HW: LLMs, vectors, RAG :)

Summary

In this 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
- use lightning.ai [https://lightning.ai] to run RAG on lightning's 'GPU-accelerated cloud VM' platform
- use HuggingFace [https://huggingface.co/] to run a RAG application and describe its components

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

Q1. (4 points)

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)

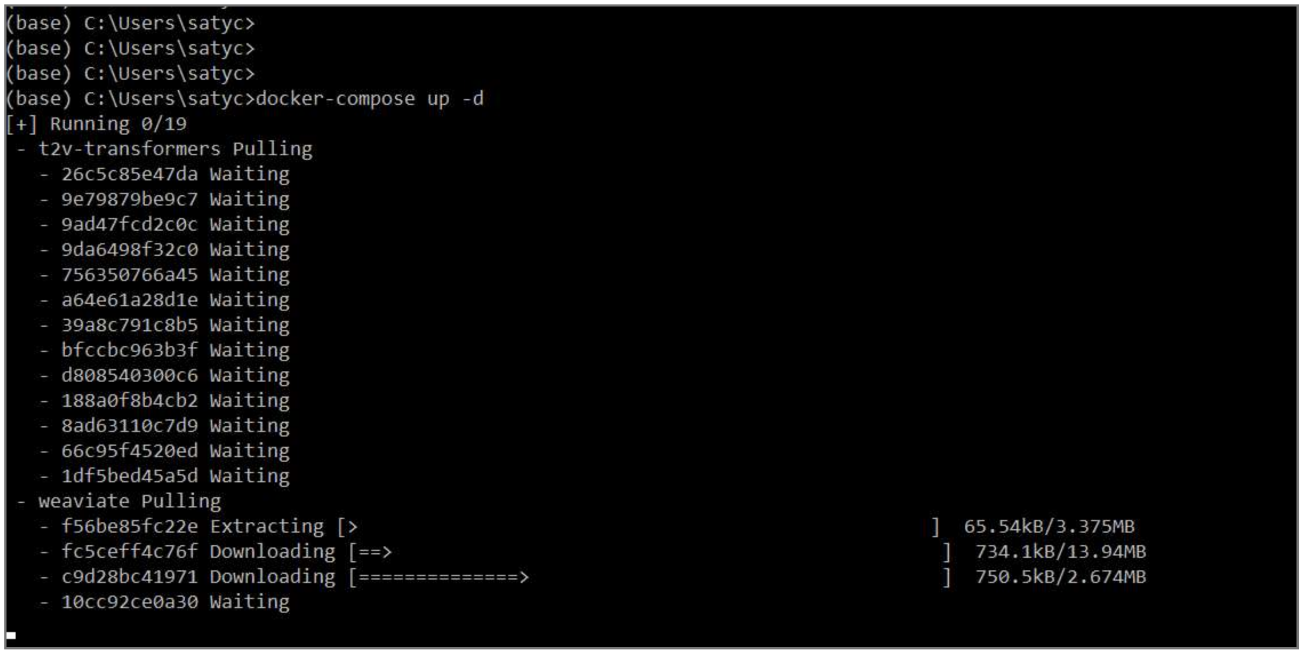
The following sections describe the above steps.

1. Installing Weaviate and a vectorizer module

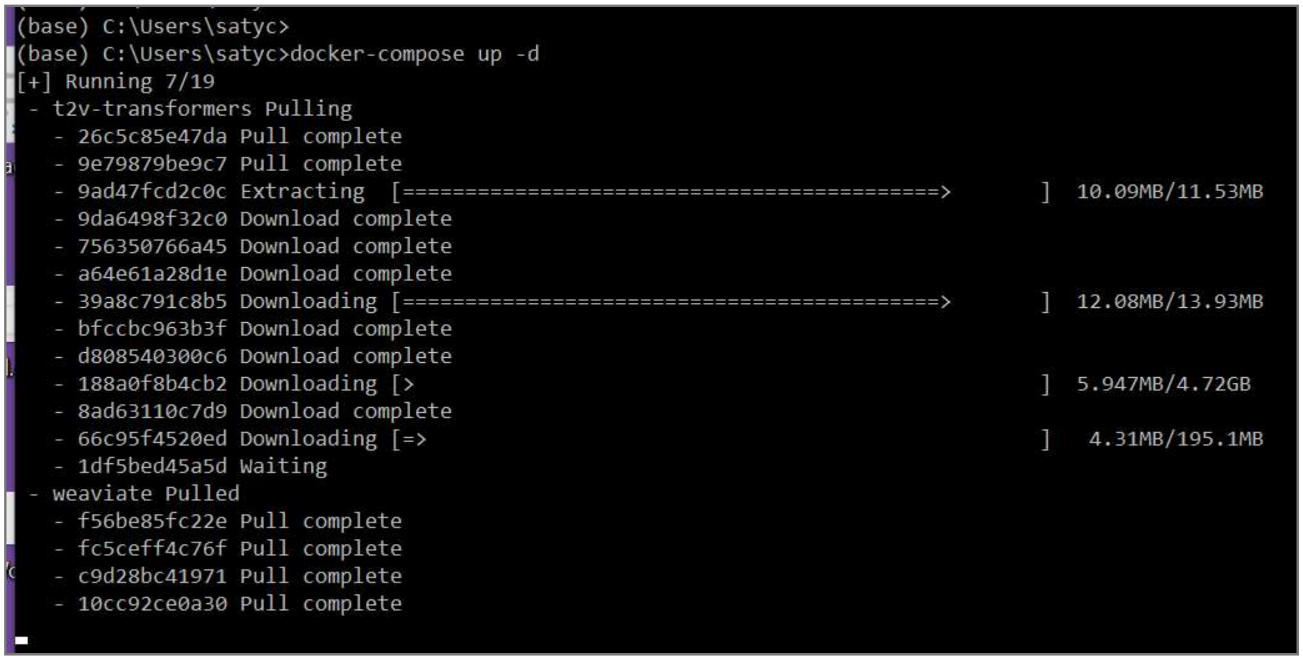
After installing Docker, bring it up (eg. on Windows, run Docker Desktop). Then, in your (ana)conda shell, run this docker-compose command that uses this yaml 'docker-compose.yml' config file to pull in two images: the 'weaviate' one, and a text2vec transformer called 't2v-transformers':

