HW: LLMs, vectors, RAG :)

Summary

In this final HW, you will:

- use Weaviate [https://weaviate.io/], which is a vector DB - stores data as vectors after vectorizing, and computes a search query by vectorizing it and does similarity search with existing vectors
- crawl the web using a Node package, to compile a 'knowledge base' [to use subsequently (not part of the hw) as input to build a custom GPT (!)]
- using a Python module, perform RAG [retrieval augmentation] on a 'small', locally-hosted LLM [make that an 'S'LM :)]
- use https://lightning.ai to run RAG on their CPU+GPU platform

These are cutting-edge techniques to know, from a future/career POV :) Plus, they are simply, FUN!!

Please make sure you have these installed, before starting: git, Docker, Node, Python (or Conda/Anaconda), VS 2022 [with 'Desktop development with C++' checked].

Note: you need to do all four, Q1..Q4 (not pick just one!) :)

Q1.

Description

We are going to use vector-based similarity search, to retrieve search results that are not keyword-driven.

The (three) steps we need are really simple:

- install Weaviate plus vectorizer via Docker as images, run them as containers
- specify a schema for data, upload data/knowledge (in .json format) to have it be vectorized
- run a query (which also gets vectorized and then sim-searched), get back results (as JSON)

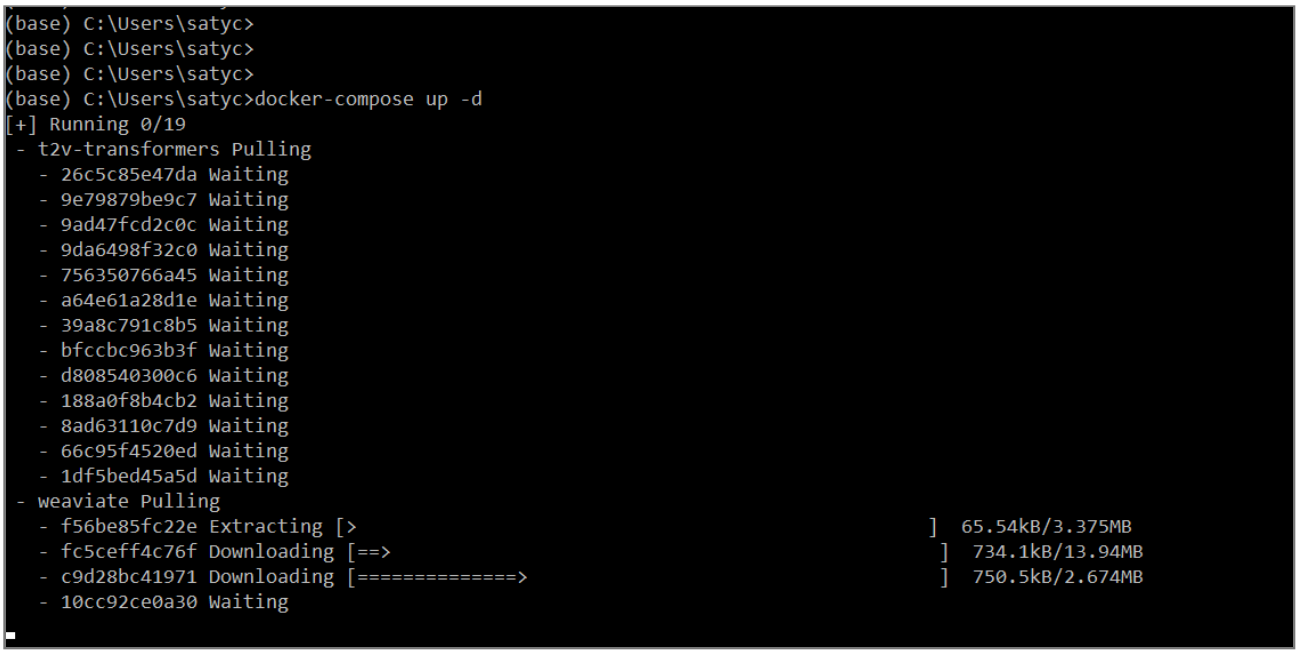
The following sections describe the above steps.

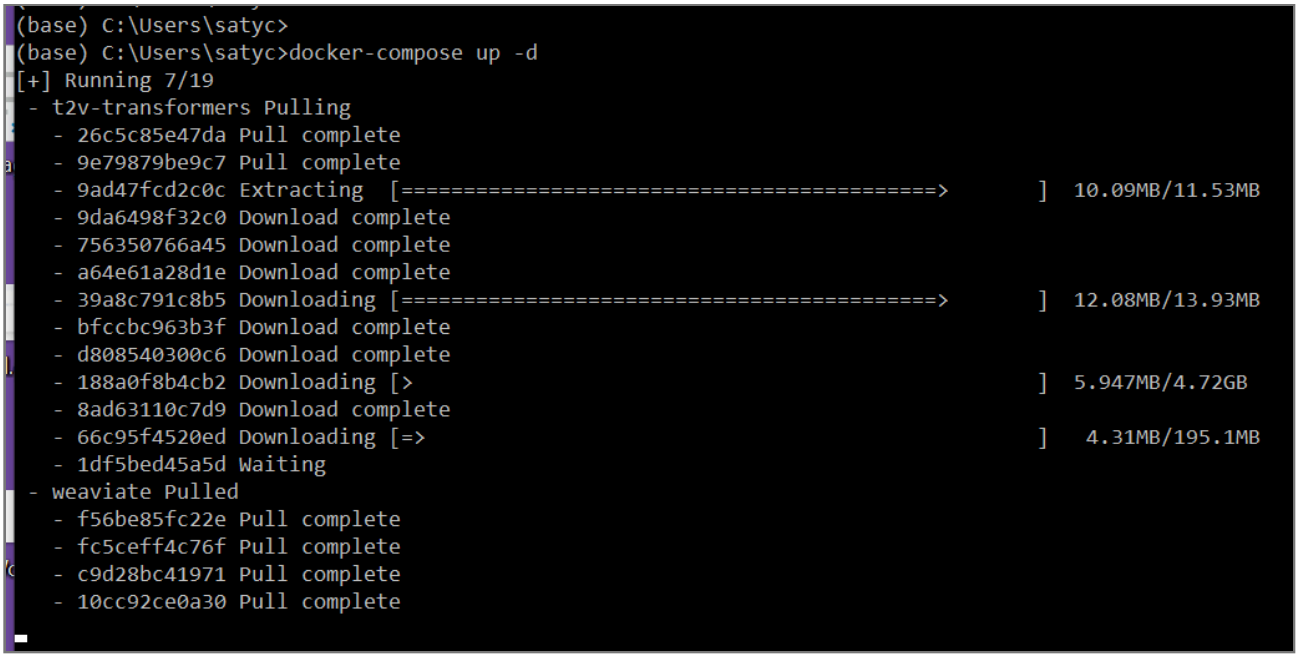
1. Installing Weaviate and a vectorizer module

After installing Docker, bring it up (eg. on Windows, run Docker Desktop). Then, in your (ana)conda shell, run this docker-compose command that uses this yaml 'docker-compose.yml' config file to pull in two images: the 'weaviate' one, and a text2vec transformer called 't2v-transformers':

