en\x02coder based on the original implementation de\x02scribed in Vaswani et al. (2017) and released
in\nthe tensor2tensor library.1 Because the use'),
 Document(id='db0bae0a-9e10-424b-b65e-c29e35d64163', metadata={'page': 2, 'total_pages': 16, 'source':
'/content/drive/MyDrive/nlp/bert_article.pdf', 'keywords': '', 'author': 'Jacob Devlin ; Ming-Wei Chang ; Kenton
Lee ; Kristina Toutanova', 'creationdate': '2019-04-29T17:36:03+00:00', 'subject': 'N19-1 2019', 'title': 'BERT:
Pre-training of Deep Bidirectional Transformers for Language Understanding', 'creator': 'LaTeX with hyperref
package', 'producer': 'pdfTeX-1.40.18', 'moddate': '2019-04-29T17:36:03+00:00'}, page_content='4173\nBERT
BERT\nE[CLS] E1\n E[SP] ... EN\nE1' ... EM'\nC T1 T[SP] ... TN\nT1' ... TM'\n[CLS] Tok 1 [SP] ... Tok N Tok 1
... TokM\nQuestion Paragraph\nStart/End Span\nBERT\nE[CLS] E1\n E[SP] ... EN\nE1' ... EM'\nC T1 T[SP] ...
TN\nT1' ... TM'\n[CLS] Tok 1 [SP] ... Tok N Tok 1 ... TokM\nMasked Sentence A Masked Sentence B\nPre-training
Fine-Tuning\nNSP Mask LM Mask LM\nUnlabeled Sentence A and B Pair \nSQuAD\nQuestion Answer Pair\nMNLI NER\nFigure
1: Overall pre-training and fine-tuning procedures for BERT. Apart from output layers, the same architec\x02tures
are used in both pre-training and fine-tuning. The same pre-trained model parameters are used to
initialize\nmodels for different down-stream tasks. During fine-tuning, all parameters are fine-tuned. [CLS] is a
special\nsymbol added in front of every input example, and [SEP] is a special separator token (e.g. separating
ques\x02tions/answers).\ning and auto-encoder objectives have been used\nfor pre-training such models (Howard and
Ruder,'),
 Document(id='b94e7da0-751b-47e0-ab26-94e0e1289db6', metadata={'creationdate': '2019-04-29T17:36:03+00:00',
'title': 'BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding', 'total_pages': 16,
'moddate': '2019-04-29T17:36:03+00:00', 'producer': 'pdfTeX-1.40.18', 'subject': 'N19-1 2019', 'keywords': '',
'page': 0, 'source': '/content/drive/MyDrive/nlp/bert_article.pdf', 'author': 'Jacob Devlin ; Ming-Wei Chang ;
Kenton Lee ; Kristina Toutanova', 'creator': 'LaTeX with hyperref package'}, page_content='to create state-of-
the-art models for a wide\nrange of tasks, such as question answering and\nlanguage inference, without
substantial task\x02specific architecture modifications.\nBERT is conceptually simple and empirically\npowerful.
It obtains new state-of-the-art re\x02sults on eleven natural language processing\ntasks, including pushing the
GLUE score to\n80.5% (7.7% point absolute improvement),\nMultiNLI accuracy to 86.7% (4.6% absolute\nimprovement),
SQuAD v1.1 question answer\x02ing Test F1 to 93.2 (1.5 point absolute im\x02provement) and SQuAD v2.0 Test F1 to
83.1\n(5.1 point absolute improvement).\n1 Introduction\nLanguage model pre-training has been shown to\nbe
effective for improving many natural language\nprocessing tasks (Dai and Le, 2015; Peters et al.,\n2018a; Radford
et al., 2018; Howard and Ruder,\n2018). These include sentence-level tasks such as\nnatural language inference
(Bowman et al., 2015;\nWilliams et al., 2018) and paraphrasing (Dolan'),
 Document(id='01826491-be9a-4ad6-8e8b-d51e7bd6fa6f', metadata={'producer': 'pdfTeX-1.40.18', 'keywords': '',
'source': '/content/drive/MyDrive/nlp/bert_article.pdf', 'total_pages': 16, 'title': 'BERT: Pre-training of Deep
Bidirectional Transformers for Language Understanding', 'creator': 'LaTeX with hyperref package', 'moddate':
'2019-04-29T17:36:03+00:00', 'author': 'Jacob Devlin ; Ming-Wei Chang ; Kenton Lee ; Kristina Toutanova', 'page':
0, 'creationdate': '2019-04-29T17:36:03+00:00', 'subject': 'N19-1 2019'}, page_content='example, in OpenAI GPT,
the authors use a left-to\x02right architecture, where every token can only at\x02tend to previous tokens in the
self-attention layers\nof the Transformer (Vaswani et al., 2017). Such re\x02strictions are sub-optimal for
sentence-level tasks,\nand could be very harmful when applying fine\x02tuning based approaches to token-level
tasks such\nas question answering, where it is crucial to incor\x02porate context from both directions.\nIn this
paper, we improve the fine-tuning based\napproaches by proposing BERT: Bidirectional\nEncoder Representations
from Transformers.\nBERT alleviates the previously mentioned unidi\x02rectionality constraint by using a "masked
lan\x02guage model" (MLM) pre-training objective, in\x02spired by the Cloze task (Taylor, 1953). The\nmasked
language model randomly masks some of\nthe tokens from the input, and the objective is to\npredict the original
vocabulary id of the masked'),
```

