Retrieval Augmented Generation (RAG) with Langchain

Using IBM Granite Models

In this notebook

This notebook contains instructions for performing Retrieval Augumented Generation (RAG). RAG is an architectural pattern that can be used to augment the performance of language models by recalling factual information from a knowledge base, and adding that information to the model query. The most common approach in RAG is to create dense vector representations of the knowledge base in order to retrieve text chunks that are semantically similar to a given user query.

RAG use cases include:

- Customer service: Answering questions about a product or service using facts from the product documentation.
- Domain knowledge: Exploring a specialized domain (e.g., finance) using facts from papers or articles in the knowledge base.
- News chat: Chatting about current events by calling up relevant recent news articles.

In its simplest form, RAG requires 3 steps:

- Initial setup:
  - Index knowledge-base passages for efficient retrieval. In this recipe, we take embeddings of the passages, and store them in a vector database.
- Upon each user query:
  - Retrieve relevant passages from the database. In this recipe, we use an embedding of the query to retrieve semantically similar passages.
  - Generate a response by feeding retrieved passage into a large language model, along with the user query.

Setting up the environment

Ensure you are running python 3.10, 3.11, or 3.12 in a freshly-created virtual environment.

```
import sys
assert sys.version_info >= (3, 10) and sys.version_info < (3, 13), "Use Python 3.10, 3.11, or 3.12
```

Install dependencies

Granite utils provides some helpful functions for recipes.

```
%pip install git+https://github.com/ibm-granite-community/utils \
    transformers \
    langchain_community \
    'langchain_huggingface[full]' \
    langchain_milvus \
    replicate \
    wget
```

```
ty-utils, wget
.toml) ... done
bm_granite_community_utils-0.1.dev81-py3-none-any.whl size=12941 sha256=12ef081601bce35002493
wheels/7b/e8/64/5f8c4595097153effea0a6587a85a9ebf314924faf4e61c449

l size=9655 sha256=391b0a4642251c49d4f9dfd1ebafe1a4dd0e4ba0fd391a9a4d86cf64d6db613e
dbd1c6861731779f62cc4babcb234387e11d697df70ee97

nvidia-nvjitlink-cu12, nvidia-curand-cu12, nvidia-cufft-cu12, nvidia-cuda-runtime-cu12, nvidi

.82

2

82



1



2.5.82

5.82

5.82

82

5.82

82

2







1.3

3





3.83

83
to account all the packages that are installed. This behaviour is the source of the following
grpcio 1.67.1 which is incompatible.
 httpx-sse-0.4.1 ibm-granite-community-utils-0.1.dev81 langchain_community-0.3.27 langchain_h
```

Selecting System Components

Choose your Embeddings Model

Specify the model to use for generating embedding vectors from text.

To use a model from a provider other than Huggingface, replace this code cell with one from this Embeddings Model recipe.

```
from langchain_huggingface import HuggingFaceEmbeddings
from transformers import AutoTokenizer

embeddings_model_path = "ibm-granite/granite-embedding-30m-english"
embeddings_model = HuggingFaceEmbeddings(
    model_name=embeddings_model_path,
)
embeddings_tokenizer = AutoTokenizer.from_pretrained(embeddings_model_path)
```

```
/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://huggin
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 d
  warnings.warn(
modules.json: 100%                                            350/350 [00:00<00:00, 28.0kB/s]

README.md:      467k/? [00:00<00:00, 12.3MB/s]

sentence_bert_config.json: 100%                                        54.0/54.0 [00:00<00:00, 5.17kB/s]

config.json: 100%                                             683/683 [00:00<00:00, 57.6kB/s]

model.safetensors: 100%                                          60.6M/60.6M [00:08<00:00, 7.28MB/s]

tokenizer_config.json:      1.09k/? [00:00<00:00, 94.8kB/s]

vocab.json:      798k/? [00:00<00:00, 11.1MB/s]

merges.txt:      456k/? [00:00<00:00, 19.0MB/s]

tokenizer.json:      1.36M/? [00:00<00:00, 37.6MB/s]

special_tokens_map.json: 100%                                         239/239 [00:00<00:00, 15.0kB/s]

config.json: 100%                                             191/191 [00:00<00:00, 17.5kB/s]
```

Choose your Vector Database

Specify the database to use for storing and retrieving embedding vectors.

To connect to a vector database other than Milvus substitute this code cell with one from this Vector Store recipe.

```
from langchain_milvus import Milvus
import tempfile

db_file = tempfile.NamedTemporaryFile(prefix="milvus_", suffix=".db", delete=False).name
print(f"The vector database will be saved to {db_file}")

vector_db = Milvus(
    embedding_function=embeddings_model,
    connection_args={"uri": db_file},
    auto_id=True,
    index_params={"index_type": "AUTOINDEX"},
)
```

```
The vector database will be saved to /tmp/milvus_kk6rsx2x.db
```

Choose your LLM

The LLM will be used for answering the question, given the retrieved text.

Select a Granite Code model from the `ibm-granite` org on Replicate. Here we use the Replicate Langchain client to connect to the model.