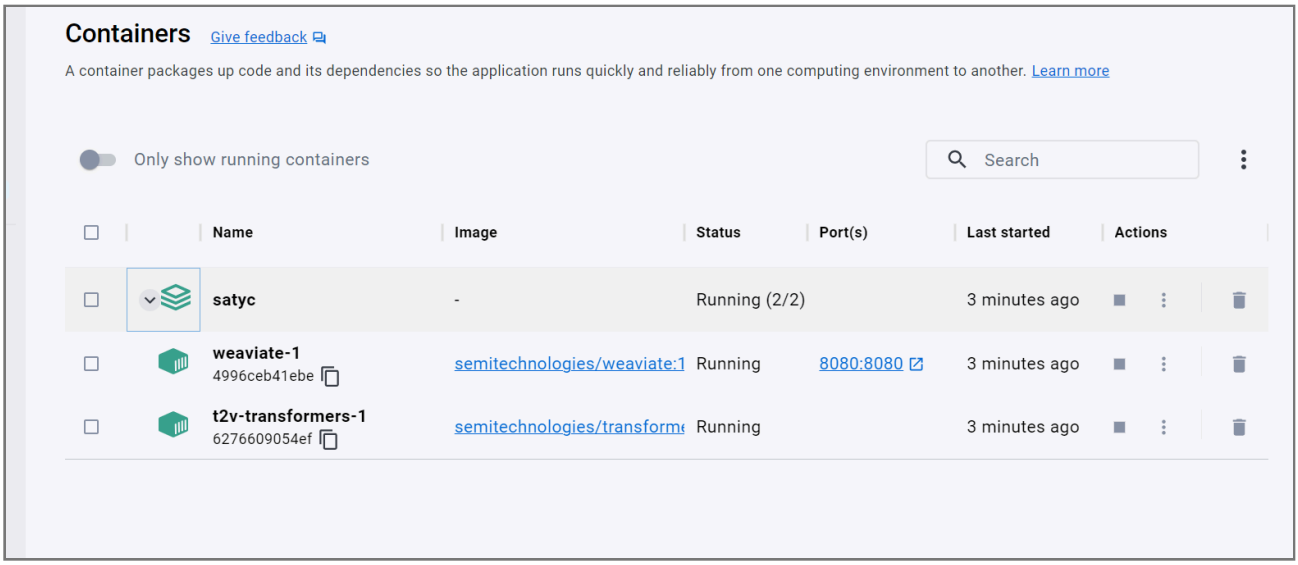
```
docker-compose up -d
```

These screenshots show the progress, completion, and subsequently, two containers automatically being started (one for weaviate, one for t2v-transformers):





```
(base) C:\Users\satyc>docker-compose up -d
[+] Running 19/19
 - t2v-transformers Pulled                                                    1450.6s
   - 26c5c85e47da Pull complete                                                39.2s
   - 9e79879be9c7 Pull complete                                                39.7s
   - 9ad47fcd2c0c Pull complete                                                41.5s
   - 9da6498f32c0 Pull complete                                                41.7s
   - 756350766a45 Pull complete                                                42.6s
   - a64e61a28d1e Pull complete                                                43.3s
   - 39a8c791c8b5 Pull complete                                                46.0s
   - bfccbc963b3f Pull complete                                                47.9s
   - d808540300c6 Pull complete                                                48.0s
   - 188a0f8b4cb2 Pull complete                                              1423.4s
   - 8ad63110c7d9 Pull complete                                              1424.9s
   - 66c95f4520ed Pull complete                                              1442.9s
   - 1df5bed45a5d Pull complete                                              1443.5s
 - weaviate Pulled                                                             28.5s
   - f56be85fc22e Pull complete                                                 3.5s
   - fc5ceff4c76f Pull complete                                                23.0s
   - c9d28bc41971 Pull complete                                                23.6s
   - 10cc92ce0a30 Pull complete                                                23.8s
[+] Running 3/3
 - Network satyc_default              Created                                  1.2s
 - Container satyc-weaviate-1         Started                                 10.7s
 - Container satyc-t2v-transformers-1 Started                                 10.7s
```



Yeay! Now we have the vectorizer transformer (to convert sentences to vectors), and weaviate (our vector DB search engine) running! On to data handling :)

2. Loading data to search for

This is the data (knowledge, aka external memory, ie. prompt augmentation source) that we'd like searched, part of which will get returned to us as results. The data is represented as an array of JSON documents. Here is our data file, conveniently named data.json (you can rename it if you like) [you can visualize it better using https://jsoncrack.com] - place it in the 'root' directory of your webserver (see below). As you can see, each datum/'row'/JSON contains three k:v pairs, with 'Category', 'Question', 'Answer' as keys - as you might guess, it seems to be in Jeopardy(TM) answer-question (reversed) format :) The file is actually called jeopardy-tiny.json, I simply made a local copy called data.json.

The overall idea is this: we'd get the 10 documents vectorized, then specify a query word, eg. 'biology', and automagically have that pull up related docs, eg. the 'DNA' one (even if the search result doesn't contain 'biology' in it)! This is a really useful **semantic** search feature where we don't need to specify exact keywords to search for.

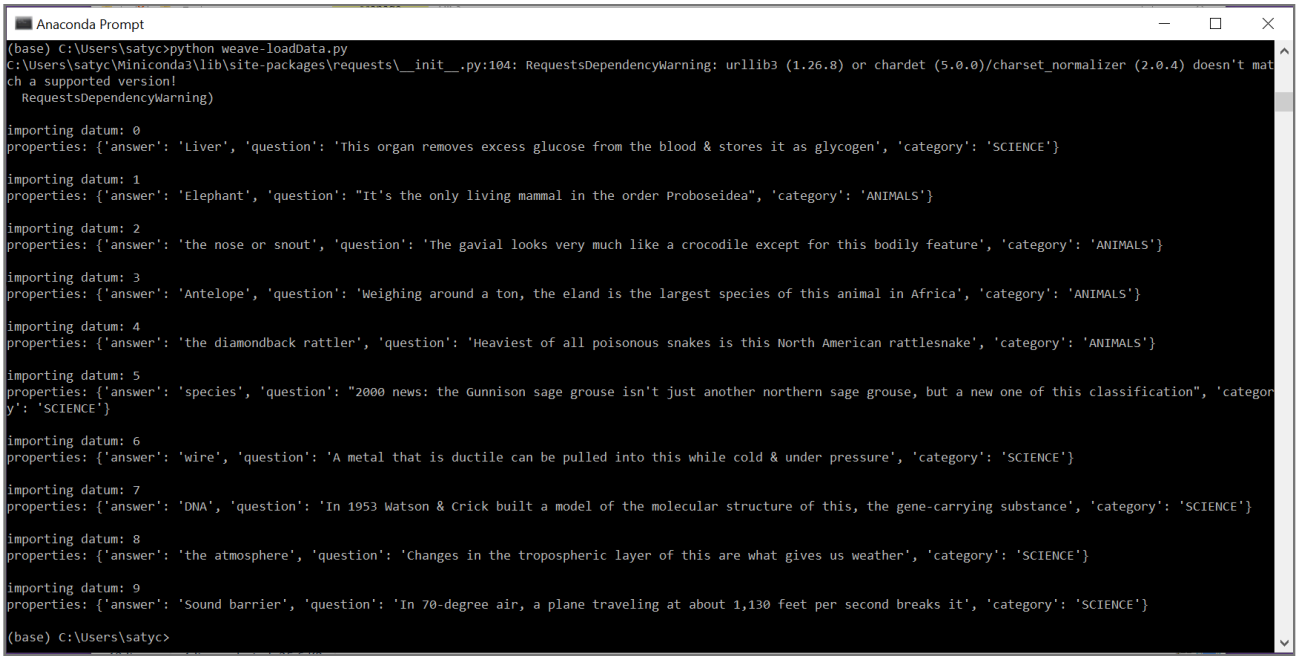
Start by installing the weaviate Python client:

```
pip install weaviate-client
```

So, how to submit our JSON data, to get it vectorized? Simply use this Python script, do:

```
python weave-loadData.py
```

You will see this:



If you look in the script, you'll see that we are creating a schema - we create a class called 'SimSearch' (you can call it something else if you like). The data we load into the DB, will be associated with this class (the last line in the script does this via add_data_object()).

