Practical 5

Run rag locally using free embedding, phis index

Introduction to RAG

Retrieval-Augmented Generation (RAG) is an advanced Natural Language Processing (NLP) technique that enhances the ability of language models to generate accurate and informative responses by supplementing them with relevant external knowledge. Unlike traditional models that generate text based solely on pre-trained parameters, RAG combines retrieval-based and generation-based mechanisms. It retrieves relevant documents or context from a large corpus using a similarity search and feeds this context into a language model to generate more informed and contextually accurate outputs.

Purpose of Using RAG

The goal of building a RAG application is to overcome the limitations of static knowledge in pre-trained models. Even powerful models like Meta's LLaMA 2 (Large Language Model Meta AI) are limited to knowledge up to their training cutoff date. By integrating retrieval systems like FAISS, we can supply the model with up-to-date or domain-specific information during inference time, resulting in more relevant and dynamic responses.

Overview of Technologies Used

1. Google Colab

Google Colab is a cloud-based Jupyter notebook environment that allows you to write and execute Python code using free GPU/TPU support. It is ideal for experimenting with machine learning workflows without needing local hardware.

2. Hugging Face Transformers

Hugging Face provides a powerful ecosystem of pre-trained language models and tools for NLP. Their transformers library simplifies the integration of state-of-the-art models, including LLaMA 2 (via Hugging Face's transformers or through their text-generation-inference API).

3. FAISS (Facebook AI Similarity Search)

FAISS is an efficient library developed by Facebook AI Research to perform fast similarity search and clustering of dense vectors. In the context of RAG, FAISS is used to index and retrieve text embeddings that are similar to the user's query, serving as a retrieval backbone.

4. LLaMA 2 7B

LLaMA 2 7B is a 7-billion parameter open-source large language model by Meta, known for its competitive performance and smaller size compared to other LLMs. It is a suitable model for integration into lightweight RAG systems, especially in environments like Colab with limited resources.

Workflow of the RAG Application

1. Document Preparation and Embedding

The first step in creating a RAG system is to prepare the knowledge base. This involves:

- Collecting documents (e.g., PDFs, web articles, structured text).
- Chunking them into manageable segments (usually around 100-500 words).
- Converting each chunk into a dense vector using a sentence embedding model (e.g., sentence-transformers, BERT, or Hugging Face's embedding models).

These dense vectors capture the semantic meaning of each chunk and serve as the basis for retrieval.

2. Indexing with FAISS

Once the embeddings are generated, they are added to a FAISS index. This index is built for efficient nearest-neighbor search. The FAISS index enables rapid retrieval of relevant chunks based on the similarity between a user's query and the embedded document vectors.

FAISS supports different types of indexing techniques (Flat, IVFFlat, HNSW, etc.), depending on the trade-off between speed, memory, and accuracy.

3. Query Embedding and Retrieval

When the user inputs a question, the system:

- Embeds the query using the same embedding model used for the documents.
- Searches the FAISS index for the top-k most relevant document chunks.
- Retrieves those top-k chunks as external context for the language model.

This retrieval step ensures that the language model has up-to-date and task-specific information available during generation.

4. Augmented Generation using LLaMA 2 7B

The final step involves passing the retrieved context along with the user's original query to the LLaMA 2 7B model. The prompt typically includes both:

- A structured context section with the retrieved information.

- The user’s question or command.

The model then generates a response grounded in the retrieved content, improving factual accuracy and relevance. This generation can be done using Hugging Face’s transformers library if the model is loaded locally or by using the Hugging Face Inference API for hosted LLaMA models (especially if Colab’s compute is insufficient for loading 7B+ models).

Benefits of This RAG Setup

- **Scalability**: Easily handles a growing corpus by updating FAISS indices.
- **Accuracy**: Reduces hallucination in generative models by grounding answers in retrieved data.
- **Efficiency**: FAISS enables low-latency retrieval, and LLaMA 2 7B balances performance with compute requirements.
- **Modularity**: Each component (embedding, indexing, generation) is replaceable and customizable.

