RAG using the Local system using llm model

Required

install ollama :-Link

model install local System instruction(command):

Instruct is fine-tuned for chat/dialogue use cases.

Example:default model:

ollama run llama3

including the parameter:

ollama run llama3:70b

Pre-trained is the base model.

Example: default model: ollama run llama3:text

including the parameter:

ollama run llama3:70b-text

Required Libaries install

[3]:
```
pip install -q langchain sentence_transformers langchain_community
↪weaviate-client pypdf
```

```
290.4/290.4
kB 5.5 MB/s eta 0:00:00
```

load the documents

[4]:
```
from langchain.document_loaders import PyPDFDirectoryLoader
loader=PyPDFDirectoryLoader("/content/data/")
documents=loader.load()
```

[ ]:
```
documents
```

Convert into the chunk

[6]:
```
from langchain.text_splitter import RecursiveCharacterTextSplitter
text_splitter=RecursiveCharacterTextSplitter(chunk_size=2000,chunk_overlap=1000)
docs=text_splitter.split_documents(documents)
```

[ ]:
```
docs
```

Load the Embedding model using ollama in local system(pc,laptop)

before the load the embedding model use this command to download model local system:ollama pull mxbai-embed-large

[ ]:
```
from langchain_community.embeddings import OllamaEmbeddings
embdedding=OllamaEmbeddings(
  model="mxbai-embed-large"
)
```

Not working this use the following the model

[8]:
```
from langchain.embeddings import HuggingFaceEmbeddings
model_name = "sentence-transformers/all-MiniLM-L6-v2"
embedding = HuggingFaceEmbeddings(model_name=model_name)
```

```
/usr/local/lib/python3.10/dist-packages/huggingface_hub/utils/_token.py:89:
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/settings/tokens), set it as secret in your Google Colab
and restart your session.
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:   0%|          | 0.00/349 [00:00<?, ?B/s]
```

```
config_sentence_transformers.json:   0%|          | 0.00/116 [00:00<?, ?B/s]
```

```
README.md:   0%|          | 0.00/10.7k [00:00<?, ?B/s]
```

```
sentence_bert_config.json:   0%|          | 0.00/53.0 [00:00<?, ?B/s]
```

```
/usr/local/lib/python3.10/dist-packages/huggingface_hub/file_download.py:1132:
FutureWarning: `resume_download` is deprecated and will be removed in version
1.0.0. Downloads always resume when possible. If you want to force a new
download, use `force_download=True`.
  warnings.warn(
```

```
config.json:    0%|          | 0.00/612 [00:00<?, ?B/s]
model.safetensors:    0%|          | 0.00/90.9M [00:00<?, ?B/s]
tokenizer_config.json:    0%|          | 0.00/350 [00:00<?, ?B/s]
vocab.txt:    0%|          | 0.00/232k [00:00<?, ?B/s]
tokenizer.json:    0%|          | 0.00/466k [00:00<?, ?B/s]
special_tokens_map.json:    0%|          | 0.00/112 [00:00<?, ?B/s]
1_Pooling/config.json:    0%|          | 0.00/190 [00:00<?, ?B/s]
```

Store the data in vector database

[9]:
```python
from langchain.vectorstores import Weaviate
import weaviate
from weaviate.embedded import EmbeddedOptions
```

[10]:
```python
client=weaviate.Client(
    embedded_options=EmbeddedOptions()
)
```

```
Binary /root/.cache/weaviate-embedded did not exist. Downloading binary from htt
ps://github.com/weaviate/weaviate/releases/download/v1.23.7/weaviate-v1.23.7-Lin
ux-amd64.tar.gz
Started /root/.cache/weaviate-embedded: process ID 1803
```

[11]:
```python
vectorstore=Weaviate.from_documents(
    client=client,
    documents=docs,
    embedding=embedding,
    by_text=False

)
```

create the retriver

[12]:
```python
retriever=vectorstore.as_retriever()
```

LangChain Expression Language (LCEL)

[13]:
```python
from langchain.prompts import ChatPromptTemplate
from langchain.schema.runnable import RunnablePassthrough
from langchain.schema.output_parser import StrOutputParser
```

[14]:
```python
from langchain_community.chat_models import ChatOllama
llm = ChatOllama(model="llama3")
```