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

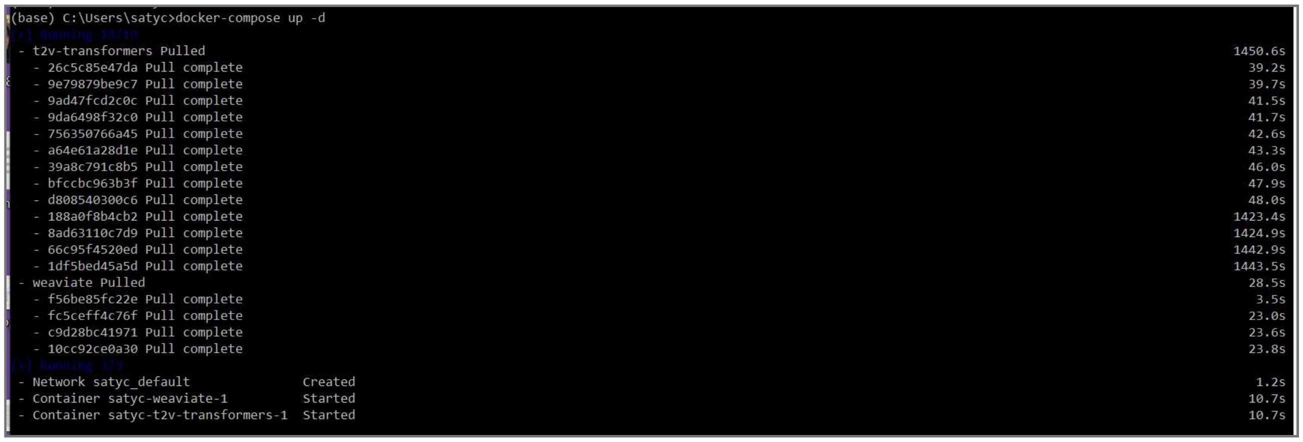
These screenshots show the progress, completion, and subsequently, two containers automatically being started (one for weaviate, one for t2v-transformers):



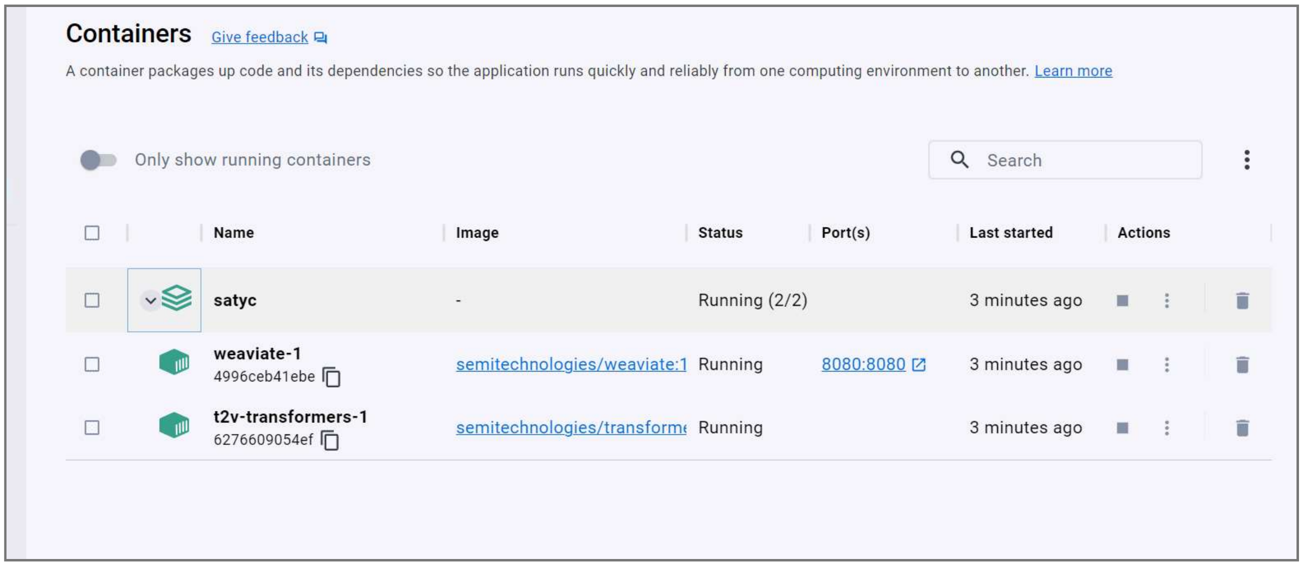
```
(base) C:\Users\satyc>
(base) C:\Users\satyc>
(base) C:\Users\satyc>
(base) C:\Users\satyc>docker-compose up -d
[+] Running 0/19
 - t2v-transformers Pulling
   - 26c5c85e47da Waiting
   - 9e79879be9c7 Waiting
   - 9ad47fcd2c0c Waiting
   - 9da6498f32c0 Waiting
   - 756350766a45 Waiting
   - a64e61a28d1e Waiting
   - 39a8c791c8b5 Waiting
   - bfccbc963b3f Waiting
   - d808540300c6 Waiting
   - 188a0f8b4cb2 Waiting
   - 8ad63110c7d9 Waiting
   - 66c95f4520ed Waiting
   - 1df5bed45a5d Waiting
 - weaviate Pulling
   - f56be85fc22e Extracting [>                                                  ]  65.54kB/3.375MB
   - fc5ceff4c76f Downloading [==>                                                ]  734.1kB/13.94MB
   - c9d28bc41971 Downloading [==============>                                    ]  750.5kB/2.674MB
   - 10cc92ce0a30 Waiting
```



```
(base) C:\Users\satyc>
(base) C:\Users\satyc>docker-compose up -d
[+] Running 7/19
 - t2v-transformers Pulling
   - 26c5c85e47da Pull complete
   - 9e79879be9c7 Pull complete
   - 9ad47fcd2c0c Extracting  [===========================================>       ]  10.09MB/11.53MB
   - 9da6498f32c0 Download complete
   - 756350766a45 Download complete
   - a64e61a28d1e Download complete
   - 39a8c791c8b5 Downloading [===========================================>       ]  12.08MB/13.93MB
   - bfccbc963b3f Download complete
   - d808540300c6 Download complete
   - 188a0f8b4cb2 Downloading [>                                                  ]  5.947MB/4.72GB
   - 8ad63110c7d9 Download complete
   - 66c95f4520ed Downloading [=>                                                 ]   4.31MB/195.1MB
   - 1df5bed45a5d Waiting
 - weaviate Pulled
   - f56be85fc22e Pull complete
   - fc5ceff4c76f Pull complete
   - c9d28bc41971 Pull complete
   - 10cc92ce0a30 Pull complete
```



```
(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
```