```
1
2 # for now we only saw our pdf relevant parts, llm was not involved in ...
3
```

Generating an Answer Based on The Similar Chunks

```
1 # vital bridge between the "Retrieval" and "Generation" phases of our RAG pipeline
2 # relevant information retrieved from our vector store (doc_search) and formats it, along w
```

Pipeline For RAG

```
1   # 1. chatbot qa yapisi prompt kismi
2
3
4 from langchain.prompts import ChatPromptTemplate
5 from langchain_openai import ChatOpenAI
6 from langchain_core.output_parsers import StrOutputParser
7 from langchain.chains.question_answering import load_qa_chain
8 from IPython.display import Markdown
9
10 # Role-based chat prompt
11 chat_template = ChatPromptTemplate.from_messages([
```

```
    (
        "system",
        "You are a scientific QA assistant. Use only the provided context. "
        "Always cite page numbers. If uncertain, say 'I don't know.' "
        "Keep answers short, neutral, and evidence-based."
    ),
    (
        "system",
        "Return output as a markdown block with sections:\n"
        "# Answer\n"
        "## Key Points (with page cites)\n"
        "### Sources\n"
        "### Confidence"
    ),
    (
        "human",
        # We inject both the raw context and a compact citations map the model can quote.
        "Context:\n{context}\n\n"
        "Citations Map (use these for page refs):\n{citations}\n\n"
        "Question: {question}"
    )
])

# For map_reduce you also need a combine prompt (roles preserved)
combine_prompt = ChatPromptTemplate.from_messages([
    ("system",
     "You are a scientific QA assistant. Combine the partial answers below into a single,
     "Resolve conflicts conservatively; if evidence is insufficient, say 'I don't know.' '
     "Keep the tone neutral and cite page numbers where available."),
    ("system",
     "Return markdown with sections:\n"
     "### Answer\n"
     "### Key Points (with page cites)\n"
     "### Sources\n"
     "### Confidence"),
    ("human",
     "Partial answers from multiple chunks:\n{summaries}\n\nProduce the final answer.")
])



            # 1. chatbot qa structure


#get_answers, consolidates all the previous steps of the Retrieval Augmented Generation (I

#This function represents the complete workflow from receiving a user's question to generi


def get_answers(query, k=4):
    from langchain_openai import ChatOpenAI
    from langchain_core.output_parsers import StrOutputParser
    from IPython.display import Markdown

    # 1) Retrieve top-k documents
    docs = retriever.invoke(query, k=k)

    # 2) Build context
    context = "\n\n".join(d.page_content for d in docs)

    # 3) Build a citations map the model can quote in output
    #    (ensures it actually has page numbers available)
    citations = "\n".join(
        f"[{i+1}] {d.metadata.get('source','unknown')} — p.{d.metadata.get('page','?')}"
        for i, d in enumerate(docs)
    )

    # 4) LLM
    llm = ChatOpenAI(model_name="gpt-4o-mini", temperature=0, max_tokens=256)
    # max_tokens=1024)

```

```
    # 5) Chain
    chain = chat_template | llm | StrOutputParser()

    # 6) Invoke
    answer = chain.invoke({"question": query, "context": context, "citations": citations}

    # 7) Return (avoid adding another Sources block — model already includes it)
```

```
our_query = "What are the two pre-training tasks used in BERT, and what does each one trai

answer = get_answers(our_query, k=5)
answer
```

Answer

The two pre-training tasks used in BERT are:

1. **Masked Language Model (MLM)**: This task involves masking a percentage of the input tokens and training the model to predict these masked tokens based on their context. It allows the model to learn bidirectional representations by considering both left and right context.
2. **Next Sentence Prediction (NSP)**: This task trains the model to predict whether a given sentence B follows sentence A in the original text. It helps the model understand the relationship between sentence pairs.

Key Points (with page cites)

- BERT uses two unsupervised tasks for pre-training: Masked Language Model (MLM) and Next Sentence Prediction (NSP) (p.3).
- MLM trains the model to predict masked tokens in a sentence (p.3).
- NSP trains the model to determine if one sentence follows another (p.3).

Sources

- [1] /content/drive/MyDrive/nlp/bert_article.pdf — p.3

Confidence

High

```
our_query = "What ROUGE-L score does BERT report on CNN/DailyMail abstractive summarizatio

answer = get_answers(our_query, k=5)
answer
```

I don't know.