To connect to a model on a provider other than Replicate, substitute this code cell with one from the LLM component recipe.

```
from langchain_community.llms import Replicate
from ibm_granite_community.notebook_utils import get_env_var

model_path = "ibm-granite/granite-3.3-8b-instruct"
model = Replicate(
    model=model_path,
    replicate_api_token=get_env_var('REPLICATE_API_TOKEN'),
)
tokenizer = AutoTokenizer.from_pretrained(model_path)
```

```
REPLICATE_API_TOKEN loaded from Google Colab secret.
tokenizer_config.json:   9.93k/? [00:00<00:00, 513kB/s]
vocab.json:   777k/? [00:00<00:00, 31.2MB/s]
merges.txt:   442k/? [00:00<00:00, 12.2MB/s]
tokenizer.json:   3.48M/? [00:00<00:00, 65.7MB/s]
added_tokens.json: 100%                                207/207 [00:00<00:00, 7.11kB/s]
special_tokens_map.json: 100%                          801/801 [00:00<00:00, 76.0kB/s]
```

Building the Vector Database

In this example, we take the State of the Union speech text, split it into chunks, derive embedding vectors using the embedding model, and load it into the vector database for querying.

Download the document

Here we use President Biden's State of the Union address from March 1, 2022.

```
import os
import wget

filename = 'state_of_the_union.txt'
url = 'https://raw.githubusercontent.com/IBM/watson-machine-learning-samples/master/cloud/data/fou

if not os.path.isfile(filename):
  wget.download(url, out=filename)
```

Split the document into chunks

Split the document into text segments that can fit into the model's context window.

```
from langchain.document_loaders import TextLoader
from langchain.text_splitter import CharacterTextSplitter

loader = TextLoader(filename)
documents = loader.load()
text_splitter = CharacterTextSplitter.from_huggingface_tokenizer(
    tokenizer=embeddings_tokenizer,
    chunk_size=embeddings_tokenizer.max_len_single_sentence,
    chunk_overlap=0,
)
texts = text_splitter.split_documents(documents)
doc_id = 0
for text in texts:
    text.metadata["doc_id"] = (doc_id:=doc_id+1)
print(f"{len(texts)} text document chunks created")
```

```
19 text document chunks created
```

Populate the vector database

NOTE: Population of the vector database may take over a minute depending on your embedding model and service.

```
ids = vector_db.add_documents(texts)
print(f"{len(ids)} documents added to the vector database")
```

```
19 documents added to the vector database
```

Querying the Vector Database

Conduct a similarity search

Search the database for similar documents by proximity of the embedded vector in vector space.

```
query = "What did the president say about Ketanji Brown Jackson?"
docs = vector_db.similarity_search(query)
print(f"{len(docs)} documents returned")
for doc in docs:
    print(doc)
    print("=" * 80)  # Separator for clarity
```

```
The United States is a member along with 29 other nations.

It matters. American diplomacy matters. American resolve matters.

Putin’s latest attack on Ukraine was premeditated and unprovoked.

He rejected repeated efforts at diplomacy.

He thought the West and NATO wouldn’t respond. And he thought he could divide us at home. Put

We prepared extensively and carefully.' metadata={'doc_id': 1, 'pk': 459639652374806528, 'sou
================================================================================
page_content='It’s time for Americans to get back to work and fill our great downtowns again.

We’re doing that here in the federal government. The vast majority of federal workers will on

Our schools are open. Let’s keep it that way. Our kids need to be in school.

And with 75% of adult Americans fully vaccinated and hospitalizations down by 77%, most Ameri

We achieved this because we provided free vaccines, treatments, tests, and masks.

Of course, continuing this costs money.

I will soon send Congress a request.

The vast majority of Americans have used these tools and may want to again, so I expect Congr

Fourth, we will continue vaccinating the world.

We’ve sent 475 Million vaccine doses to 112 countries, more than any other nation.

And we won’t stop.

We have lost so much to COVID-19. Time with one another. And worst of all, so much loss of li

Let’s use this moment to reset. Let’s stop looking at COVID-19 as a partisan dividing line an

Let’s stop seeing each other as enemies, and start seeing each other for who we really are: F

We can’t change how divided we’ve been. But we can change how we move forward—on COVID-19 and

I recently visited the New York City Police Department days after the funerals of Officer Wil

They were responding to a 9-1-1 call when a man shot and killed them with a stolen gun.

Officer Mora was 27 years old.

Officer Rivera was 22.

Both Dominican Americans who’d grown up on the same streets they later chose to patrol as pol
================================================================================
```

Answering Questions

Automate the RAG pipeline

Build a RAG chain with the model and the document retriever.

First we create the prompts for Granite to perform the RAG query. We use the Granite chat template and supply the placeholder values that the LangChain RAG pipeline will replace.

Next, we construct the RAG pipeline by using the Granite prompt templates previously created.

```
from ibm_granite_community.langchain import TokenizerChatPromptTemplate, create_stuff_documents_ch
from langchain.chains.retrieval import create_retrieval_chain

# Create a Granite prompt for question-answering with the retrieved context
prompt_template = TokenizerChatPromptTemplate.from_template("{input}", tokenizer=tokenizer)

# Assemble the retrieval-augmented generation chain
combine_docs_chain = create_stuff_documents_chain(
    llm=model,
    prompt=prompt_template,
)
rag_chain = create_retrieval_chain(
    retriever=vector_db.as_retriever(),
    combine_docs_chain=combine_docs_chain,
)
```

Generate a retrieval-augmented response to a question

Use the RAG chain to process a question. The document chunks relevant to that question are retrieved and used as context.

```
output = rag_chain.invoke({"input": query})

print(output['answer'])
```

```
The president nominated Ketanji Brown Jackson to the United States Supreme Court, describing h
```

Double-click (or enter) to edit