RAG Model: Technical How to Guide

CSCI 89b: Natural Language Processing
Final Project Report: Erin Rebholz, December 2024

Project Motivation:

This project leverages a RAG, or Retrieval Augmented Generation, model and incorporates a large language model to retrieve answers to questions only from specified materials. The recommended readings, lecture notes, and syllabus from CS89B Natural Language Processing have been used to test RAG model application in a working chatbot. Some of the downsides of large language models are a tendency toward hallucination (made up answers) and a lack of specific referenced content to validate information returned by the model. A RAG model helps to mitigate these downsides by drawing answers from specific content and by creating traceback mechanisms to the sources of model answers.

In the project, two types of prompts are developed to test the RAG model. The first is a prompt of general knowledge with specific text references. The second is using a selection of quiz questions from the class as a 'ground truth' to test the answers of the RAG model. Answers for both tests are also compared to 'out of the box' answers returned by the GPT 3.5 turbo model of OpenAi.

This 'Technical How to Guide' is meant to provide reference materials in how to get the code base to work. Implementation requires coordination of multiple packages with interdependencies, API keys and use of the Linux subsystem on a Windows machine.

This 'Technical How to Guide' complements the final presentation of results and demonstration code base also developed for this project. The final presentation, in particular, delves further into methodology and results of the tests for the RAG model implementation to support queries of general course materials based on the course content and a quick test of the model's ability to answer the Quiz questions from the course.

I. General Technology Setup

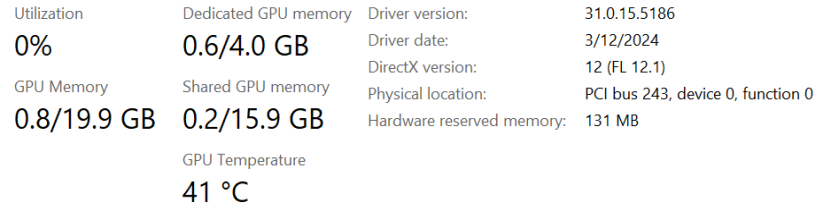
To support this project, I used a laptop equipped with a NVIDIA RTX 3050 GPU. While the GPU was not necessary for accessing and using our chosen LLM, the Windows Linux subsystem was required for using our chosen vector storage application.

General Specifications of Laptop Used for Work:

Device name	Erin_AI_Laptop
Processor	11th Gen Intel(R) Core(TM) i7-11370H @ 3.30GHz 3.30 GHz
Installed RAM	32.0 GB (31.8 GB usable)
Device ID	76660F0F-5C00-48FE-B55F-D2BFAAB3F814
Product ID	00356-06305-62694-AAOEM
System type	64-bit operating system, x64-based processor
Pen and touch	Pen and touch support with 10 touch points

NVIDIA GeForce RTX 3050 Ti Laptop GPU

RAG Model: Technical How to Guide



Linux general set up:

- Launch a new Anaconda Prompt Window
- wsl.exe → to launch Windows Linux environment

To create a new environment from YAML file:

A yaml file from a previous class was supplemented with additional packages that were important for RAG implementation. Creating a new environment for to support the code eliminates the need to pip install the langchain, openai, weaviate and related packages. However, the pip install information is commented out within the code example.

conda env create –file nlp_89b.yml

To load a new environment:

conda activate nlp89b

To launch VSCode:

code . <optional file name in directory to launch>

To Launch Jupyter

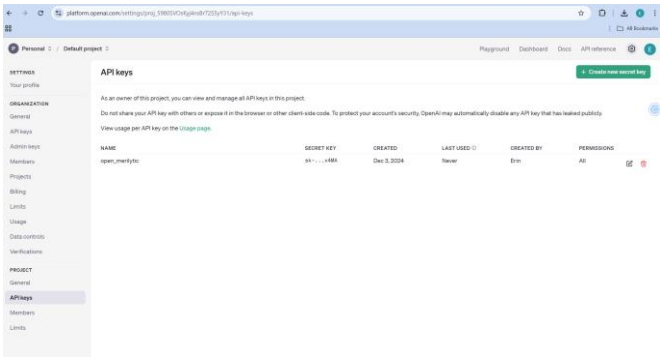
jupyter-lab <optional file name in directory to launch>

II. Open AI Setup:

Open AI is used in the project as both the large language model as well as for creating the emebddings for the documents that will be used in the RAG model. In order to use the Open AI API, you need to sign up for an API key and to create a small budget (>$5) in order to pay for API calls and usage.