Use Cases

- **Document Question Answering**: Answering queries based on internal company documentation.
- **Personalized Chatbots**: Creating assistants grounded in personal notes or knowledge bases.
- **Academic Research Assistants**: Summarizing or answering questions from research papers.
- **Customer Support**: Automating responses based on historical customer service data.

```
!pip install transformers accelerate faiss-cpu sentence-transformers PyMuPDF
```

```
Requirement already satisfied: transformers in /usr/local/lib/python3.11/dist-packages (4.51.1)
Requirement already satisfied: accelerate in /usr/local/lib/python3.11/dist-packages (1.5.2)
Collecting faiss-cpu
  Downloading faiss_cpu-1.10.0-cp311-cp311-manylinux_2_28_x86_64.whl.metadata (4.4 kB)
Requirement already satisfied: sentence-transformers in /usr/local/lib/python3.11/dist-packages (3.4.1)
Collecting PyMuPDF
  Downloading pymupdf-1.25.5-cp39-abi3-manylinux2014_x86_64.manylinux_2_17_x86_64.whl.metadata (3.4 kB)
Requirement already satisfied: filelock in /usr/local/lib/python3.11/dist-packages (from transformers) (3.18.0)
Requirement already satisfied: huggingface-hub<1.0,>=0.30.0 in /usr/local/lib/python3.11/dist-packages (from transformers) (0.30.2)
Requirement already satisfied: numpy>=1.17 in /usr/local/lib/python3.11/dist-packages (from transformers) (2.0.2)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.11/dist-packages (from transformers) (24.2)
Requirement already satisfied: pyyaml>=5.1 in /usr/local/lib/python3.11/dist-packages (from transformers) (6.0.2)
Requirement already satisfied: regex!=2019.12.17 in /usr/local/lib/python3.11/dist-packages (from transformers) (2024.11.6)
Requirement already satisfied: requests in /usr/local/lib/python3.11/dist-packages (from transformers) (2.32.3)
Requirement already satisfied: tokenizers<0.22,>=0.21 in /usr/local/lib/python3.11/dist-packages (from transformers) (0.21.1)
Requirement already satisfied: safetensors>=0.4.3 in /usr/local/lib/python3.11/dist-packages (from transformers) (0.5.3)
Requirement already satisfied: tqdm>=4.27 in /usr/local/lib/python3.11/dist-packages (from transformers) (4.67.1)
Requirement already satisfied: psutil in /usr/local/lib/python3.11/dist-packages (from accelerate) (5.9.5)
Requirement already satisfied: torch>=2.0.0 in /usr/local/lib/python3.11/dist-packages (from accelerate) (2.6.0+cu124)
Requirement already satisfied: scikit-learn in /usr/local/lib/python3.11/dist-packages (from sentence-transformers) (1.6.1)
Requirement already satisfied: scipy in /usr/local/lib/python3.11/dist-packages (from sentence-transformers) (1.14.1)
Requirement already satisfied: Pillow in /usr/local/lib/python3.11/dist-packages (from sentence-transformers) (11.1.0)
Requirement already satisfied: fsspec>=2023.5.0 in /usr/local/lib/python3.11/dist-packages (from huggingface-hub<1.0,>=0.30.0->transf
Requirement already satisfied: typing-extensions>=3.7.4.3 in /usr/local/lib/python3.11/dist-packages (from huggingface-hub<1.0,>=0.30
Requirement already satisfied: networkx in /usr/local/lib/python3.11/dist-packages (from torch>=2.0.0->accelerate) (3.4.2)
Requirement already satisfied: jinja2 in /usr/local/lib/python3.11/dist-packages (from torch>=2.0.0->accelerate) (3.1.6)
Collecting nvidia-cuda-nvrtc-cu12==12.4.127 (from torch>=2.0.0->accelerate)