NOTE - **you NEED to run a local webserver** [in a separate ana/conda (or other) shell], eg. via python 'serveit.py' - it's what will 'serve' data.json to weaviate :)

Great! Now we have specified our searchable data, which has been first vectorized (by 't2v-transformers'), then stored as vectors (in weaviate).

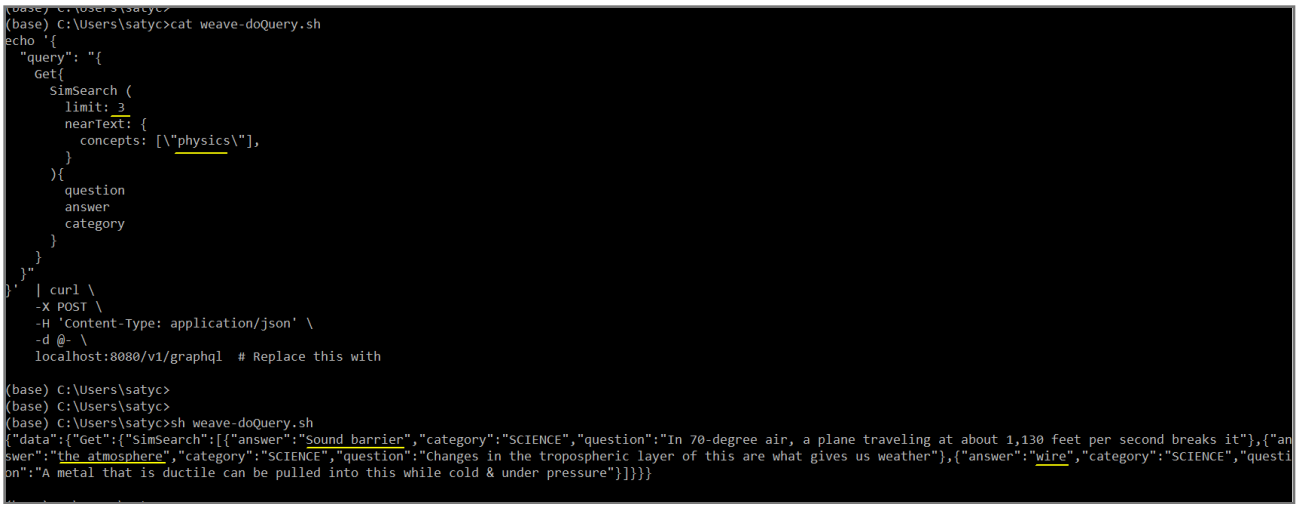
Only one thing left: querying!

3. Querying our vectorized data

To query, use this simple shell script called weave-doQuery.sh, and run this:

```
sh weave-doQuery.sh
```

As you can see in the script, we search for 'physics'-related docs, and sure enough, that's what we get:



Why is this exciting? Because the word 'physics' isn't in any of our results!

Now it's your turn:

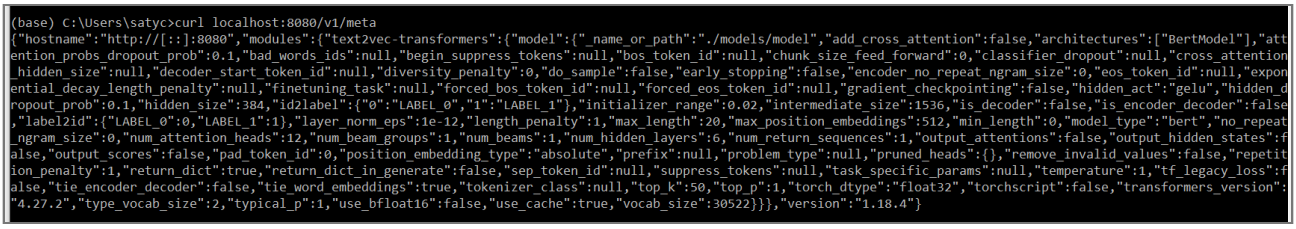
• first, MODIFY the contents of data.json, to replace the 10 docs in it, with your own data, where you'd replace ("Category","Question","Answer") with ANYTHING you like, eg. ("Author","Book","Summary"), ("MusicGenre","SongTitle","Artist"), ("School","CourseName","CourseDesc"), etc, etc - HAVE fun coming up with this! You can certainly add more docs, eg. have 20 of them instead of 10

• next, MODIFY the query keyword(s) in the query .sh file - eg. you can query for 'computer science' courses, 'female' singer, 'American' books, ['Indian','Chinese'] food dishes (the query list can contain multiple items), etc. Like in the above screenshot, 'cat' the query, then run it, and get a screenshot to submit. BE SURE to also modify the data loader .py script, to put in your keys (instead of ("Category","Question","Answer"))

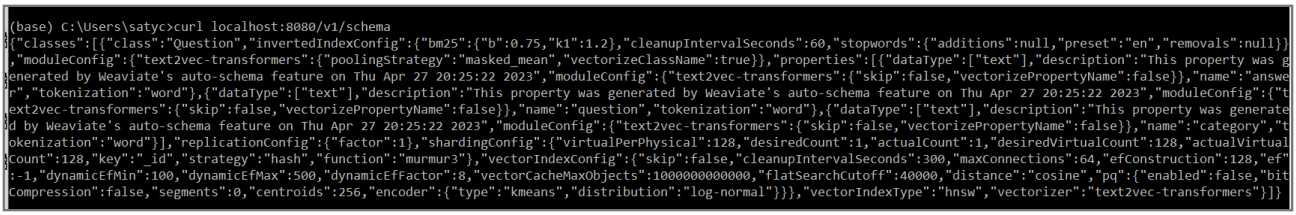
That's it, you're done :) In RL you will have a .json or .csv file (or data in other formats) with BILLIONS of items! Later, do feel free to play with bigger JSON files, eg. this 200K Jeopardy JSON file :)

FYI/'extras'

Here are two more things you can do, via 'curl':