```
our_query = "In the context of fine-tuning, what minimal number of parameters need to be r

answer = get_answers(our_query, k=5)
answer


```

Answer

The minimal number of parameters that need to be randomly initialized when incorporating BERT with an extra output layer is very small, as the task-specific models are formed by adding only one additional output layer to the pre-trained model.

Key Points (with page cites)

- The task-specific models incorporate BERT with one additional output layer, requiring only a minimal number of parameters to be learned from scratch (page 4184).

Sources

- [1] /content/drive/MyDrive/nlp/bert_article.pdf — p.8
- [2] /content/drive/MyDrive/nlp/bert_article.pdf — p.12
- [3] /content/drive/MyDrive/nlp/bert_article.pdf — p.13
- [4] /content/drive/MyDrive/nlp/bert_article.pdf — p.13
- [5] /content/drive/MyDrive/nlp/bert_article.pdf — p.2

Confidence

High

```
                          # some question
# Relevant

# 1) What are the two pre-training tasks used in BERT, and what does each one train the mo
# 2) How do BERTBASE and BERTLARGE differ in L (layers), H (hidden size), A (attention hea


# Irrelevant

# 3) What is the chemical formula of water?
# 4) What is the tallest mountain in South America?
# 5) What ROUGE-L score does BERT report on CNN/DailyMail abstractive summarization using a
```

```
#                        works well  but may not be cost effective

from langchain.chains import RetrievalQA
from langchain_core.prompts import ChatPromptTemplate
from langchain_openai import ChatOpenAI

llm = ChatOpenAI(model="gpt-4o-mini", temperature=0)


QUESTION_PROMPT = ChatPromptTemplate.from_template(
    """You are an data scientist assistant answering a user question using ONLY the provi

REQUIREMENTS
- If the context is insufficient, say: "I don't know based on the provided context."
- Be precise and neutral; do not speculate.
- Keep the main answer within 3–8 sentences unless explicitly asked for more.
- Cite support in brackets using metadata if available.

CONTEXT
{context}

QUESTION
{question}

ANSWER"""
)

COMBINE_PROMPT = ChatPromptTemplate.from_template(
    """You receive multiple candidate answers synthesized from different context chunks.

TASK
- Produce ONE final answer.
- Prefer statements directly supported by the context; drop anything weakly supported.
- If candidates conflict and evidence is unclear, state the uncertainty or contraversarie
- Keep the main answer within 5–12 sentences.
- Consolidate citations in brackets if metadata is present.

CANDIDATE ANSWERS
{summaries}

FINAL ANSWER"""
)

# RetrievalQA icinde retriever en alakalı döküman parçalarını çeker,seçtiğin chain_type'a

def get_answers(query, k=5):
    retriever = index.as_retriever(search_kwargs={"k": k})
    qa = RetrievalQA.from_chain_type(                        ####################
        llm=llm,
        retriever=retriever,
        chain_type="map_reduce",   # or "stuff" for simpler
```

```
        chain_type_kwargs={
            "question_prompt": QUESTION_PROMPT,
            "combine_prompt": COMBINE_PROMPT
        },
        return_source_documents=True,
    )
    out = qa.invoke({"query": query})   # in LC 0.2+, key is "query"
    return out['result']
```

```
our_query = " What are the two pre-training tasks used in BERT, and what does each one tra

answer = get_answers(our_query, k=7)
answer
```

```
'The two pre-training tasks used in BERT are the Masked Language Model (MLM) and Next Sentence Prediction (NSP). Th
e MLM involves masking a percentage of the input tokens and training the model to predict these masked tokens based
on their context, allowing the model to learn deep bidirectional representations. The NSP task trains the model to
determine if a given sentence logically follows another, which helps in understanding the relationship between sent
```

```
Markdown(answer)
```

The two pre-training tasks used in BERT are the Masked Language Model (MLM) and Next Sentence Prediction (NSP). The MLM involves masking a percentage of the input tokens and training the model to predict these masked tokens based on their context, allowing the model to learn deep bidirectional representations. The NSP task trains the model to determine if a given sentence logically follows another, which helps in understanding the relationship between sentences and is crucial for tasks requiring context comprehension across multiple sentences [Figure 1].

```
our_query = "what is the relation between bert and tpu?"

answer = get_answers(our_query, k=5)
Markdown(answer)
```

Based on the provided context, there is insufficient information to formulate a definitive answer.

```

our_query = "In the context of fine-tuning, what minimal number of parameters need to be r

answer = get_answers(our_query, k=5)
Markdown(answer)

```

In the context of fine-tuning BERT with an additional output layer, the minimal number of parameters that need to be learned from scratch is the number of parameters in the new output layer itself. All other parameters of the pre-trained BERT model are typically retained and fine-tuned rather than initialized randomly. Therefore, only the parameters specific to the additional output layer would be initialized randomly.

```
def chat(k=5):
    from IPython.display import display
    while True:
        q = input("Q (blank/quit to exit): ").strip()
        if not q or q.lower() in {"q", "quit", "exit"}:
            break
        display(get_answers(q, k=k))
        print("-" * 80)
```

```
chat()
```

```
Q (blank/quit to exit): q
```

```
Start coding or generate with AI.
```

```
Start coding or generate with AI.
```