

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 'SLM :)]
- 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>
(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
- 26c5c85e47da Pull complete
- 9e79879be9c7 Pull complete
- 9ad47fcd2c0c Pull complete
- 9da6498f32c0 Pull complete
- 756350766a45 Pull complete
- a64e61a28d1e Pull complete
- 39a8c791c8b5 Pull complete
- bfccbc963b3f Pull complete
- d808540300c6 Pull complete
- 188a0f8b4cb2 Pull complete
- 8ad63110c7d9 Pull complete
- 66c95f4520ed Pull complete
- 1df5bed45a5d Pull complete
- weaviate Pulled
- f56be85fc22e Pull complete
- fc5ceff4c76f Pull complete
- c9d28bc41971 Pull complete
- 10cc92ce0a30 Pull complete
[*] Running 0/1
- Network satyc_default Created
- Container satyc-weaviate-1 Started
- Container satyc-t2v-transformers-1 Started













```

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](#)

☐ Only show running containers

Search

<input type="checkbox"/>	Name	Image	Status	Port(s)	Last started	Actions
<input type="checkbox"/>	 satyc	-	Running (2/2)		3 minutes ago	  
<input type="checkbox"/>	 weaviate-1 4996ceb41ebe	semitechnologies/weaviate:1	Running	8080:8080	3 minutes ago	  
<input type="checkbox"/>	 t2v-transformers-1 6276609054ef	semitechnologies/transformers	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 automatically 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:

```
Anaconda Prompt
(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 match 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 Proboscidea', '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', 'category': '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>
(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>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"}, {"answer":"the atmosphere","category":"SCIENCE","question":"Changes in the tropospheric layer of this are what gives us weather"}, {"answer":"wire","category":"SCIENCE","question":"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'

