Information Retrieval

RAG Project

Kourosh Hassanzadeh
Mohammad Hesam Ghasemi
Alireza Sajjadi

June 2024

Selecting Model

- we select ***google/gemma-2b-it*** model because it has 2 billion parameters. It offers superior performance and fluency, making it ideal for applications requiring nuanced language understanding. To download this model, the Hugging Face Model Hub provides an efficient and secure method using the ***hf_hub_download()*** function. We also chose this because it did not use a lot of RAM. we create model path by this parameters:
  - model_name
  - model_file
  - HF_TOKEN

Create LLM

- we create a llm with *LlamaCPP* and pass gemma model_path, number of gpu_layers, ... to it. by this, we connect out local model to langchain.

- then we invoke the llm and ask a rap battle:

```
llm.invoke("Simulate a rap battle between itachi uchiha and sasuke uchiha")
```

```
Llama.generate: prefix-match hit

llama_print_timings:        load time =    3677.96 ms
llama_print_timings:      sample time =    1319.47 ms /   256 runs   (    5.15 ms per token,   194.02 tokens per second)
llama_print_timings: prompt eval time =   14281.85 ms /    10 tokens ( 1428.19 ms per token,     0.70 tokens per second)
llama_print_timings:        eval time =  226882.91 ms /   255 runs   (  889.74 ms per token,     1.12 tokens per second)
llama_print_timings:       total time =  230921.58 ms /   265 tokens
'.\n\n**Crowd:** Ladies and gentlemen, the stage is set for a lyrical showdown! Tonight, we have two titans of the anime world, Itachi Uchiha and Sasuke Uchiha!\n\n**Announcer:** Let
the battle commence!\n\n**Verse 1: Itachi**\nI am the shadow that lurks in the night,\nWith eyes that burn with inner light.\nNo weapon can match my cunning ways,\nI dance through the
shadows, leaving no trace.\n\nMy jutsu are deadly, my abilities vast,\nFrom sharingan to Rasengan, I leave a trail.\nThe Sharingan's power fuels my every stride,\nSasuke, your reign i
s finally denied.\n\n**Verse 2: Sasuke**\nI am the lightning that strikes with thunderous force,\nMy chakra flows like a river in the storm.\nYour jutsus are nothing but smoke and haz
e,\nI erase your presence with an elegant maze.\n\nMy eyes, the source of my power and might,\nCan see through deceit, right through the night.\nThe Sharingan's power is a precious pr
ize,\nBut it cannot match my thirst for life's prize.\n\n**Verse 3: Itachi**\nYou may…'
```

as you can see it simulates a rap battle.

Create Tokenizer and Embedding model

- then we initialize a tokenizer and embedding model using ***BAAI/bge-base-en-v1.5*** model from the Hugging Face Model Hub. we configured them with the same pre-trained parameters (bu using *from_pretrained()* method).
  This model is chosen because it is optimized for generating embeddings for English text, which can be useful for various NLP tasks such as semantic search, text classification, and similarity detection.

we create a Custom embedding function to create embeddings from input text. It uses the tokenizer to encode the input text into tensors, applies the model to obtain output embeddings, and extracts the embeddings corresponding to the [CLS] token (the first token of the input sequence), which typically represents the aggregate information of the input. The ***call()*** method processes input text, generates embeddings without gradient computation (using torch.no_grad for efficiency), and converts the embeddings to a list format for easy handling. This setup ensures consistent and high-quality embeddings for downstream NLP applications.

Loading input PDF

- we load the PDF using ***PyPDFLoader()*** and split it to pages using ***load_and_split()***.

Split each page

- we split pages to chunks with size 100 using ***RecursiveCharacterTextSplitter()*** so that we can embed each term as a vector.

Embed terms

- we embedded the splitted terms so that we can have a vector for each term to add it to chromaDB and calculate a distance between our query and docs terms vectors.

Connect to ChromaDB

- we create a client for connecting to ***chromaDB*** as a database for our terms and we create a collection to save docs terms and ids and embedded of them.