  Downloading nvidia_cuda_nvrtc_cu12-12.4.127-py3-none-manylinux2014_x86_64.whl.metadata (1.5 kB)
Collecting nvidia-cuda-runtime-cu12==12.4.127 (from torch>=2.0.0->accelerate)
  Downloading nvidia_cuda_runtime_cu12-12.4.127-py3-none-manylinux2014_x86_64.whl.metadata (1.5 kB)
Collecting nvidia-cuda-cupti-cu12==12.4.127 (from torch>=2.0.0->accelerate)
  Downloading nvidia_cuda_cupti_cu12-12.4.127-py3-none-manylinux2014_x86_64.whl.metadata (1.6 kB)
Collecting nvidia-cudnn-cu12==9.1.0.70 (from torch>=2.0.0->accelerate)
  Downloading nvidia_cudnn_cu12-9.1.0.70-py3-none-manylinux2014_x86_64.whl.metadata (1.6 kB)
Collecting nvidia-cublas-cu12==12.4.5.8 (from torch>=2.0.0->accelerate)
  Downloading nvidia_cublas_cu12-12.4.5.8-py3-none-manylinux2014_x86_64.whl.metadata (1.5 kB)
Collecting nvidia-cufft-cu12==11.2.1.3 (from torch>=2.0.0->accelerate)
  Downloading nvidia_cufft_cu12-11.2.1.3-py3-none-manylinux2014_x86_64.whl.metadata (1.5 kB)
Collecting nvidia-curand-cu12==10.3.5.147 (from torch>=2.0.0->accelerate)
  Downloading nvidia_curand_cu12-10.3.5.147-py3-none-manylinux2014_x86_64.whl.metadata (1.5 kB)
Collecting nvidia-cusolver-cu12==11.6.1.9 (from torch>=2.0.0->accelerate)
  Downloading nvidia_cusolver_cu12-11.6.1.9-py3-none-manylinux2014_x86_64.whl.metadata (1.6 kB)
Collecting nvidia-cusparse-cu12==12.3.1.170 (from torch>=2.0.0->accelerate)
  Downloading nvidia_cusparse_cu12-12.3.1.170-py3-none-manylinux2014_x86_64.whl.metadata (1.6 kB)
Requirement already satisfied: nvidia-cusparselt-cu12==0.6.2 in /usr/local/lib/python3.11/dist-packages (from torch>=2.0.0->accelerat
Requirement already satisfied: nvidia-nccl-cu12==2.21.5 in /usr/local/lib/python3.11/dist-packages (from torch>=2.0.0->accelerate) (2
Requirement already satisfied: nvidia-nvtx-cu12==12.4.127 in /usr/local/lib/python3.11/dist-packages (from torch>=2.0.0->accelerate)
Collecting nvidia-nvjitlink-cu12==12.4.127 (from torch>=2.0.0->accelerate)
  Downloading nvidia_nvjitlink_cu12-12.4.127-py3-none-manylinux2014_x86_64.whl.metadata (1.5 kB)
Requirement already satisfied: triton==3.2.0 in /usr/local/lib/python3.11/dist-packages (from torch>=2.0.0->accelerate) (3.2.0)
Requirement already satisfied: sympy==1.13.1 in /usr/local/lib/python3.11/dist-packages (from torch>=2.0.0->accelerate) (1.13.1)
Requirement already satisfied: mpmath<1.4,>=1.1.0 in /usr/local/lib/python3.11/dist-packages (from sympy==1.13.1->torch>=2.0.0->accel
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.11/dist-packages (from requests->transformers) (3.4
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.11/dist-packages (from requests->transformers) (3.10)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.11/dist-packages (from requests->transformers) (2.3.0)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.11/dist-packages (from requests->transformers) (2025.1.31
Requirement already satisfied: joblib>=1.2.0 in /usr/local/lib/python3.11/dist-packages (from scikit-learn->sentence-transformers) (1
```

```
import fitz  # PyMuPDF
from google.colab import files

# Upload PDF
uploaded = files.upload()

# Extract text from uploaded PDF
pdf_text = ""
for filename in uploaded.keys():
    with fitz.open(filename) as doc:
        for page in doc:
            pdf_text += page.get_text()
```