Containers Give feedback

A container packages up code and its dependencies so the application runs quickly and reliably from one computing environment to another. Learn more

Name	Image	Status	Port(s)	Last started	Actions
satyc	-	Running (2/2)		3 minutes ago	
weaviate-1 4996ceb41ebe	semitechnologies/weaviate:1	Running	8080:8080	3 minutes ago	
t2v-transformers-1 6276609054ef	semitechnologies/transform	Running		3 minutes ago	

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.

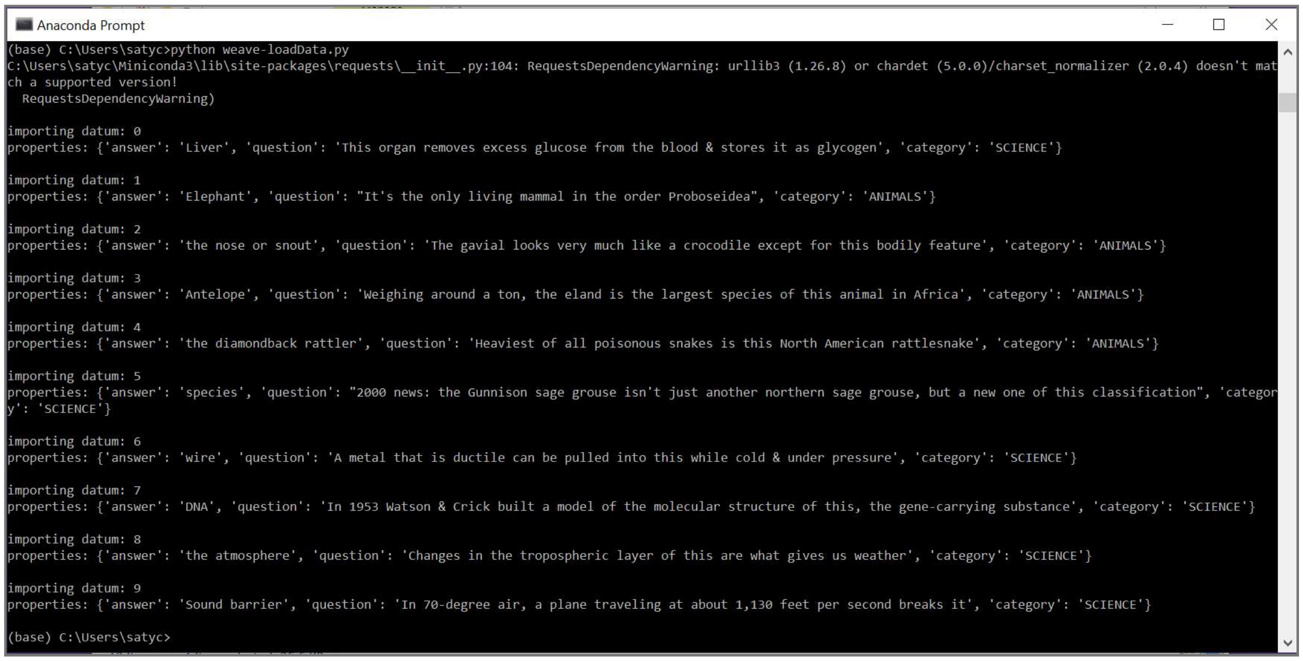
Start by installing the weaviate Python client:

```
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:



```
(base) C:\Users\satyc>python weave-loadData.py
C:\Users\satyc\Miniconda3\lib\site-packages\requests\__init__.py:104: RequestsDependencyWarning: urllib3 (1.26.8) or chardet (5.0.0)/charset_normalizer (2.0.4) doesn't mat
ch a supported version!
  RequestsDependencyWarning)

importing datum: 0
properties: {'answer': 'Liver', 'question': 'This organ removes excess glucose from the blood & stores it as glycogen', 'category': 'SCIENCE'}

importing datum: 1
properties: {'answer': 'Elephant', 'question': "It's the only living mammal in the order Proboseidea", 'category': 'ANIMALS'}

importing datum: 2
properties: {'answer': 'the nose or snout', 'question': 'The gavial looks very much like a crocodile except for this bodily feature', 'category': 'ANIMALS'}

importing datum: 3
properties: {'answer': 'Antelope', 'question': 'Weighing around a ton, the eland is the largest species of this animal in Africa', 'category': 'ANIMALS'}

importing datum: 4
properties: {'answer': 'the diamondback rattler', 'question': 'Heaviest of all poisonous snakes is this North American rattlesnake', 'category': 'ANIMALS'}

importing datum: 5
properties: {'answer': 'species', 'question': "2000 news: the Gunnison sage grouse isn't just another northern sage grouse, but a new one of this classification", 'categor
y': 'SCIENCE'}

importing datum: 6
properties: {'answer': 'wire', 'question': 'A metal that is ductile can be pulled into this while cold & under pressure', 'category': 'SCIENCE'}

importing datum: 7
properties: {'answer': 'DNA', 'question': 'In 1953 Watson & Crick built a model of the molecular structure of this, the gene-carrying substance', 'category': 'SCIENCE'}

importing datum: 8
properties: {'answer': 'the atmosphere', 'question': 'Changes in the tropospheric layer of this are what gives us weather', 'category': 'SCIENCE'}

importing datum: 9
properties: {'answer': 'Sound barrier', 'question': 'In 70-degree air, a plane traveling at about 1,130 feet per second breaks it', 'category': 'SCIENCE'}