Sign up for an open_ai key:
https://platform.openai.com/settings/proj_5980SVOsKyj4nsBr72S5yY31/api-keys

RAG Model: Technical How to Guide



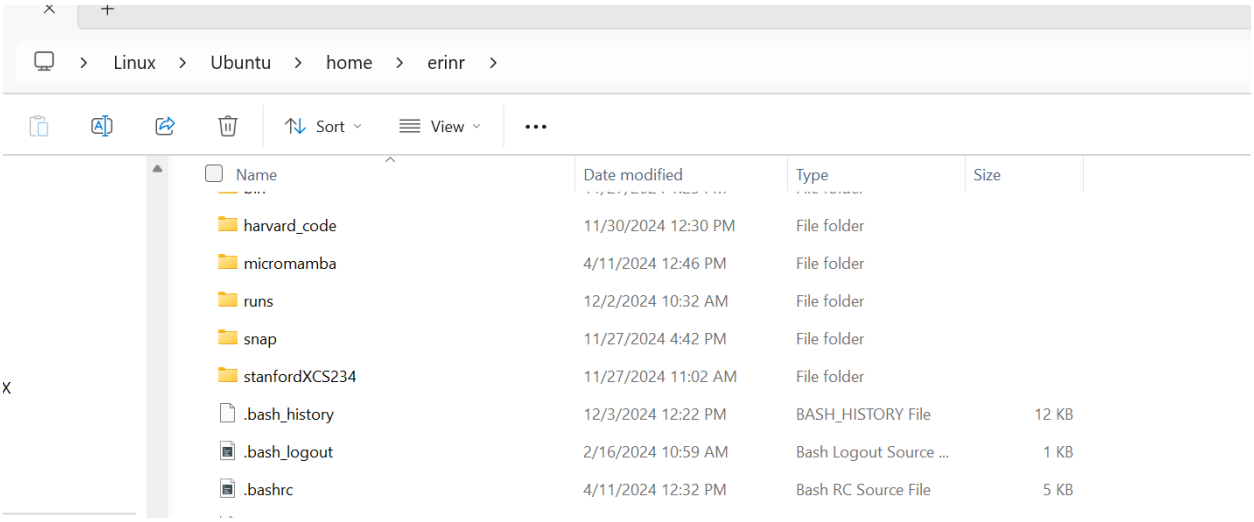
Pricing and Costs:

Pricing is based on number of tokens passed to the models/embeddings and price increases based on the complexity of the models used. For this project purposes, I used the older GPT-3.5 Turbo and basic embeddings, keeping costs low. I set up a small budget $25 for the project, but I used less than $2 in total of the budget.

Saving OpenAI Environment Variable in Linux:
https://hackernoon.com/mastering-environment-variables-in-linux-a-comprehensive-guide

In order to access the secret key for Open AI you need to save it as an environment variable in your user profile. For Linux, which we used due to a dependency of one of our packages, you'll need to locate the ./bashrc file in the linux home directory.

Find your .bashrc file in your linux user directory:



Add a line to the code to the .bashrc file:

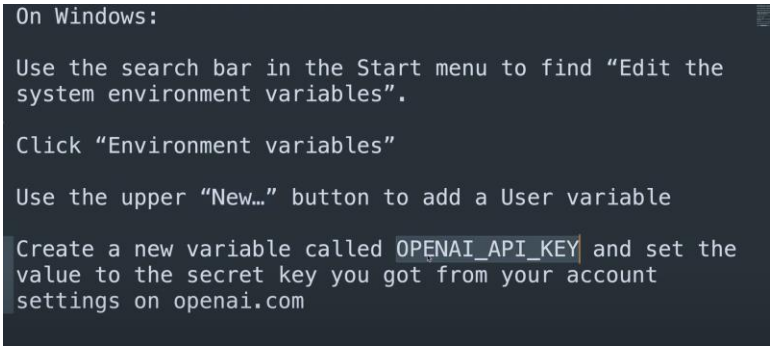
You can open the bashrc file in VS Code or a text editor and add the text to the end of the file:

export OPENAI_API_KEY= <your-secret-key-no-quotes>

Windows environment variable set-up:

If you'd like to use the OPEN_AI_KEY on your windows system, you can also set an environment variable on windows:

From: https://www.youtube.com/watch?v=wpL7z8rYShg



III. Weaviate Vector Store

Storage of embeddings in a vector store is an important component for leveraging large language models at scale.