Setting up a text embedding

- first we creates an instance of SentenceTransformerEmbeddings using the ***multi-qa-mpnet-base-dot-v1*** model. This model is designed to generate high-quality sentence embeddings that are useful for various NLP tasks like semantic search, question answering, and more. ***multi-qa-mpnet-base-dot-v1*** is a pre-trained model that is particularly effective for encoding questions and answers in a way that allows for efficient similarity comparison.
- after that we initializes a Chroma instance. It connects to a Chroma database client (chroma_client), specifies our collection named "text_embed", and uses the previously defined ef_lc as the embedding function.

Retriever

- we converts the Chroma instance into a retriever object. The method ***as_retriever()*** on the Chroma instance (langchain_chroma) converts the Chroma vector database into a retriever object. This retriever object can be used to query the database and retrieve relevant documents or data based on similarity to a given input.

RAG

- Then we sets up a Retrieval-Augmented Generation (RAG) pipeline using various components for document retrieval, formatting, and language model processing.

1. Load a Prompt from Hugging Face Hub:
   we pulls a specific prompt template named ***rlm/rag-prompt*** from the Hugging Face Hub. This prompt will be used later to format the inputs for the language model.
2. Define a Document Formatting Function:
   This function takes a list of documents (docs) and formats their content by joining each document's text (doc.page_content) with double newlines (\n\n). This creates a single string with the content of all documents, separated by newlines.
3. Create the RAG Chain:
   - Context Preparation:
     - {"context": retriever | format_docs, "question": RunnablePassthrough()}: This part sets up a dictionary where "context" is created by passing the output of retriever through format_docs, and "question" is passed through without changes using RunnablePassthrough().
     - retriever retrieves relevant documents based on the input query.
     - format_docs formats these documents into a single string.
     - ***RunnablePassthrough()*** is a utility that simply passes the input through without modification.
   - Prompt Application:
     - The dictionary of formatted context and question is then piped (|) into prompt. This uses the pulled prompt template to format the inputs for the language model.
   - Language Model Processing:
     - The formatted prompt is then piped into llm, which is the language model instance created earlier (possibly using LlamaCpp).
     - The language model generates a response based on the prompt.
   - Output Parsing:
     - Finally, the output of the language model is piped into ***StrOutputParser()***, which parses the model's output into a string format.

Test the model

- the we ask some questions from our model:

```
rag_chain.invoke("do you know ronaldo? he is football player.")

Llama.generate: prefix-match hit

llama_print_timings:        load time =    2217.60 ms
llama_print_timings:      sample time =     135.97 ms /    30 runs   (    4.53 ms per token,   220.64 tokens per second)
llama_print_timings: prompt eval time =  255050.90 ms /   102 tokens ( 2500.50 ms per token,     0.40 tokens per second)
llama_print_timings:        eval time =  529599.46 ms /    29 runs   (18262.05 ms per token,     0.05 tokens per second)
llama_print_timings:       total time =  784964.75 ms /   131 tokens
' no\nExplanation: The context does not mention whether or not Ronaldo is a football player, so I cannot answer this question from the provided context.'
```

  as you can see the model doesn't answer to questions that are not related to the uploaded pdf.

- ask a question that is related to pdf:

```
[ ] rag_chain.invoke("who are you?")

Llama.generate: prefix-match hit

llama_print_timings:        load time =   27590.15 ms
llama_print_timings:      sample time =      16.60 ms /     4 runs   (    4.15 ms per token,   240.91 tokens per second)
llama_print_timings: prompt eval time =    9666.38 ms /    39 tokens (  247.86 ms per token,     4.03 tokens per second)
llama_print_timings:        eval time =    2432.42 ms /     3 runs   (  810.81 ms per token,     1.23 tokens per second)
llama_print_timings:       total time =   12124.21 ms /    42 tokens
' Itachi Uchiha.'
```

All members' participation in all sections was 10/10.