(base) C:\Users\satyc>
```

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).

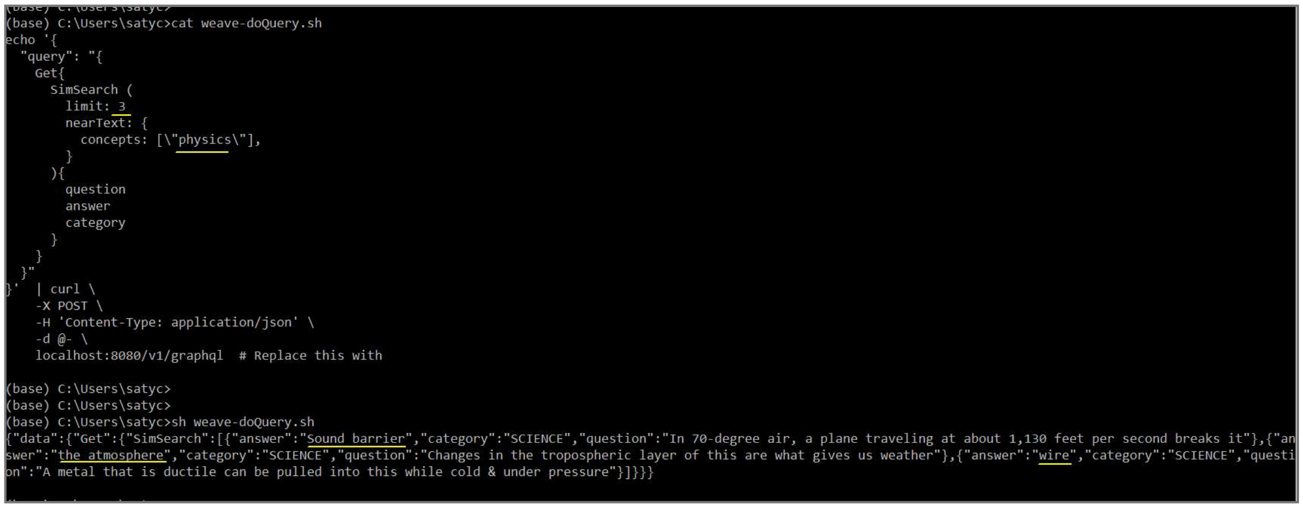
Only one thing left: querying!

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:



```
(base) C:\Users\satyc>cat weave-doQuery.sh
echo '{
  "query": "{
    Get{
      SimSearch (
        limit: 3
        nearText: {
          concepts: [\"physics\"],
        }
      ){
        question
        answer
        category
      }
    }
  }"
}'  | curl \
    -X POST \
    -H 'Content-Type: application/json' \
    -d @- \
    localhost:8080/v1/graphql  # Replace this with

(base) C:\Users\satyc>
(base) C:\Users\satyc>
(base) C:\Users\satyc>sh weave-doQuery.sh
{"data":{"Get":{"SimSearch":[{"answer":"Sound barrier","category":"SCIENCE","question":"In 70-degree air, a plane traveling at about 1,130 feet per second breaks it"},{"an
swer":"the atmosphere","category":"SCIENCE","question":"Changes in the tropospheric layer of this are what gives us weather"},{"answer":"wire","category":"SCIENCE","questi
on":"A metal that is ductile can be pulled into this while cold & under pressure"}]}}}
```

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

Now it's your turn:

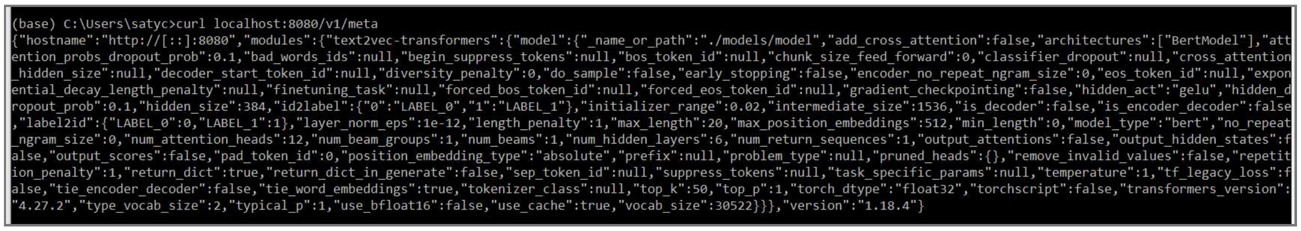
· first, MODIFY the contents of data.json, to replace the 10 docs in it, with your own data, where you'd replace ("Category","Question","Answer") with ANYTHING you like, eg. ("Author","Book","Summary"), ("MusicGenre","SongTitle","Artist"), ("School","CourseName","CourseDesc"), etc, etc - HAVE fun coming 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 :)

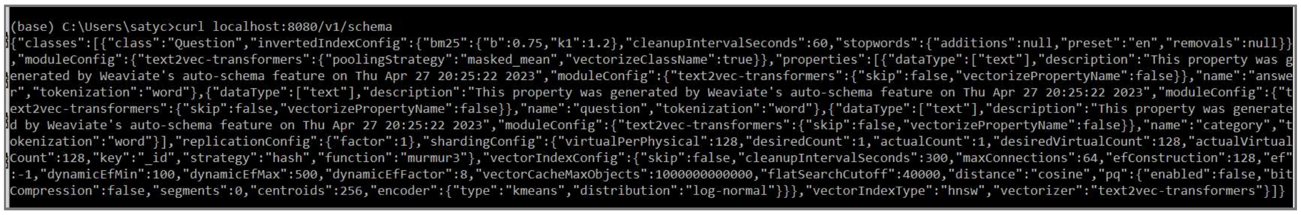
FYI/'extras' [for you to pursue later]

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