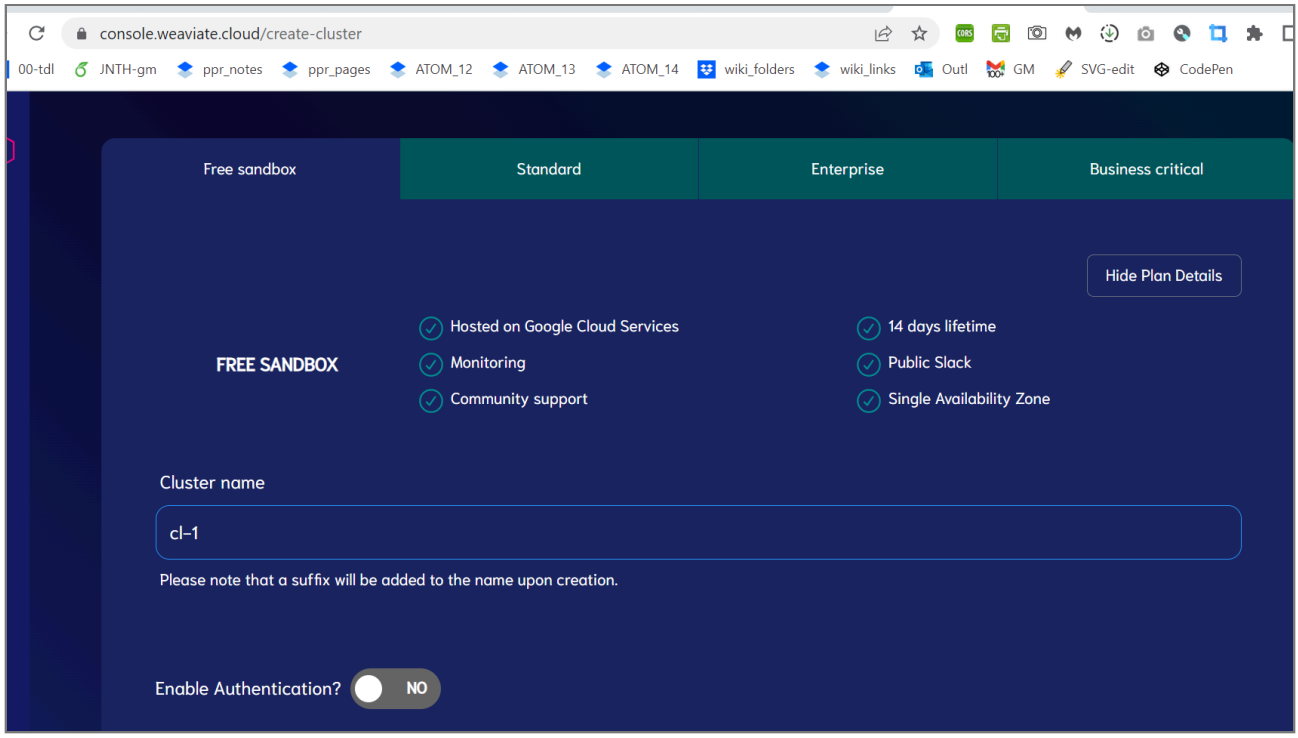


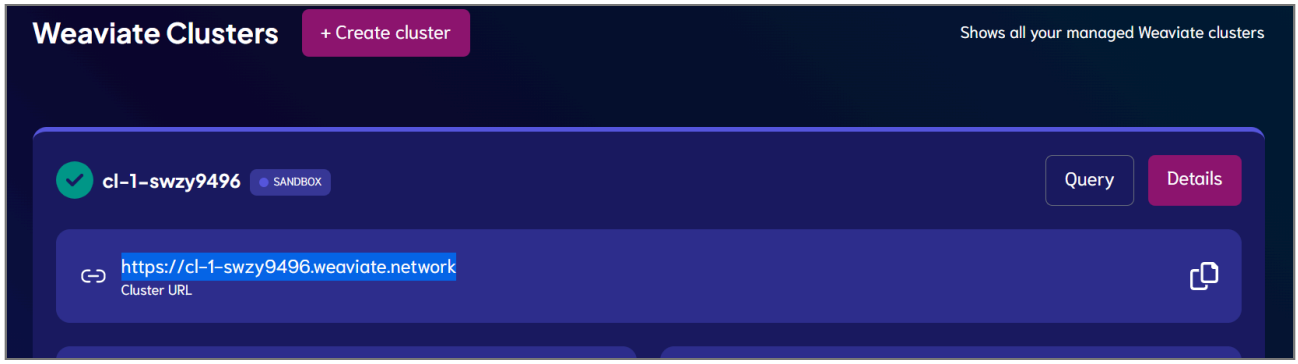
[you can also do 'http://localhost:8080/v1/meta' in your browser]



[you can also do 'http://localhost:8080/v1/schema' in your browser]

Weaviate has a cloud version too, called WCS - you can try that as an alternative to using the Dockerized version:





Run this :)

Also, for fun, see if you can print the raw vectors for the data (the 10 docs)...

More info:

- https://weaviate.io/developers/weaviate/quickstart/end-to-end
- https://weaviate.io/developers/weaviate/installation/docker-compose
- https://medium.com/semi-technologies/what-weaviate-users-should-know-about-docker-containers-1601c6afa079
- https://weaviate.io/developers/weaviate/modules/retriever-vectorizer-modules/text2vec-transformers

---

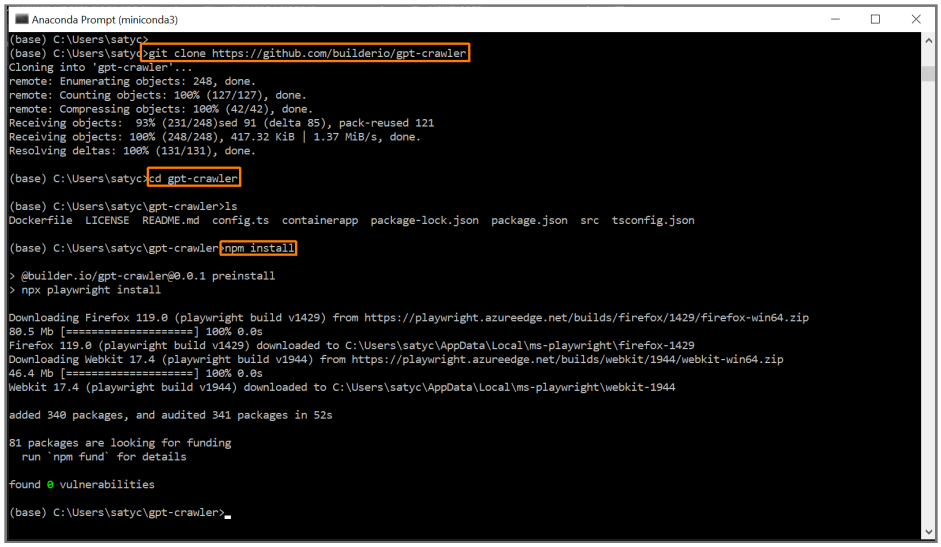
Q2.

You are going to run a crawler on a set of pages that you know contain 'good' data - that could be used by an LLM to answer questions 'intelligently' (ie. not confabulate, ie not 'hallucinate', ie. not make up BS based on its core, general-purpose pre-training!).

The crawled results get conveniently packaged into a single output.json file. For this qn, please specify what group of pages you

crawled [you can pick any that you like], and, submit your output.json (see below for how to generate it).

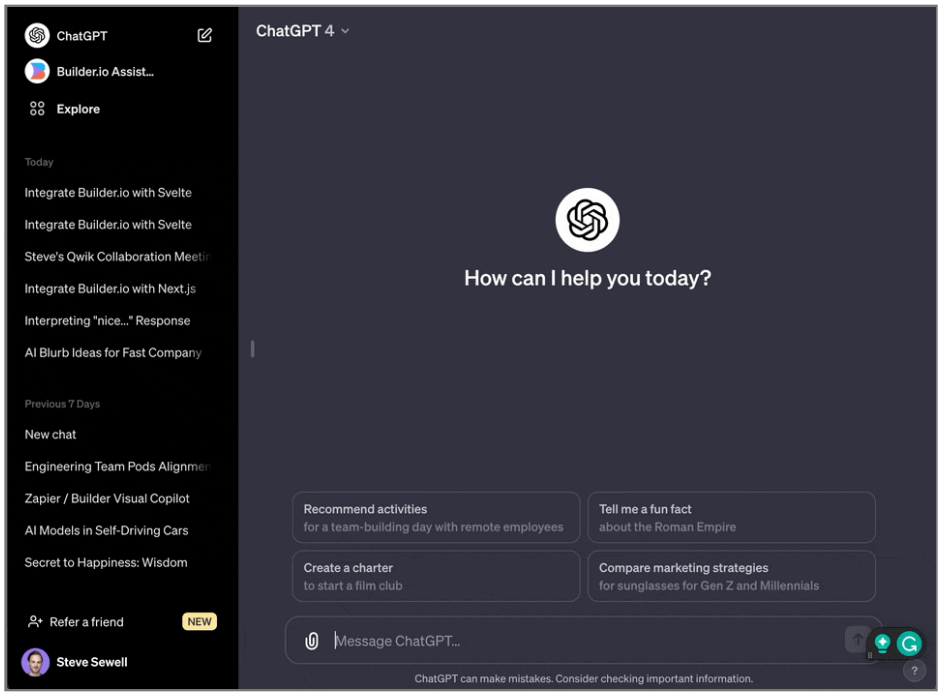
Take a look:



You'll need to git-clone 'gpt-crawler' from https://github.com/BuilderIO/gpt-crawler. Then do 'npm install' to download the needed Node packages. Then edit config.ts [https://github.com/BuilderIO/gpt-crawler/blob/main/config.ts] to specify your crawl path, then simply run the crawler via npm.start! Voila - a resulting output.json, after the crawling is completed.

For this hw, you'll simply submit your output.json - but its true purpose is to serve as input for a cstom GPT :)

From builder.io's GitHub page:

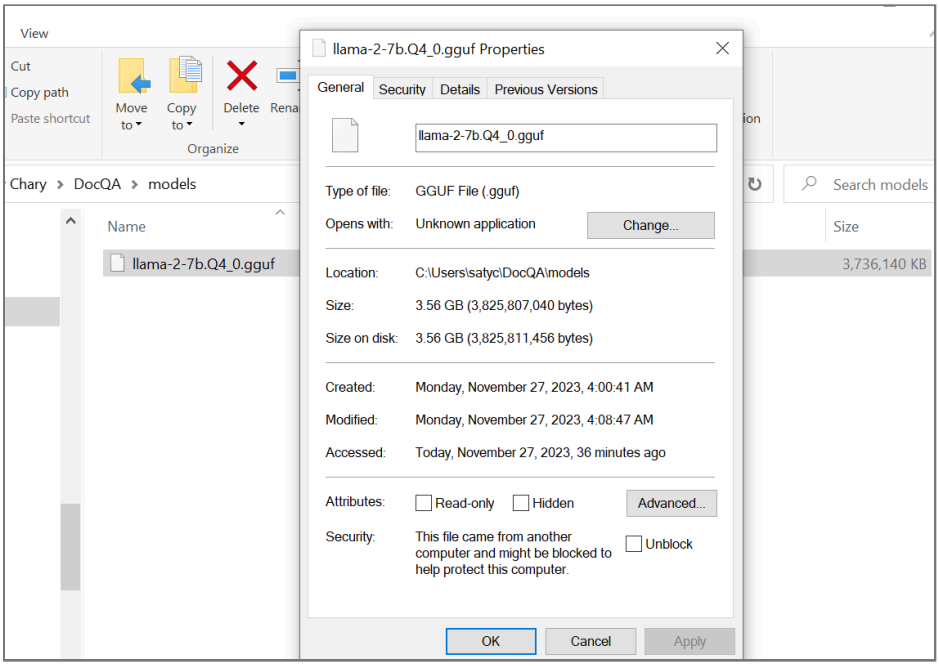


Amazing! You can use this to create all sorts of SMEs [subject matter experts] in the future, by simply scraping **existing** docs on the web.

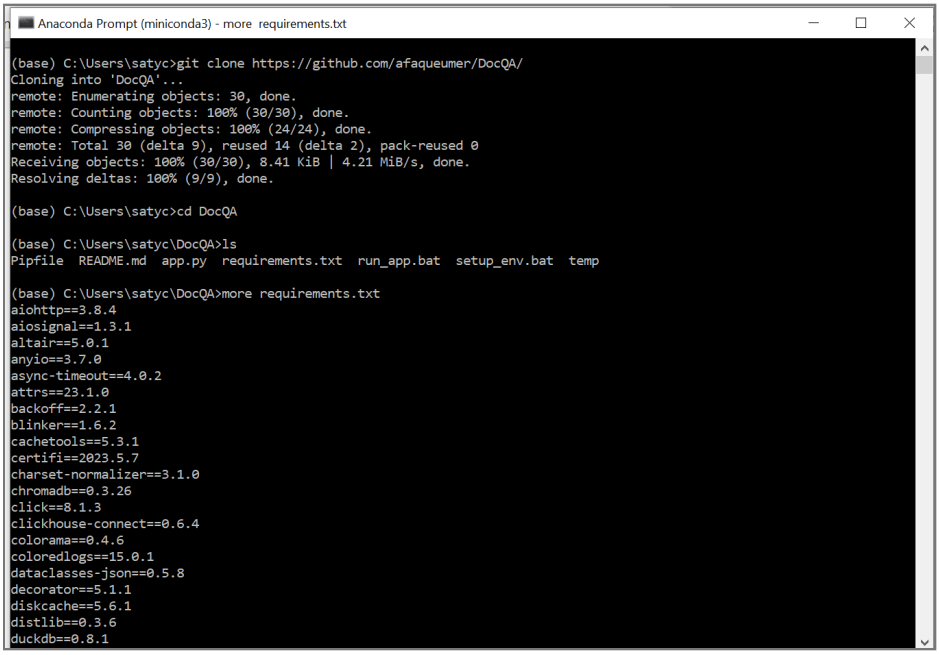
Q3.

For this question, you are going to download a small (3.56G) model (with 7B parameters, compared to GPT-4's 1T for ex!), and use it along with an external knowledge source (a simple text file) vectorized using Chroma (a popular vector DB), and ask questions

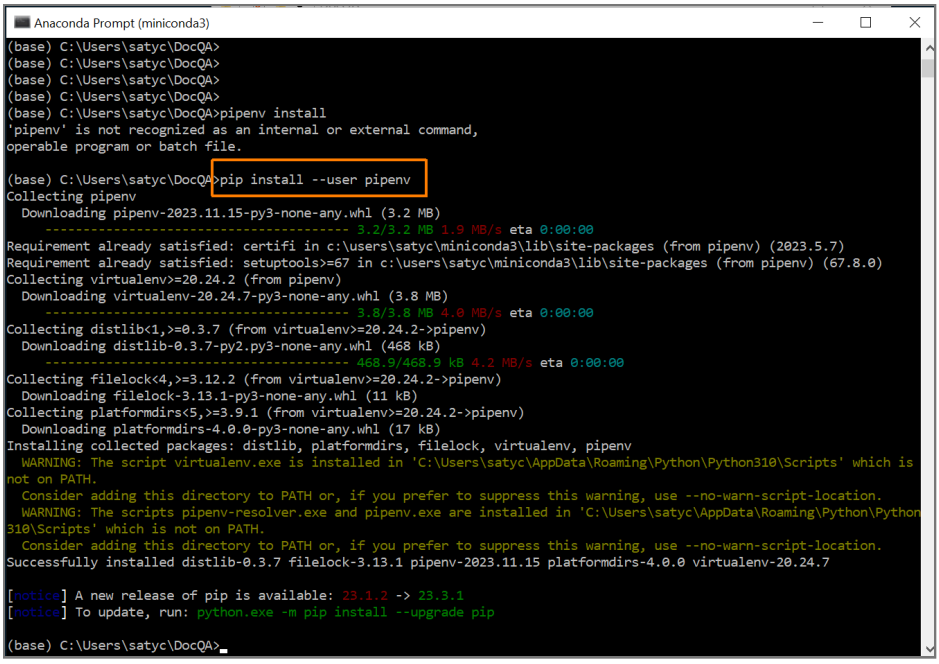
whose answers would be found in the text file :) Fun!



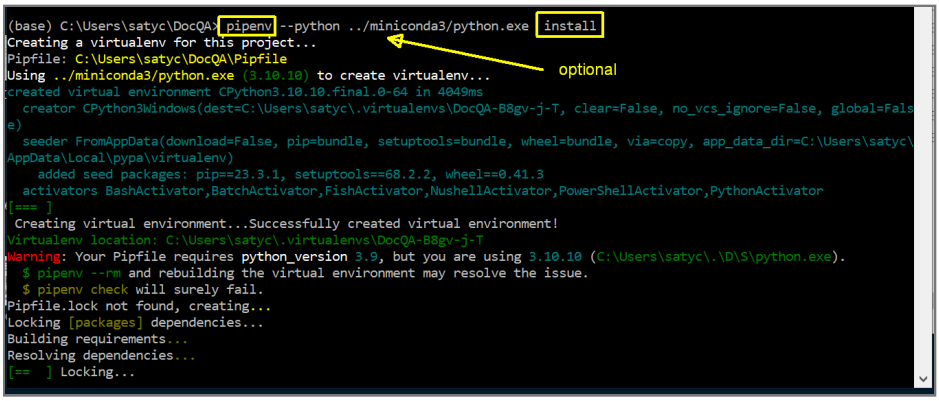
git clone this: https://github.com/afaqueumer/DocQA - and cd into it. You'll see a Python script (app.py) and a requirements.txt file.