Weaviate was chosen for this project because it offers an 'embedded' option that allows us to use our local machine to create a vector store that is available when we start up our python notebook/file, without having to create a local server/containerized system and without having to store and use a cloud-based system. You can sign up for additional premium Weviate services that provide these features. However, given our data in this project is a manageable size and can be easily processed each time we access the file, it's helpful to have a lighter install option in the 'embedded' option.

For additional code and installation, instructions see:
https://weaviate.io/developers/weaviate/installation/embedded

Other options that could be considered for a vector store could be Pinecone and Meta's FAISS service:

https://ai.meta.com/tools/faiss/ (Works well with Meta's Llama LLMs)

https://www.pinecone.io/product/ (Pitched as a 'production ready' Weaviate. Weaviate is open source)

IV. Langchain Model Orchestration

https://www.langchain.com/

Langchain provides an orchestration framework for building 'agentic' workflows, RAG models and deploying and managing LLMs generally. Langchain works out of the package from within your Python notebook, without a need to sign up for a service directly or pay.

RAG Model: Technical How to Guide

Within the working code for the project, the following Langchain functions are used:

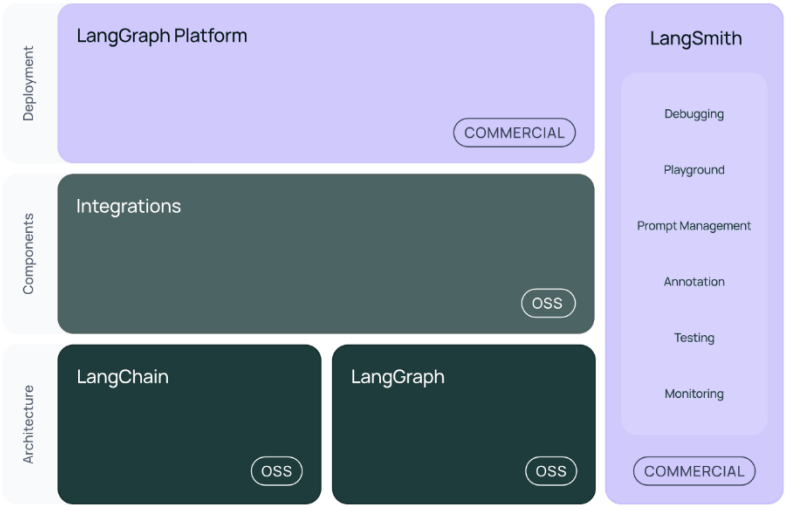
Function:	Purpose:	Example Code:
PyPDFLoader: from langchain.document_loaders import PyPDFLoader	Imports a single pdf and splits into pages to create a set of docs.	data_load=PyPDFLoader('data/ed3bookaug20_2024.pdf') data_test=data_load.load_and_split()
PyPDFDirectoryLoader: from langchain_community.document_loaders import PyPDFDirectoryLoader	Imports a full directory/folder of pdfs in a single operation. Includes document meta data for referencing	loader = PyPDFDirectoryLoader("data/") docs = loader.load()
Recursive Character Text Splitter: from langchain.text_splitter import RecursiveCharacterTextSplitter	Splits texts into chunks of managable size to feed into Embeddings model	dir_split=RecursiveCharacterTextSplitter(separators=["\n\n","\n", " ", ""], chunk_size=500, chunk_overlap=50) dir_chunks = dir_split.split_documents(docs)
Chat Prompt Template: from langchain.prompts import ChatPromptTemplate	Provides a set of instructions to the LLM as an input that accompanies the direct question being asked to the LLM	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. Provide specific text references. Use three sentences maximum and keep the answer concise. Question: {question} Context: {context} Answer: """ prompt = ChatPromptTemplate.from_template(template) print(prompt)
Create the RAG Chain: from langchain_openai import ChatOpenAI	Create the retrieval augmented generation mechanism	llm = ChatOpenAI(model_name="gpt-3.5-turbo", temperature=0) rag_chain = (

from langchain.schema.runnable import RunnablePassthrough from langchain.schema.output_parser import StrOutputParser		{"context": retriever, "question": RunnablePassthrough()} | prompt | llm | StrOutputParser())

Note that for the project, I chose to use the RecursiveCharacterTextSplitter over the basic TextSplitter, also available through Langchain. The TextSplitter relies on fixed character lengths and overlap settings to split text into chunks. The RecursiveCharacterTextSplittter, divides the text iteratively and in a hierarchical manner based on the chosen separators ("\n\n", "\n", " ","" are standard). The advantage of the recursive approach is that more natural break points in the text may be selected, vs. the strict character level approach in the basic splitter, which creates more artificial ccharacter-basedcut points.

For the project implementation, a target character length of 500 and a 50 character overlap between chunks was selected and appeared to perform well in the RAG model tests.

In addition to base Langchain architecture, there are several scaled services that are offered commercially to support deployment and management of the Langchain suite of tools:



V. Data Sets Used for Retrieval:

To support the Retrieval Augmented Generation model, I used the following course materials to develop documents and embeddings to support the RAG prompt model:

- **Course Syllabus**
- **Lecture Notes:** Up to Lecture 8, with the potential to retrain later in the course.
- **Textbooks:**

- "Speech and Language Processing An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition with Language Models" Third Edition draft. DANIEL JURAFSKY, Stanford University and JAMES H. MARTIN, University of Colorado at Boulder, https://web.stanford.edu/~jurafsky/slp3/ed3bookaug20_2024.pdf, accessed 11/10/2024
- Deep Learning with Python FRANÇOIS CHOLLET, ISBN 9781617294433, 2021, https://sourestdeeds.github.io/pdf/Deep%20Learning%20with%20Python.pdf, accessed 11/10/2024

- **Additional Materials:**
  - stm: An R Package for Structural Topic Models MARGARET E. ROBERTS University of California, San Diego BRANDON M. STEWART Princeton University DUSTIN TINGLEY Harvard University, Journal of Statistical Software October 2019, Volume 91, Issue 2. doi: 10.18637/jss.v091.i02

An initial import of these texts into python yields the following number of pages and characters by source:

Source	Number Pages	Thousand Characters
Lecture Notes (Kurochkin)	259	132.007
Speech and Language Processing (Jurafsky.Martin)	599	1807.905
Deep Learning (Chollet)	504	1121.421
stm Vignette (Roberts, Steward, Tingley)	40	97.78
Course Syllabus (Kurochkin)	11	16.764

VI. Project Wrap-up

The biggest technical hurdles of the project was updating the packages that work together best to access key features of Langchain, Open-AI, and Weaviate. Given the pace of advancement in the industry these packages seem to change frequently, with sample code bases from even the beginning of 2024 requiring updates to packages and references. Fortunately, the error tracing provided useful instructions in package updates, depreciation warnings and required code changes.

Bibliography: Reference Sources for RAG Implementation:

- Monigatti, Leonie; "Retrieval-Augmented Generation (RAG): From Theory to LangChain Implementation", November 14, 2023, https://towardsdatascience.com/retrieval-augmented-generation-rag-from-theory-to-langchain-implementation-4e9bd5f6a4f2, accessed 12/5/2024
- Santhosh, Roshan**;** "Building a RAG chain using LangChain Expression Language (LCEL)"; April 11, 2024**;** https://towardsdatascience.com/building-a-rag-chain-using-langchain-expression-language-lcel-3688260cad05, accessed 12/5/2024
- Trisal, Rahul, 'Build a HR QnA App using Retrieval Augmentation Generation (RAG), AWS Bedrock, FAISS, Langchain', https://www.youtube.com/watch?v=aNj-1wehoEo&t=0s, accessed 12/5/2024
- Choi, Judy; Ragas Tutorial ihttps://github.com/Judy-Choi/ragas_tutorial/blob/main/ragas_tutorial.ipynb, accessed 12/5/2024
- Amazon Bedrock SDK Examples, aws-doc-sdk-examples/python/example_code/bedrock at main · awsdocs/aws-doc-sdk-examples · GitHub, accessed 12/5/2024
- Thippireddy, Bharath; "Setup OpenAI API Key"; https://www.youtube.com/watch?v=wpL7z8rYShg; accessed 12/5/2024
- Mishra, Anurag; "Five Levels of Chunking Strategies in RAG| Notes from Greg's Video"; Jan 14, 2024; https://medium.com/@anuragmishra_27746/five-levels-of-chunking-strategies-in-rag-notes-from-gregs-video-7b735895694d; Accessed 12/5/2024
- Stoll , Markus, "Visualize your RAG Data — Evaluate your Retrieval-Augmented Generation System with Ragas"; Towards Data Science; Mar 3, 2024 https://towardsdatascience.com/visualize-your-rag-data-evaluate-your-retrieval-augmented-generation-system-with-ragas-fc2486308557; Accessed 12/12/2024