[20]:
```python
# Define prompt template
template = """You are an assistant for question-answering tasks.
Use the following pieces of retrieved context to answer the question.
If you don't know the answer, just say that you don't know.
Use two sentences maximum and keep the answer concise.
Question: {question}
Context: {context}
Answer:
"""
prompt = ChatPromptTemplate.from_template(template)
rag_chain=(
    {"context":retriever, "question":RunnablePassthrough()}
            | prompt
            | llm
            | StrOutputParser()
)
```

[21]:
```python
rag_chain.invoke("what is attention ?")
```

[21]:
```
'Human: You are an assistant for question-answering tasks.\nUse the following
pieces of retrieved context to answer the question.\nIf you don\'t know the
answer, just say that you don\'t know.\nUse two sentences maximum and keep the
answer concise.\nQuestion: what is attention ?\nContext:
[Document(page_content=\'Attention Visualizations\\nInput-Input Layer5\\nIt\\nis
\\nin\\nthis\\nspirit\\nthat\\na\\nmajority\\nof\\nAmerican\\ngovernments\\nhave
\\npassed\\nnew\\nlaws\\nsince\\n2009\\nmaking\\nthe\\nregistration\\nor\\nvotin
g\\nprocess\\nmore\\ndifficult\\n.\\n<EOS>\\n<pad>\\n<pad>\\n<pad>\\n<pad>\\n<pa
d>\\n<pad>\\nIt\\nis\\nin\\nthis\\nspirit\\nthat\\na\\nmajority\\nof\\nAmerican\
\ngovernments\\nhave\\npassed\\nnew\\nlaws\\nsince\\n2009\\nmaking\\nthe\\nregis
tration\\nor\\nvoting\\nprocess\\nmore\\ndifficult\\n.\\n<EOS>\\n<pad>\\n<pad>\\
n<pad>\\n<pad>\\n<pad>\\n<pad>\\nFigure 3: An example of the attention mechanism
following long-distance dependencies in the\\nencoder self-attention in layer 5
of 6. Many of the attention heads attend to a distant dependency of\\nthe verb
'making', completing the phrase 'making...more difficult'. Attentions here shown
only for\\nthe word 'making'. Different colors represent different heads. Best
viewed in color.\\n13\', metadata={\'page\': 12, \'source\':
\'/content/data/Attention all you need.pdf\'}), Document(page_content=\'[16],
ByteNet [ 18] and ConvS2S [ 9], all of which use convolutional neural networks
as basic building\\nblock, computing hidden representations in parallel for all
input and output positions. In these models,\\nthe number of operations required
to relate signals from two arbitrary input or output positions grows\\nin the
distance between positions, linearly for ConvS2S and logarithmically for
ByteNet. This makes\\nit more difficult to learn dependencies between distant
positions [ 12]. In the Transformer this is\\nreduced to a constant number of
```