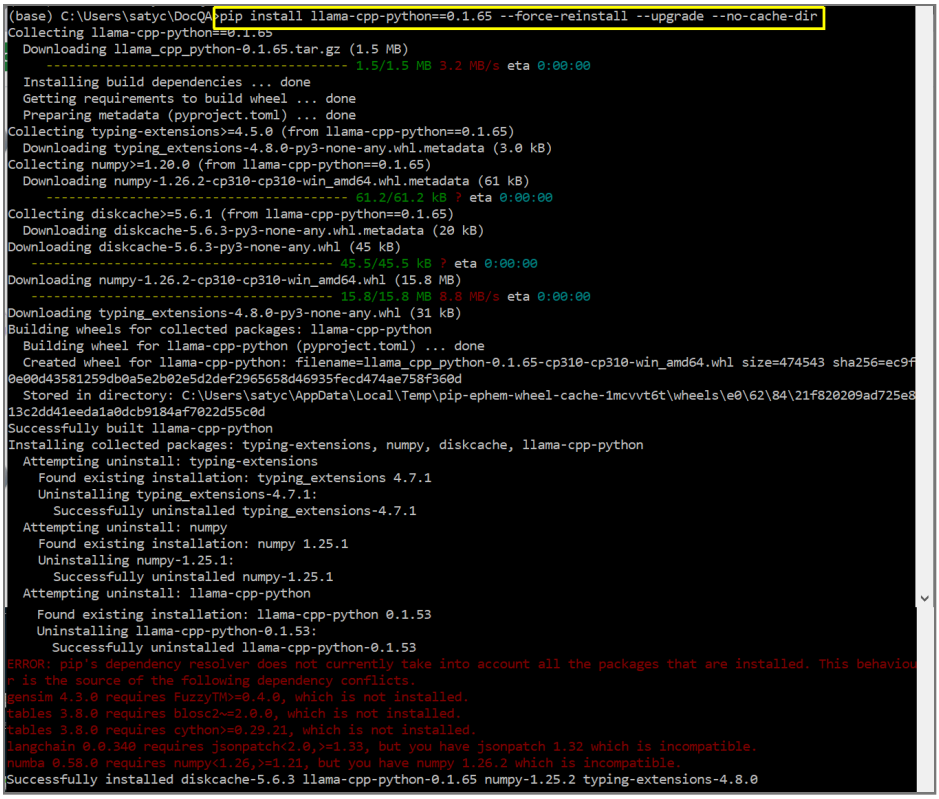
Install pipenv:



Install the required components (Chroma, LangChain etc) like so:

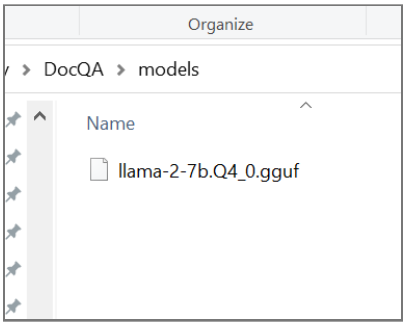


Turns out we need a newer version of llama-cpp-python, one of the modules we just installed - so do this:

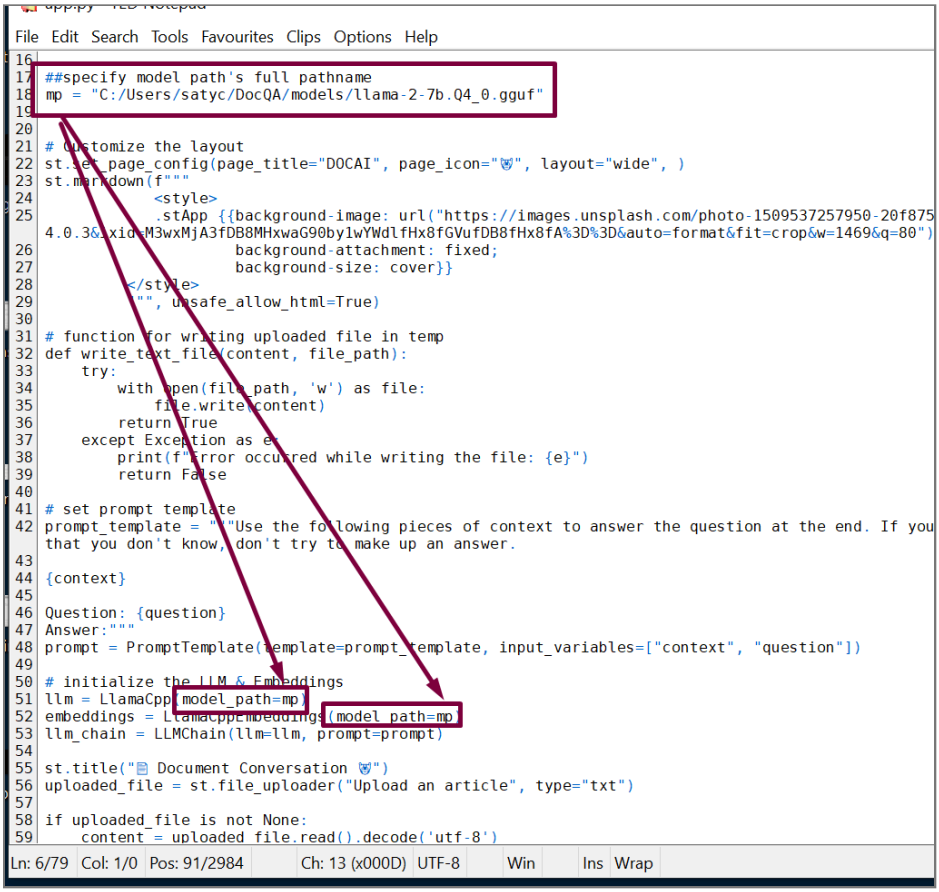


Next, let's grab this LLM: https://huggingface.co/TheBloke/Llama-2-7B-GGUF/blob/main/llama-2-7b.Q4_0.gguf - and save it to a

models/ folder inside your DocQA one:



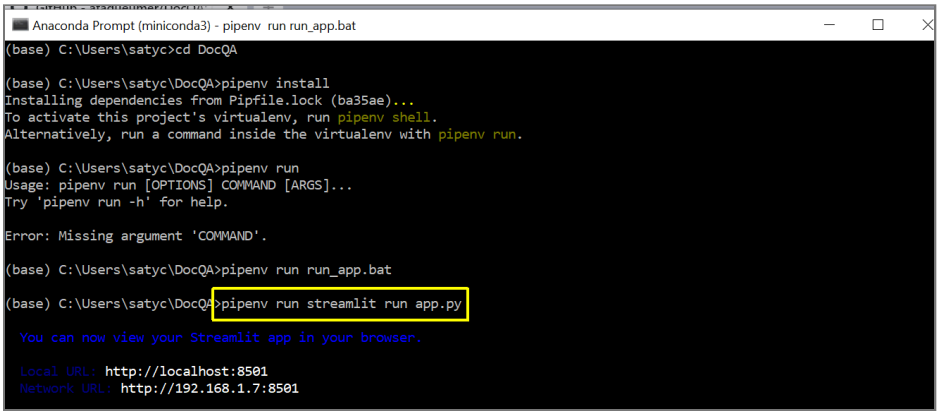
Modify app.py to specify this LLM:



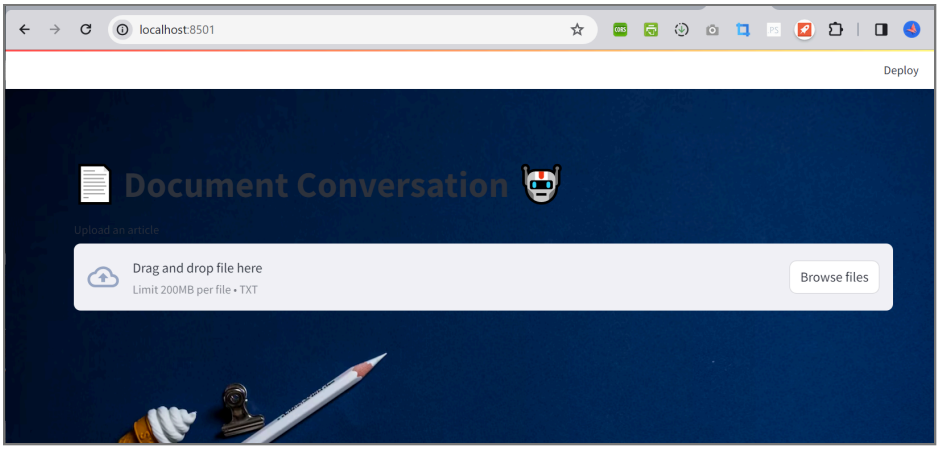
If you are curious about the .gguf format used to specify the LLM, read this.

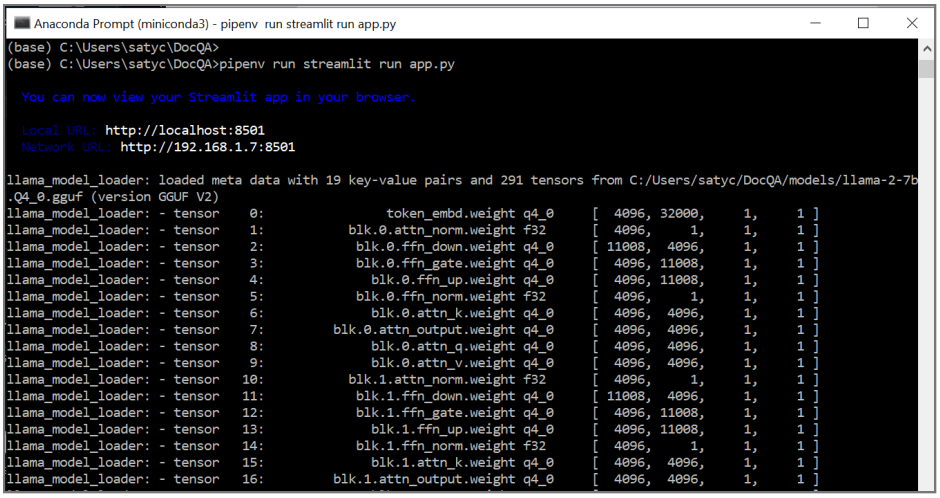
Now we have all the pieces! These include the req'd Python modules, the LLM, and an app.py that will launch a UI via 'Streamlit' . Run

this [pipenv run streamlit run app.py]:



OMG - our chat UI in a browser, via a local webserver [the console prints info about the LLM]:

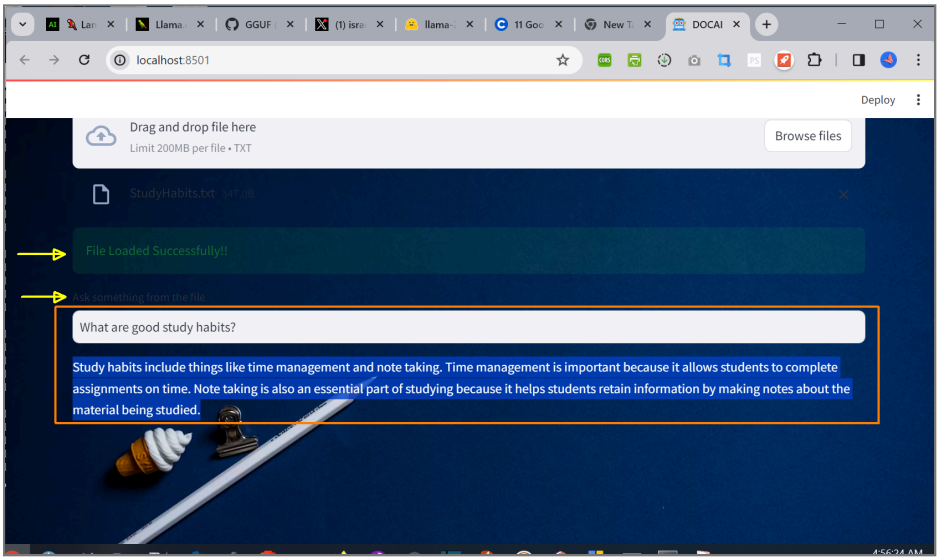




Now we need a simple text file to use for asking questions from (ie. 'external memory' for the LLM). I used https://www.coursera.org/articles/study-habits page, to make this file, for ex.

We are now ready to chat with our doc! Upload the .txt, wait a few minutes for the contents to get vectorized and indexed :) When

that is done, **ask a question - and get an answer!** Like so:



That's quite impressive!

You would need to create a text file of your own [you could even type in your own text, about anything!], upload, ask a question, then get a screenshot of the Q and A. You'd submit the text file and the screenshot.

The above is what the new 'magic' (ChatGPT etc) is about!! Later, you can try out many other models, other language tasks, reading PDF, etc. Such custom 'agents' are sure to become commonplace, serving/dispensing expertise/advice in myriad areas of life.

Here is more, related to Q3.

Q4.

This is a quick, easy and useful one!