```
(base) C:\Users\satyc>curl localhost:8080/v1/meta
{"hostname":"http://[::]:8080","modules":{"text2vec-transformers":{"model":{"_name_or_path":"./models/model","add_cross_attention":false,"architectures":["BertModel"],"att
ention_probs_dropout_prob":0.1,"bad_words_ids":null,"begin_suppress_tokens":null,"bos_token_id":null,"chunk_size_feed_forward":0,"classifier_dropout":null,"cross_attention
_hidden_size":null,"decoder_start_token_id":null,"diversity_penalty":0,"do_sample":false,"early_stopping":false,"encoder_no_repeat_ngram_size":0,"eos_token_id":null,"expon
ential_decay_length_penalty":null,"finetuning_task":null,"forced_bos_token_id":null,"forced_eos_token_id":null,"gradient_checkpointing":false,"hidden_act":"gelu","hidden_d
ropout_prob":0.1,"hidden_size":384,"id2label":{"0":"LABEL_0","1":"LABEL_1"},"initializer_range":0.02,"intermediate_size":1536,"is_decoder":false,"is_encoder_decoder":false
,"label2id":{"LABEL_0":0,"LABEL_1":1},"layer_norm_eps":1e-12,"length_penalty":1,"max_length":20,"max_position_embeddings":512,"min_length":0,"model_type":"bert","no_repeat
_ngram_size":0,"num_attention_heads":12,"num_beam_groups":1,"num_beams":1,"num_hidden_layers":6,"num_return_sequences":1,"output_attentions":false,"output_hidden_states":f
alse,"output_scores":false,"pad_token_id":0,"position_embedding_type":"absolute","prefix":null,"problem_type":null,"pruned_heads":{},"remove_invalid_values":false,"repetit
ion_penalty":1,"return_dict":true,"return_dict_in_generate":false,"sep_token_id":null,"suppress_tokens":null,"task_specific_params":null,"temperature":1,"tf_legacy_loss":f
alse,"tie_encoder_decoder":false,"tie_word_embeddings":true,"tokenizer_class":null,"top_k":50,"top_p":1,"torch_dtype":"float32","torchscript":false,"transformers_version":
"4.27.2","type_vocab_size":2,"typical_p":1,"use_bfloat16":false,"use_cache":true,"vocab_size":30522}}},"version":"1.18.4"}
```

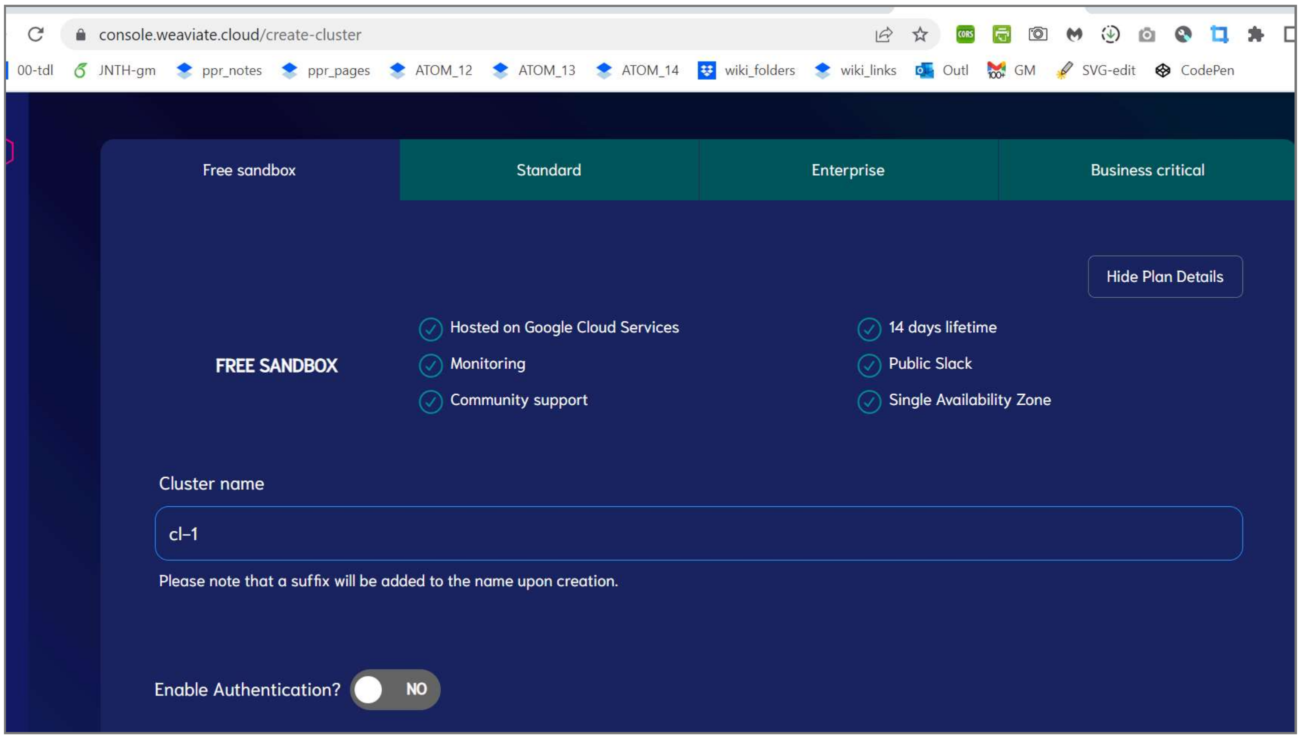
[you can also do 'http://localhost:8080/v1/meta' in your browser]