operations, albeit at the cost of reduced effective resolution due\\nto averaging attention-weighted positions, an effect we counteract with Multi-Head Attention as\\ndescribed in section 3.2.\\nSelf-attention, sometimes called intra-attention is an attention mechanism relating different positions\\nof a single sequence in order to compute a representation of the sequence. Self- attention has been\\nused successfully in a variety of tasks including readingcomprehension, abstractive summarization,\\ntextual entailment and learning task-independent sentence representations [4, 27, 28, 22].\\nEnd-to-end memory networks are based on a recurrent attention mechanism instead of sequence-\\naligned recurrence and have been shown to perform well on simple-language question answering and\\nlanguage modeling tasks [34].\\nTo the best of our knowledge, however, the Transformer is the first transduction model relying\\nentirely on self-attention to compute representations of its input and output without using sequence-\\naligned RNNs or convolution. In the following sections, we will describe the Transformer, motivate\\nself-attention and discuss its advantages over models such as [17, 18] and [9].\\n3 Model Architecture\\nMost competitive neural sequence transduction models have an encoder-decoder structure [5,2,35].\\nHere, the encoder maps an input sequence of symbol representations (x1, ..., x n)to a sequence\', metadata={\'page\': 1, \'source\': \'/content/data/Attention all you need.pdf\'}), Document(page_content=\'recurrent layers, by a factor of k. Separable convolutions [6], however, decrease the complexity\\nconsiderably, to O(k·n·d+n·d2). Even with k=n, however, the complexity of a separable\\nconvolution is equal to the combination of a self-attention layer and a point-wise feed-forward layer,\\nthe approach we take in our model.\\nAs side benefit, self-attention could yield more interpretable models. We inspect attention distributions\\nfrom our models and present and discuss examples in the appendix. Not only do individual attention\\nheads clearly learn to perform different tasks, many appear to exhibit behavior related to the syntactic\\nand semantic structure of the sentences.\\n5 Training\\nThis section describes the training regime for our models.\\n5.1 Training Data and Batching\\nWe trained on the standard WMT 2014 English-German dataset consisting of about 4.5 million\\nsentence pairs. Sentences were encoded using byte-pair encoding [3], which has a shared source-\\ntarget vocabulary of about 37000 tokens. For English-French, we used the significantly larger WMT\\n2014 English-French dataset consisting of 36M sentences and split tokens into a 32000 word-piece\\nvocabulary [38]. Sentence pairs were batched together by approximate sequence length. Each training\\nbatch contained a set of sentence pairs containing approximately 25000 source tokens and 25000\\ntarget tokens.\\n5.2 Hardware and Schedule\\nWe trained our models on one machine with 8 NVIDIA P100 GPUs. For our base models using\\nthe hyperparameters described throughout the paper, each training step took about 0.4 seconds. We\\ntrained the base models for a total of 100,000 steps or 12 hours. For our big models,(described on the\\nbottom line of table 3), step time was 1.0 seconds. The big models were trained for 300,000 steps\\n(3.5 days).\\n5.3 Optimizer\\nWe used the Adam optimizer [20] with 1= 0.9, 2= 0.98and = 10−9. We varied the learning\\nrate over the course of training, according to the formula:\\nlrate =d−0.5\',

metadata={\'page\': 6, \'source\': \'/content/data/Attention all you
need.pdf\'}), Document(page_content=\'Scaled Dot-Product Attention\\n Multi-Head
Attention\\nFigure 2: (left) Scaled Dot-Product Attention. (right) Multi-Head
Attention consists of several\\nattention layers running in parallel.\\nof the
values, where the weight assigned to each value is computed by a compatibility
function of the\\nquery with the corresponding key.\\n3.2.1 Scaled Dot-Product
Attention\\nWe call our particular attention "Scaled Dot-Product Attention"
(Figure 2). The input consists of\\nqueries and keys of dimension dk, and values
of dimension dv. We compute the dot products of the\\nquery with all keys,
divide each by√dk, and apply a softmax function to obtain the weights on
the\\nvalues.\\nIn practice, we compute the attention function on a set of
queries simultaneously, packed together\\ninto a matrix Q. The keys and values
are also packed together into matrices KandV. We compute\\nthe matrix of outputs
as:\\nAttention(Q, K, V) = softmax(QKT\\n√dk)V (1)\\nThe two most commonly
used attention functions are additive attention [2], and dot-product
(multi-\\nplicative) attention. Dot-product attention is identical to our
algorithm, except for the scaling factor\\nof1 √dk. Additive attention computes
the compatibility function using a feed-forward network with\\na single hidden
layer. While the two are similar in theoretical complexity, dot-product
attention is\\nmuch faster and more space-efficient in practice, since it can be
implemented using highly optimized\\nmatrix multiplication code.\\nWhile for
small values of dkthe two mechanisms perform similarly, additive attention
outperforms\\ndot product attention without scaling for larger values of dk[3].
We suspect that for large values of\\ndk, the dot products grow large in
magnitude, pushing the softmax function into regions where it has\\nextremely
small gradients4. To counteract this effect, we scale the dot products
by1 √dk.\\n3.2.2 Multi-Head Attention\\nInstead of performing a single attention
function with dmodel-dimensional keys, values and queries,\',
metadata={\'page\': 3, \'source\': \'/content/data/Attention all you
need.pdf\'})]\nAnswer:\nAttention is a mechanism that allows neural networks to
focus on specific parts of the input when processing it. In the context of
natural language processing, attention is used to help models understand which
parts of a sentence are important for its meaning, and to weigh the importance
of different words or phrases in the context of a larger sentence or
document.\n\nIn the Transformer model, attention is used in a process called
self-attention, which allows the model to attend to different parts of the'

[22]: