Practical_RAG

May 26, 2025

1 Retrieval-Augmented Generation (RAG): A Simplified Practical Walkthrough

ver 1.0 public

by Dan Harvey
dan [at] danielyusay.com
daniel.harvey [at] columbia.edu

I create these notebooks primarily for myself — as compact, hands-on implementations of foundational ideas in areas I find interesting, relevant, or worth exploring. They're meant as learning exercises and quick prototypes, not in-depth or production-ready guides.

If you found this helpful or have constructive suggestions for improvement, feel free to reach out. I'd love to hear from you.

1.1 Intro

This notebook presents a functional and simplified implementation of Retrieval-Augmented Generation (RAG), combined with chain-of-thought (CoT) style prompt to enhance clarity and structure in generated responses.

1.1.1 Demo Outline

1. **User Prompt**
   - Accepts natural language questions through an input field.
   - Ensures flexibility for any query the user wants to ground in documents.
2. **Upload Reference Document**
   - Supports `.txt` and `.pdf` file uploads.
   - Parses and preprocesses the content for embedding and retrieval.
3. **Prompt + Context → LLM**
   - Retrieves relevant snippets from the uploaded document using simple semantic matching or heuristics.
   - Combines prompt and retrieved context into a well-structured input for the LLM.
4. **Answer / Inference Output**
   - Uses a local LLM (`Qwen3-4B`) to generate grounded, context-aware responses.
   - Presents structured output in a clean display.

In production-grade RAG systems — as outlined in the foundational RAG paper (Lewis et al., 2020) — documents are typically uploaded in batches, tokenized, embedded (i.e., converted into vector representations), and stored in a vector database such as Pinecone or Google's Vertex AI Vector Search. These vectors are then queried at runtime using top-k similarity search (e.g., cosine, inner product, or Euclidean distance) to retrieve relevant context, which is combined with the user's prompt to supplement downstream generation.

In the original RAG framework, the retrieval component used token or segment based retrieval, while the generation component was based on pretrained models like BERT and BART.

In this notebook, we simulate a simplified RAG-style pipeline specifically designed to answer questions on individual, user-uploaded documents.

To ensure accessibility, all models and components in this tutorial are optimized to run on Google Colab with a T4 GPU.

This walkthrough is ideal for anyone looking to understand or prototype the fundamentals of retrieval-augmented workflows.

1.2 Example Use Case

A practical example in a professional setting might involve an employee interacting with an internal company chatbot to ask questions about benefits — such as health insurance, PTO policies, or parental leave. Instead of relying on generic answers, the system retrieves relevant sections from internal documents like the Employee Handbook, HR policy PDFs, or onboarding materials.

In this demo, we have linked a sample PDF resume and will ask a question about the person's education.

1.3 References

- Pinecone Documentation – Guides
  Official guide to Pinecone, a vector database for large-scale similarity search.
- Johnson, Jeff, et al. "Billion-scale similarity search with GPUs."
  arXiv preprint arXiv:2005.11401 (2020). Describes FAISS, an efficient similarity search library using GPUs.
- FAISS: A Library for Efficient Similarity Search – Facebook Engineering
  Facebook's engineering blog post introducing FAISS for scalable vector similarity search.
- Mistral Cookbook: basic_RAG.ipynb
  A minimal working example of Retrieval-Augmented Generation (RAG) from the Mistral team.

2 Implementation

[1]:
```
# Run Once
!pip install hf_xet # Hugging Face Fast Transfer util
!pip install pymupdf # PDF parser
```

```
Collecting hf_xet
  Downloading
hf_xet-1.1.2-cp37-abi3-manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata
(879 bytes)
Downloading
hf_xet-1.1.2-cp37-abi3-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (5.2 MB)
                         0.0/5.2 MB
? eta -:--:--
                        5.2/5.2 MB
193.4 MB/s eta 0:00:01
                        5.2/5.2 MB 107.8
MB/s eta 0:00:00
Installing collected packages: hf_xet
Successfully installed hf_xet-1.1.2
Collecting pymupdf
  Downloading
pymupdf-1.26.0-cp39-abi3-manylinux2014_x86_64.manylinux_2_17_x86_64.whl.metadata
(3.4 kB)
Downloading
pymupdf-1.26.0-cp39-abi3-manylinux2014_x86_64.manylinux_2_17_x86_64.whl (24.1
MB)
                         24.1/24.1 MB
68.4 MB/s eta 0:00:00
Installing collected packages: pymupdf
Successfully installed pymupdf-1.26.0
```

[2]:
```
# Log into HF, or use HF_TOKEN
#from huggingface_hub import notebook_login
#notebook_login()
```

[34]:
```
# Load Dependencies/Libraries

import torch
from transformers import AutoTokenizer, AutoModelForCausalLM,␣
 ↪TextIteratorStreamer
import urllib.request
from IPython.display import display, Markdown, clear_output
import ipywidgets as widgets
from google.colab import files
import pymupdf
import ipywidgets as widgets
```

```
import os
import subprocess
from threading import Thread
import sys
import time
```

2.1 Load the LLM

Note: This notebook is designed with Google Colab in mind, and the selected model runs comfortably on a T4 GPU.