```
(base) C:\Users\satyc>curl localhost:8080/v1/schema
{"classes":[{"class":"Question","invertedIndexConfig":{"bm25":{"b":0.75,"k1":1.2},"cleanupIntervalSeconds":60,"stopwords":{"additions":null,"preset":"en","removals":null}}
,"moduleConfig":{"text2vec-transformers":{"poolingStrategy":"masked_mean","vectorizeClassName":true}},"properties":[{"dataType":["text"],"description":"This property was g
enerated by Weaviate's auto-schema feature on Thu Apr 27 20:25:22 2023","moduleConfig":{"text2vec-transformers":{"skip":false,"vectorizePropertyName":false}},"name":"answe
r","tokenization":"word"},{"dataType":["text"],"description":"This property was generated by Weaviate's auto-schema feature on Thu Apr 27 20:25:22 2023","moduleConfig":{"t
ext2vec-transformers":{"skip":false,"vectorizePropertyName":false}},"name":"question","tokenization":"word"},{"dataType":["text"],"description":"This property was generate
d by Weaviate's auto-schema feature on Thu Apr 27 20:25:22 2023","moduleConfig":{"text2vec-transformers":{"skip":false,"vectorizePropertyName":false}},"name":"category","t
okenization":"word"}],"replicationConfig":{"factor":1},"shardingConfig":{"virtualPerPhysical":128,"desiredCount":1,"actualCount":1,"desiredVirtualCount":128,"actualVirtual
Count":128,"key":"_id","strategy":"hash","function":"murmur3"},"vectorIndexConfig":{"skip":false,"cleanupIntervalSeconds":300,"maxConnections":64,"efConstruction":128,"ef"
:-1,"dynamicEfMin":100,"dynamicEfMax":500,"dynamicEfFactor":8,"vectorCacheMaxObjects":1000000000000,"flatSearchCutoff":40000,"distance":"cosine","pq":{"enabled":false,"bit
Compression":false,"segments":0,"centroids":256,"encoder":{"type":"kmeans","distribution":"log-normal"}}},"vectorIndexType":"hnsw","vectorizer":"text2vec-transformers"}]}
```

[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:



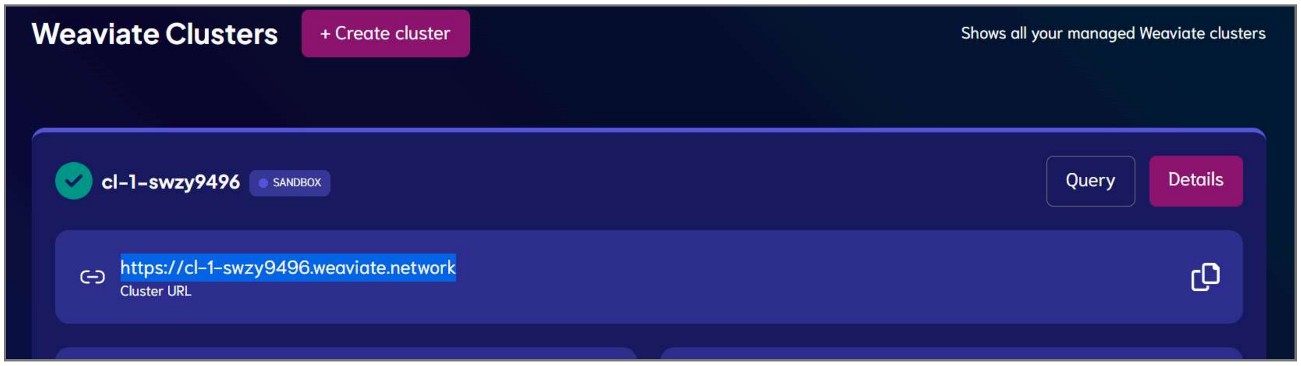
FREE SANDBOX

- Hosted on Google Cloud Services
- Monitoring
- Community support
- 14 days lifetime
- Public Slack
- Single Availability Zone

Cluster name: cl-1

Please note that a suffix will be added to the name upon creation.

Enable Authentication? NO



Shows all your managed Weaviate clusters

cl-1-swzy9496

https://cl-1-swzy9496.weaviate.network

Cluster URL

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. (3 points)

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

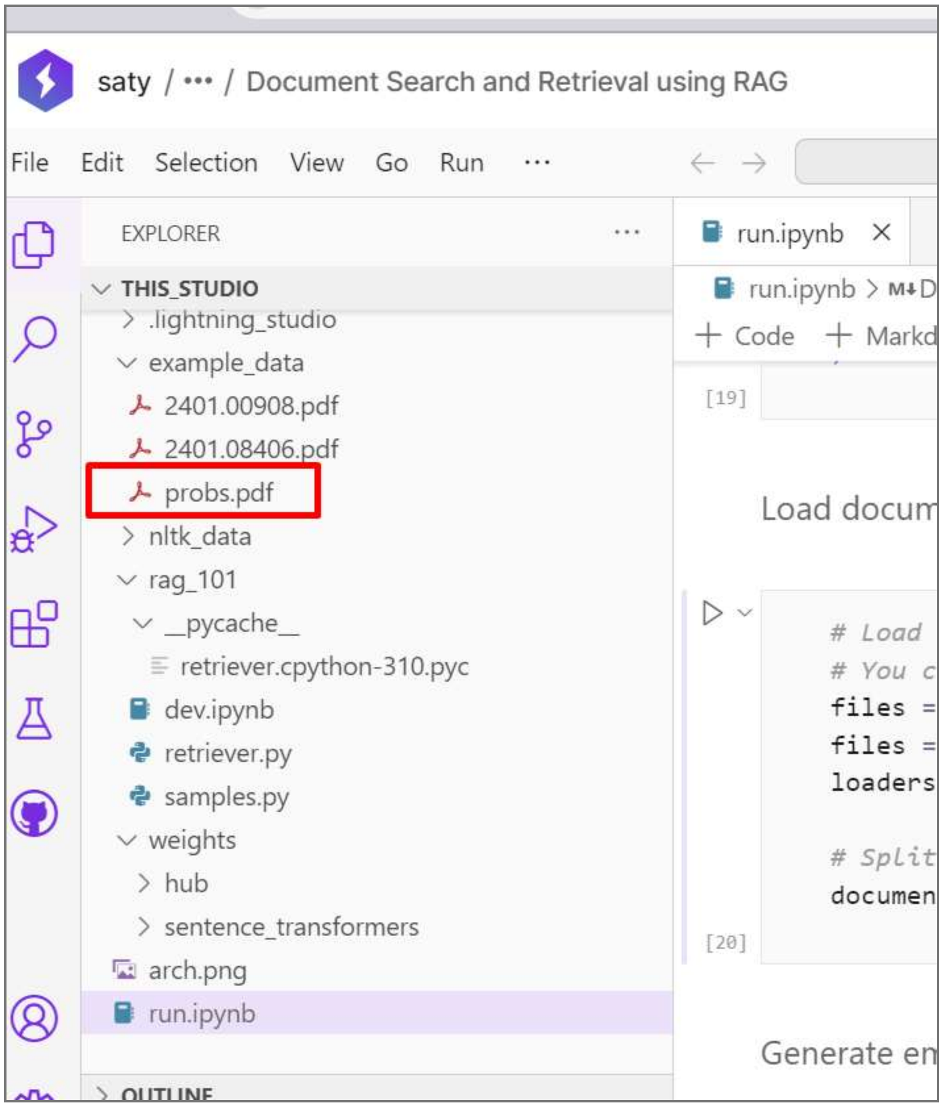
Browse through their vast collection of 'Studio' templates: https://lightning.ai/studios - when you create (instantiate) one, you get your own sandboxed environment [a 'cloud supercomputer'] that runs on lightning.ai's servers. You get unlimited

CPU use, and 22 hours of GPU use per month (PLENTY, for beginner projects).

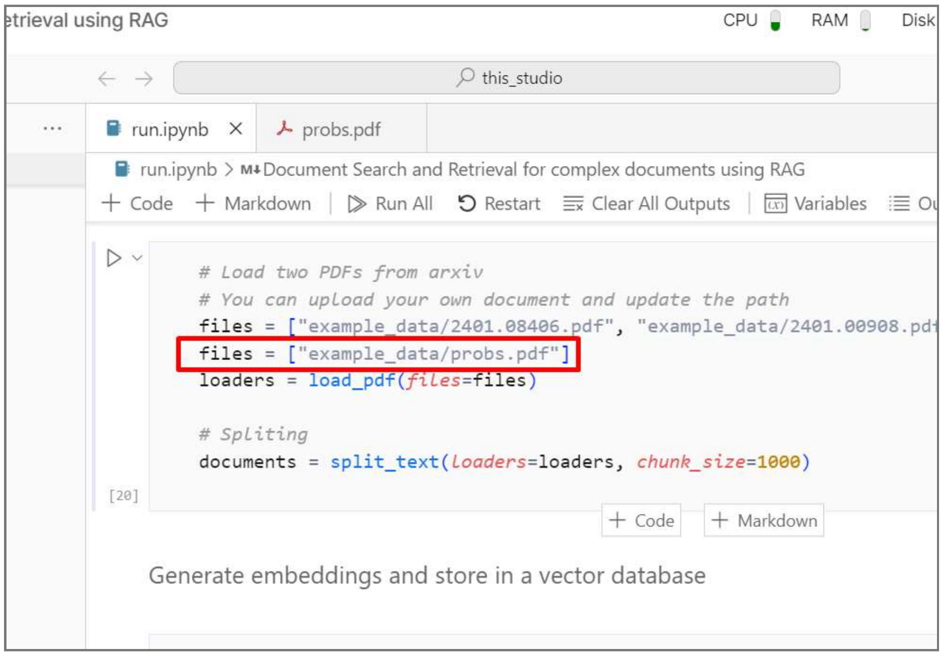
Bring up https://lightning.ai/studios?section=featured&query=document+search+and+retrieval+using+rag, then pick the first result (a studio by 'aniket') - click on it so the IDE comes up [CORRECTION: earlier I'd specified the actual link, which was https://lightning.ai/lightning-ai/studios/document-search-and-retrieval-using-rag]; you are going to use this to do RAG using your own PDF :)

Upload (drag and drop) a 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:

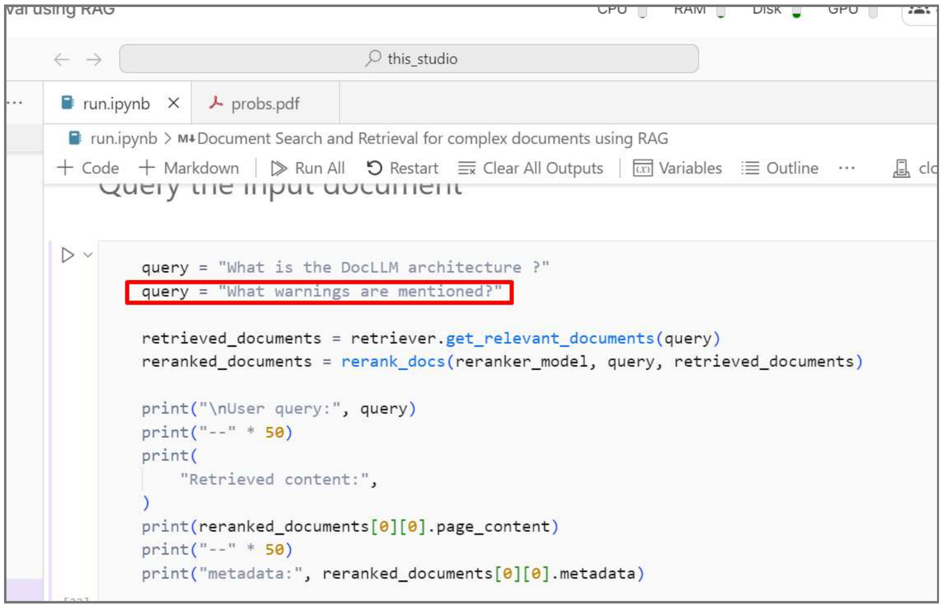

etrieval using RAG
CPU RAM Disk
this_studio
run.ipynb
probs.pdf
run.ipynb > M↓Document Search and Retrieval for complex documents using RAG
+ Code + Markdown Run All Restart Clear All Outputs Variables Ou
[20]
+ Code + Markdown