Choose Files Control Unit…3-Part-4.pdf

- **Control Unit-Unit-3-Part-4.pdf**(application/pdf) - 2787572 bytes, last modified: 8/29/2024 - 100% done

```
Saving Control Unit-Unit-3-Part-4.pdf to Control Unit-Unit-3-Part-4.pdf
```

```
from textwrap import wrap
```

```
# Split into chunks (adjust size based on model token limit)
def chunk_text(text, chunk_size=500):
    return wrap(text, chunk_size)

documents = chunk_text(pdf_text)
print(f"Total Chunks: {len(documents)}")
```

```
Total Chunks: 123
```

```
from sentence_transformers import SentenceTransformer
import faiss
import numpy as np

embedder = SentenceTransformer('all-MiniLM-L6-v2')  # Light, fast model

document_embeddings = embedder.encode(documents)

dimension = document_embeddings.shape[1]
index = faiss.IndexFlatL2(dimension)
index.add(np.array(document_embeddings))
```

```
/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/settings/tokens), set it as secre
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, 26.8kB/s]
config_sentence_transformers.json: 100%                    116/116 [00:00<00:00, 11.9kB/s]
README.md: 100%                    10.5k/10.5k [00:00<00:00, 537kB/s]
sentence_bert_config.json: 100%                    53.0/53.0 [00:00<00:00, 2.59kB/s]
config.json: 100%                    612/612 [00:00<00:00, 32.1kB/s]
Xet Storage is enabled for this repo, but the 'hf_xet' package is not installed. Falling back to regular HTTP download. For better perfo
WARNING:huggingface_hub.file_download:Xet Storage is enabled for this repo, but the 'hf_xet' package is not installed. Falling back to r
model.safetensors: 100%                    90.9M/90.9M [00:00<00:00, 194MB/s]
tokenizer_config.json: 100%                    350/350 [00:00<00:00, 28.8kB/s]
vocab.txt: 100%                    232k/232k [00:00<00:00, 1.60MB/s]
tokenizer.json: 100%                    466k/466k [00:00<00:00, 3.19MB/s]
special_tokens_map.json: 100%                    112/112 [00:00<00:00, 10.4kB/s]
config.json: 100%                    190/190 [00:00<00:00, 20.0kB/s]
```

```
def retrieve_top_k(query, k=3):
    query_embedding = embedder.encode([query])
    distances, indices = index.search(query_embedding, k)
    return [documents[i] for i in indices[0]]
```

```
# ✅ Step 5: Load Mistral 7B Instruct Model
from transformers import AutoTokenizer, AutoModelForCausalLM

model_name = "mistralai/Mistral-7B-Instruct-v0.1"

tokenizer = AutoTokenizer.from_pretrained(model_name)
model = AutoModelForCausalLM.from_pretrained(
    model_name,
    device_map="auto",
    torch_dtype="auto"
)
```

```
tokenizer_config.json: 100%                    2.10k/2.10k [00:00<00:00, 203kB/s]
tokenizer.model: 100%                          493k/493k [00:00<00:00, 6.13MB/s]
tokenizer.json: 100%                           1.80M/1.80M [00:00<00:00, 2.87MB/s]
special_tokens_map.json: 100%                  414/414 [00:00<00:00, 43.6kB/s]
config.json: 100%                              571/571 [00:00<00:00, 63.2kB/s]
model.safetensors.index.json: 100%             25.1k/25.1k [00:00<00:00, 2.75MB/s]
Fetching 2 files: 100%                         2/2 [05:27<00:00, 327.87s/it]
model-00002-of-00002.safetensors: 100%         4.54G/4.54G [04:04<00:00, 8.65MB/s]
model-00001-of-00002.safetensors: 100%         9.94G/9.94G [05:27<00:00, 24.0MB/s]
Loading checkpoint shards: 100%                2/2 [00:53<00:00, 24.60s/it]
generation_config.json: 100%                   116/116 [00:00<00:00, 8.84kB/s]
WARNING:accelerate.big_modeling:Some parameters are on the meta device because they were offloaded to the cpu.
```

```
# ✅ Step 6: Generate answers using retrieved context
import torch

def generate_answer(question, max_tokens=200):
    context = "\n".join(retrieve_top_k(question))
    prompt = f"[INST] Use the following context to answer the question.\n\nContext:\n{context}\n\nQuestion: {question} [/INST]"

    inputs = tokenizer(prompt, return_tensors="pt").to("cuda")
    output = model.generate(**inputs, max_new_tokens=max_tokens)

    return tokenizer.decode(output[0], skip_special_tokens=True)

# 🔍 Try a question
print(generate_answer("What is the main topic of the PDF?"))
```

```
Setting `pad_token_id` to `eos_token_id`:2 for open-end generation.
[INST] Use the following context to answer the question.

Context:
Execute    11: Interrupt    At the end of each of the four cycles, the ICC is set appropriately. The indirect  cycle is always followed
diagram. For an instruction cycle, various states can be null,  while others can be visited more than once.    • Instruction Address Cal
address to the decoder. The decoder  identifies  the  corresponding  micro-instructions  from  the  Control  Memory.   A micro-instructi

Question: What is the main topic of the PDF? [/INST] The main topic of the PDF is the instruction cycle of a basic computer, which is de
```