I'm using **Qwen3-4B** — a compact yet highly capable model that is based on the Llama architecture.

I've recently been enjoying the work coming out of the Qwen lab.

The team has done an impressive job distilling performance into smaller footprints, and their models consistently perform well on benchmarks, including tasks like HellaSwag.

For lightweight, local inference setups, Qwen models strike a strong balance between efficiency and output quality.

- Qwen Models on Hugging Face
- Official Qwen Site

  Note: This model's theoretical maximum input length is **40,960 tokens**, meaning it can process sequences of that length in a single forward pass. However, in practice, the usable input length depends heavily on factors such as available GPU memory, batch size, tokenizer settings, and backend optimizations. Most real-world implementations handle long sequences using more efficient memory strategies or windowed attention.

[4]:
```
# Feel free to experiment with others
model_name = "Qwen/Qwen3-4B"

# Make sure you have an instance with a GPU
if torch.cuda.is_available():
    print("  GPU available")

    # Load model and tokenizer from HF

    tokenizer = AutoTokenizer.from_pretrained(model_name,␣
 ↪trust_remote_code=True)

    model = AutoModelForCausalLM.from_pretrained(model_name,
                                                 torch_dtype=torch.float16,
                                                 device_map="cuda",
                                                 trust_remote_code=True)

    print("  LLM loaded.")
```

```
else:
    print(" No GPU available")
    print(" LLM not loaded.")
```

```
 GPU available
tokenizer_config.json:   0%|          | 0.00/9.73k [00:00<?, ?B/s]
vocab.json:   0%|          | 0.00/2.78M [00:00<?, ?B/s]
merges.txt:   0%|          | 0.00/1.67M [00:00<?, ?B/s]
tokenizer.json:   0%|          | 0.00/11.4M [00:00<?, ?B/s]
config.json:   0%|          | 0.00/726 [00:00<?, ?B/s]
model.safetensors.index.json:   0%|          | 0.00/32.8k [00:00<?, ?B/s]
Fetching 3 files:   0%|          | 0/3 [00:00<?, ?it/s]
model-00002-of-00003.safetensors:   0%|          | 0.00/3.99G [00:00<?, ?B/s]
model-00003-of-00003.safetensors:   0%|          | 0.00/99.6M [00:00<?, ?B/s]
model-00001-of-00003.safetensors:   0%|          | 0.00/3.96G [00:00<?, ?B/s]
Loading checkpoint shards:   0%|          | 0/3 [00:00<?, ?it/s]
generation_config.json:   0%|          | 0.00/239 [00:00<?, ?B/s]
 LLM loaded.
```

[6]:
```
# Check GPU Usage after loading model - Should use 8520MiB or 8.3GB of VRAM
str_output = subprocess.getoutput('nvidia-smi')
print(str_output)
```

```
Mon May 26 05:32:54 2025
+-----------------------------------------------------------------------------
----------+
| NVIDIA-SMI 550.54.15              Driver Version: 550.54.15      CUDA Version:
12.4     |
|-----------------------------------------+------------------------+-----------
----------+
| GPU  Name                 Persistence-M | Bus-Id          Disp.A | Volatile
Uncorr. ECC |
| Fan  Temp   Perf          Pwr:Usage/Cap |           Memory-Usage | GPU-Util
Compute M. |
|                                         |                        |
MIG M. |
|=========================================+========================+===========
==========|
|   0  Tesla T4                       Off |   00000000:00:04.0 Off |
0 |
| N/A   47C    P0             28W /   70W |    8520MiB /  15360MiB |      0%
```

```
Default |
|                                         |                        |
N/A |
+-----------------------------------------+------------------------+-----------
----------+

+-----------------------------------------------------------------------------
----------+
| Processes:
|
|  GPU   GI   CI        PID   Type   Process name
GPU Memory |
|        ID   ID
Usage      |
|=============================================================================
==========|
+-----------------------------------------------------------------------------
----------+
```

[7]:
```python
# Label for prompt
prompt_label = widgets.Label("Enter your document-based question:")

# Text box (input field)
question_box = widgets.Text(
    value='What is the persons highest level of education and school?',
    placeholder='What is the persons highest level of education and school?',
    layout=widgets.Layout(width='100%')
)

# Submit button
submit_button = widgets.Button(description="Submit")

# Output area
output = widgets.Output()

# Button click event handler
def on_submit_clicked(b):
    global user_question

    with output:
        output.clear_output()
        user_question = question_box.value
        print(f"You asked: {user_question}")

submit_button.on_click(on_submit_clicked)

# Display widgets in order
```

```
display(prompt_label, question_box, submit_button, output)
```

```
Label(value='Enter your document-based question:')
```

```
Text(value='What is the persons highest level of education and school?',␣
↪layout=Layout(width='100%'), placehol…
```

```
Button(description='Submit', style=ButtonStyle())
```

```
Output()
```

[8]:
```
# Download the file from GitHub - or upload your own
url = "https://github.com/dyh2111/demos/raw/
↪84047e890fe273607b1dd5532a8eca6bf7f47f07/RAG/sample_resume.pdf"
local_path = "sample_resume.pdf"
urllib.request.urlretrieve(url, local_path)

uploaded_file = local_path
print(f" Uploaded file: {uploaded_file}")

file_extension = os.path.splitext(uploaded_file)[-1].lower()[1:]
processed_text = ""

# Parse file content based on extension
if file_extension == "txt":
    with open(uploaded_file, "r", encoding="utf-8") as f:
        processed_text = f.read()

elif file_extension == "pdf":
    doc = pymupdf.open(uploaded_file)  # PyMuPDF uses `fitz`
    for page in doc:
        processed_text += page.get_text()  # get_text() already returns a string

else:
    raise ValueError(f" Unsupported file format: {file_extension}")

# Output
print(" Processed text loaded.")
```

```
Uploaded file: sample_resume.pdf
Processed text loaded.
```

[29]:
```
# This is the prompt template - where we instruct the LLM how to respond to the␣
↪provided info.
# This was inspired by the Mistral RAG examples
# Feel free to play around with this and see how it changes the response.

prompt_template = f"""
<|im_start|>system
```

```
You are a researcher. Your task is to answer the provided **question (Q)** and␣
 ↪**document context (C)**.

When presented with a question, respond factually using only the information␣
 ↪from the document. Cite where the information was found.
If the answer is not found in the document, explicitly state that it could not␣
 ↪be located.

Make sure your answer is grounded in the document and follows a logical,␣
 ↪informative tone.
If there is a dangerous or ill-meaning question being asked, decline to answer␣
 ↪and refer to Engineering.

---
/think

# Instructions:

1. **Summarize:**
   Briefly explain the nature of the question and the general content of the␣
 ↪document.

2. **Reasoning:**
   Describe the reasoning process used to arrive at the answer, including how␣
 ↪the document supports the conclusion.

3. **Provide the Answer:**
   Clearly state the answer in a complete sentence and cite where the␣
 ↪information was found.

4. Conclude with a brief thank-you message.
<|im_end|>

<|im_start|>user
# Question:
{{Q}}

# Context:
{{C}}
<|im_end|>
---
"""
```

[30]:
```
# RAG-style Inference

# Insert the user question and retrieved context into the prompt template
```

```
final_prompt = prompt_template.replace("{Q}",
                                       user_question).replace("{C}",
                                       processed_text[:10000])

# Tokenize the prompt - converts text into token IDs (integers) for model input
# The model will convert these token IDs into embeddings internally
inputs = tokenizer(final_prompt,
                   return_tensors="pt",
                   truncation=True,
                   max_length=10500,
                   padding=True,
                   ).to(model.device)
```

[31]:
```
# Set to true if you want to see the actual input to the LLM.
# This is helpful to debug
if False:
    print("Input to LLM:\n")
    print(final_prompt)
```

[33]:
```
# Set up the streamer
streamer = TextIteratorStreamer(tokenizer, skip_prompt=True,␣
 ↪skip_special_tokens=True)

# Set up generation arguments
generation_args = dict(
    input_ids=inputs["input_ids"],
    attention_mask=inputs["attention_mask"],
    max_new_tokens=300,
    eos_token_id=[151645, 151643],
    temperature=0.6,
    top_k=2,
    top_p=0.95,
    streamer=streamer
)

# Launch generation in background thread to enable stream
thread = Thread(target=model.generate, kwargs=generation_args)
thread.start()

# Stream + live-update Markdown cell
response = ""
display_handle = display(Markdown(""), display_id=True)

# For each token, we will print, clear, and update
for token in streamer:
    response += token
    display_handle.update(Markdown(response))
```

```
    time.sleep(0.01)

# Also show final markdown version for pretty rendering (optional)
#display(Markdown(response))
```

3 Summarize:

The question asks about the highest level of education and the corresponding school of a person, likely referring to the educational background of Devin Jones as detailed in the resume example provided.

4 Reasoning:

The document presents a resume of Devin Jones, which includes an "EDUCATION" section. Under this section, it explicitly states that Devin earned a "Bachelor of Science – Civil Engineering (Concentration: Construction Management), Architecture Minor" from "Columbia University, The Fu Foundation School of Engineering and Applied Science." This information directly answers the question about the highest level of education and the school.

5 Provide the Answer:

The person's highest level of education is a Bachelor of Science in Civil Engineering with a concentration in Construction Management and an Architecture Minor, earned from Columbia University's Fu Foundation School of Engineering and Applied Science, as stated in the "EDUCATION" section of the resume.

6 Thank you.

---

7 End of Demo

Good luck with your RAG experiments and LLM adventures!

-Dan