```
# Load two PDFs from arxiv
# You can upload your own document and update the path
files = ["example_data/2401.08406.pdf", "example_data/2401.00908.pdf
files = ["example_data/probs.pdf"]
loaders = load_pdf(files=files)

# Spliting
documents = split_text(loaders=loaders, chunk_size=1000)
```

Generate embeddings and store in a vector database

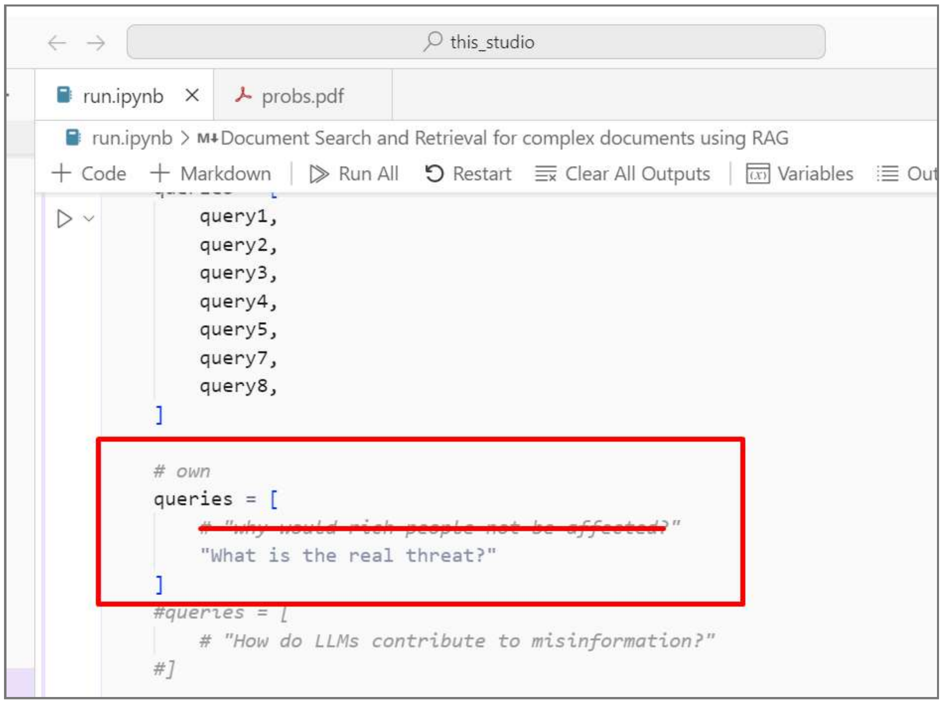
Modify the 'query' var, and the 'queries' var, to each contain a QUESTION on which you'd like to do RAG, ie. get the answers from the pdf you uploaded! THIS is the cool part - to be able to **ask questions in natural language**, rather than search by keyword, or look up words in the index [if the pdf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



```
query = "What is the DocLLM architecture ?"
query = "What warnings are mentioned?"

retrieved_documents = retriever.get_relevant_documents(query)
reranked_documents = rerank_docs(reranker_model, query, retrieved_documents)

print("\nUser query:", query)
print("--" * 50)
print(
    "Retrieved content:",
)
print(reranked_documents[0][0].page_content)
print("--" * 50)
print("metadata:", reranked_documents[0][0].metadata)
```



```
    query1,
    query2,
    query3,
    query4,
    query5,
    query7,
    query8,
]

# own
queries = [
    ~~# "Why would rich people not be affected?"~~
    "What is the real threat?"
]
#queries = [
    # "How do LLMs contribute to misinformation?"
#]
```

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 LOTS of 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 :)

Q3. (3 points)

This is also an easy+fun one...

Go to https://huggingface.co/spaces, scroll through the TONS of apps there, and pick a 'RAG' one [https://huggingface.co/spaces?sort=trending&search=RAG]. Run it, by uploading what's required (usually a PDF file).

Get a screenshot of your interacting with the app [similar to Q2].

Additionally, go through the source for the app [look in the 'Files' tab on the top right (sometimes it's in the ... icon)] and:

· describe the RAG steps involved (ie. the Python calls) - you don't need to do this in exhaustive detail, but you DO need to describe the workflow (that starts with ingesting the input, ending with output generation)

· describe how the UI works (what calls do whatwhat) - the file component, input, output. The UI is generated via Gradio [https://www.gradio.app/] or Streamlit [https://streamlit.io/], so this exercise is to get you familiar with their basics :)

Simply submit screenshots (for Q1, Q2, Q3) and a text file (for Q3), all as a single .zip file.

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! These (LLMs, vectorizing, RAG, Python-based UI spec...) are really (REALLY!!!) useful pieces of 'tech' to know. In the coming years, LLMs are sure to make their way into EVERY app/site/backend/system/process.