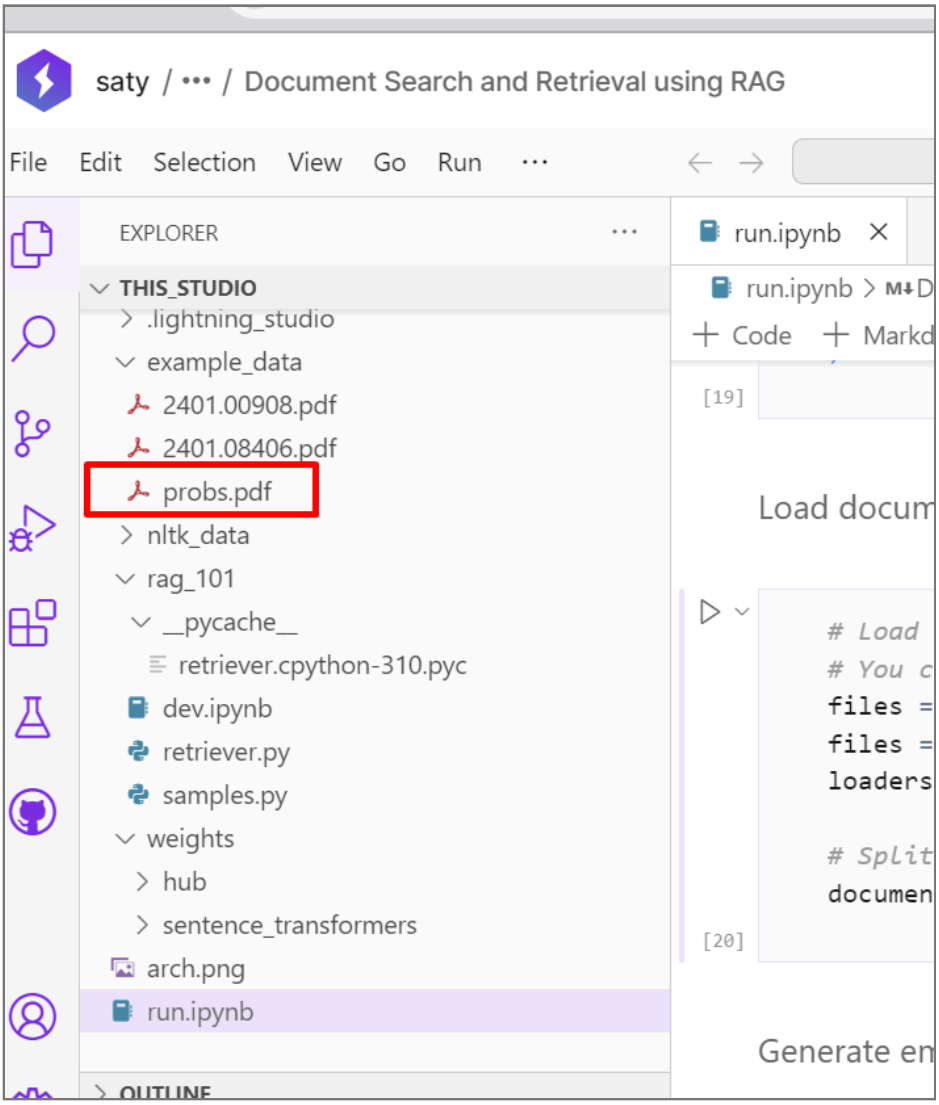
Go to https://lightning.ai/ and sign up for a free account. Then read these: https://lightning.ai/docs/overview/getting-started and https://lightning.ai/docs/overview/getting-started/studios-in-10-minutes

Browse through their vast collection of 'Studio' templates: https://lightning.ai/studios - when you create (instantiate) one, you get your own sandboxed environment [a 'cloud supercomputer'] that runs on lightning.ai's servers. You get unlimited CPU use, and 22 hours of GPU use per month (PLENTY, for beginner projects).

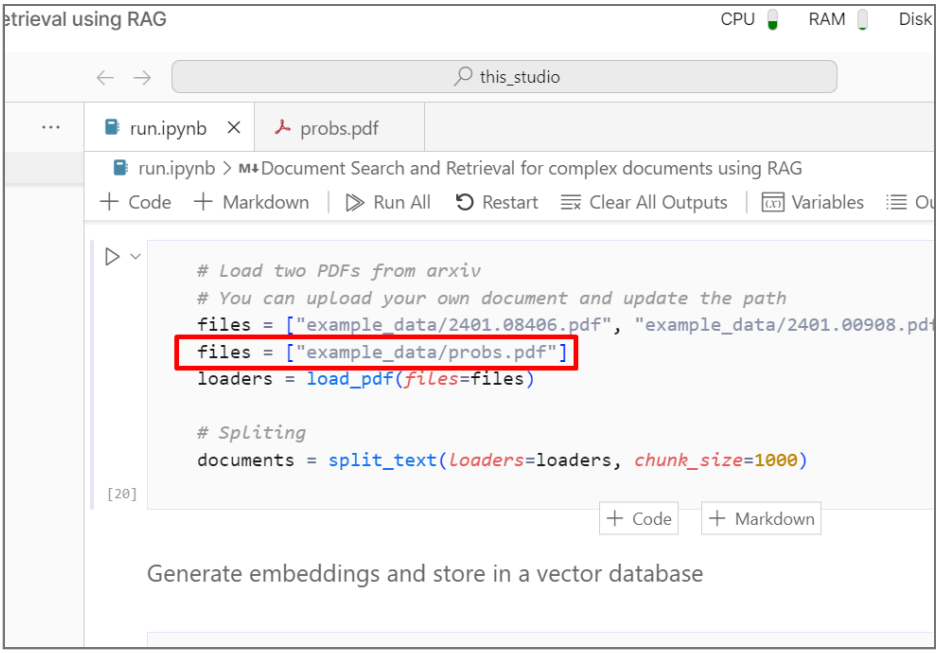
Create this Studio: https://lightning.ai/lightning-ai/studios/document-search-and-retrieval-using-rag - you are going to use this to do RAG using your own PDF :)

Upload (drag and drop) your PDF [can be on ANY topic - coding, cooking, crafting, canoeing, cattle-ranching, catfishing... (lol!)].

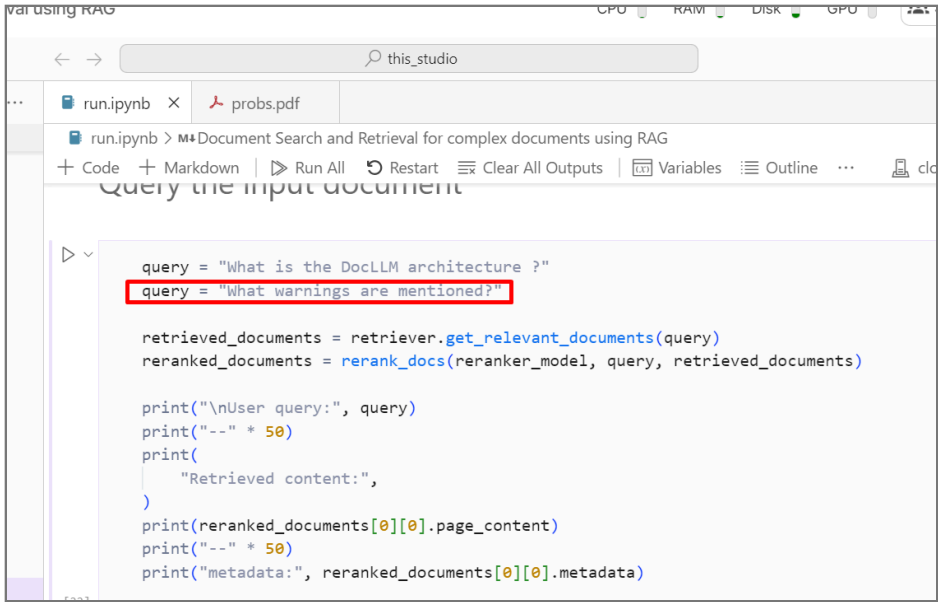
Eg. this shows the pdf I uploaded:



Next, edit run.ipynb, modify the 'files' variable to point to your pdf:



Modify the 'query' var, and the 'queries' var, to each contain a QUESTION on which you'd like to do RAG, ie. get the answers from the pdf you uploaded! THIS is the cool part - to be able to **ask questions in natural language**, rather than search by keyword, or look up words in the index [if the pdf has an index].



Read through the notebook to understand what the code does and what the RAG architecture is, then **run the code**! You'll see the two answers printed out. Submit screenshots of your pdf file upload, the two questions, and the two answers. The answers might not be what you expected (ie might be imprecise, EVEN though it's RAG!) but that's ok - there are techniques to improve the quality of the retrieval, you can dive into them later.

After the course, DO make sure to run (almost) all the templates! It's painless (zero installation!), fast (GPU execution!) and useful/informative (well-documented!). It doesn't get more cutting edge than these, for 'IR'. You can even write and run your own code in a Studio, and publish it as a template for others to use :)

Getting help

There is a hw4 'forum' on Piazza, for you to post questions/answers. You can also meet w/ the TAs, CPs, or me.

Have fun! This is a really useful piece of tech to know. **Vector DBs are sure be used more and more in the near future, as a way to provide 'infinite external runtime memory' (augmentation) for pretrained LLMs.** Read this too: https://www.linkedin.com/pulse/complete-guide-vector-databases-yugank-aman-nlcef/