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

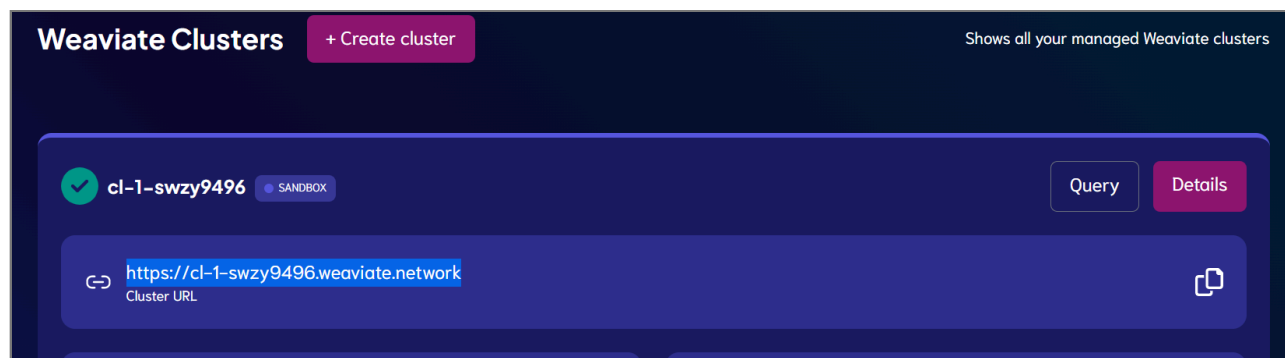
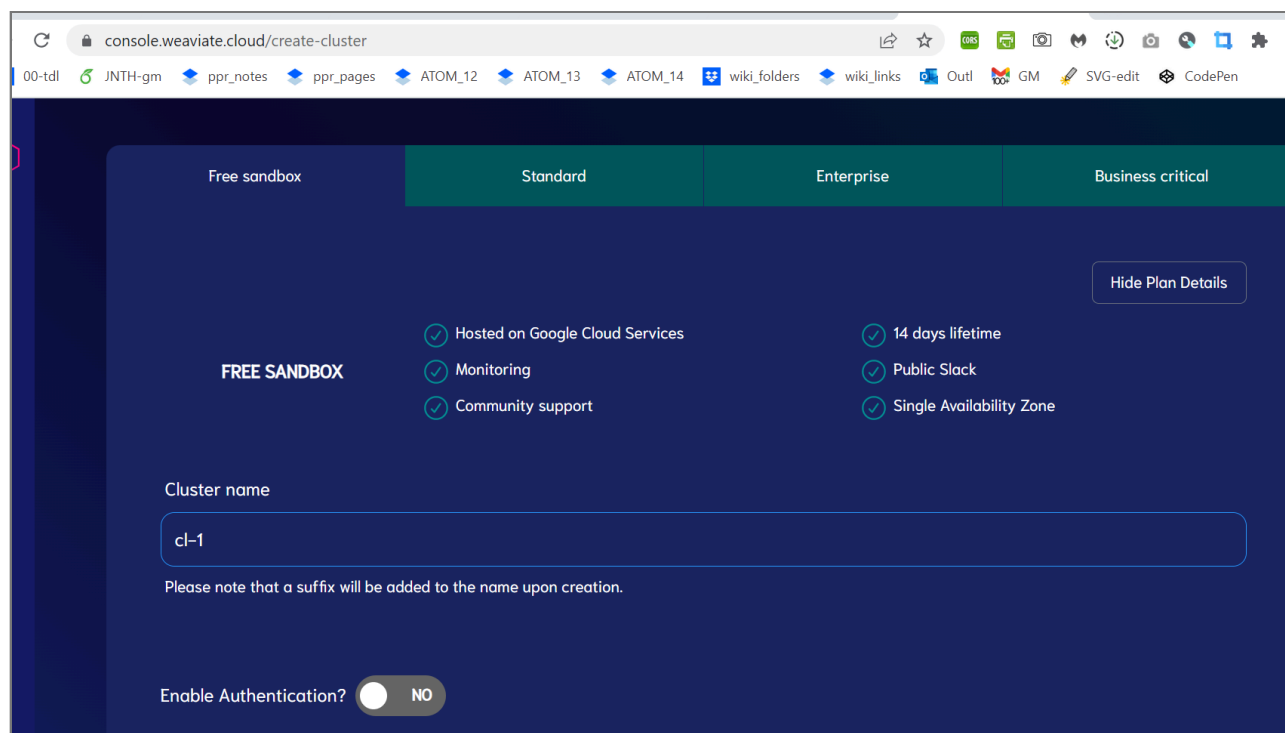
```
(base) C:\Users\satyc>curl localhost:8080/v1/meta
{"hostname":"http://[::]:8080","modules":{"text2vec-transformers":{"model":{"_name_or_path":"./models/model","add_cross_attention":false,"architectures":["BertModel"],"attention_probs_dropout_prob":0.1,"bad_words_ids":null,"begin_suppress_tokens":null,"bos_token_id":null,"chunk_size_feed_forward":0,"classifier_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,"exponential_decay_length_penalty":null,"finetuning_task":null,"forced_bos_token_id":null,"forced_eos_token_id":null,"gradient_checkpointing":false,"hidden_act":"gelu","hidden_dropout_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":false,"output_scores":false,"pad_token_id":0,"position_embedding_type":"absolute","prefix":null,"problem_type":null,"pruned_heads":{},"remove_invalid_values":false,"repetition_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":false,"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 generated by Weaviate's auto-schema feature on Thu Apr 27 20:25:22 2023","moduleConfig":{"text2vec-transformers":{"skip":false,"vectorizePropertyName":false}},"name":"answer","tokenization":"word"},"dataType":["text"],"description":"This property was generated by Weaviate's auto-schema feature on Thu Apr 27 20:25:22 2023","moduleConfig":{"text2vec-transformers":{"skip":false,"vectorizePropertyName":false}},"name":"question","tokenization":"word"},"dataType":["text"],"description":"This property was generated by Weaviate's auto-schema feature on Thu Apr 27 20:25:22 2023","moduleConfig":{"text2vec-transformers":{"skip":false,"vectorizePropertyName":false}},"name":"category","tokenization":"word"},"replicationConfig":{"factor":1},"shardingConfig":{"virtualPerPhysical":128,"desiredCount":1,"actualCount":1,"desiredVirtualCount":128,"actualVirtualCount":128,"key":"_id","strategy":"hash","function":"murmur3"},"vectorIndexConfig":{"skip":false,"cleanupIntervalSeconds":300,"maxConnections":64,"efConstruction":128,"efSearch":128,"dynamicEffMin":100,"dynamicEffMax":500,"dynamicEffFactor":8,"vectorCacheMaxObjects":1000000000000,"flatSearchCutoff":40000,"distance":"cosine","pq":{"enabled":false,"bitCompression":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:



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:

```
Anaconda Prompt (miniconda3)
(base) C:\Users\satyc>
(base) C:\Users\satyc>git clone https://github.com/builderio/gpt-crawler
Cloning into 'gpt-crawler'...
remote: Enumerating objects: 248, done.
remote: Counting objects: 100% (127/127), done.
remote: Compressing objects: 100% (42/42), done.
Receiving objects: 93% (231/248) 91 (delta 85), pack-reused 121
Receiving objects: 100% (248/248), 417.32 KiB | 1.37 MiB/s, done.
Resolving deltas: 100% (131/131), done.

(base) C:\Users\satyc>cd gpt-crawler

(base) C:\Users\satyc\gpt-crawler>ls
Dockerfile LICENSE README.md config.ts containerapp package-lock.json package.json src tsconfig.json

(base) C:\Users\satyc\gpt-crawler>npm install

> @builder-io/gpt-crawler@0.0.1 preinstall
> npx playwright install

Downloading Firefox 119.0 (playwright build v1429) from https://playwright.azureedge.net/builds/firefox/1429/firefox-win64.zip
80.5 Mb [=====] 100% 0.8s
Firefox 119.0 (playwright build v1429) downloaded to C:\Users\satyc\AppData\Local\ms-playwright\firefox-1429
Downloading Webkit 17.4 (playwright build v1944) from https://playwright.azureedge.net/builds/webkit/1944/webkit-win64.zip
46.4 Mb [=====] 100% 0.8s
Webkit 17.4 (playwright build v1944) downloaded to C:\Users\satyc\AppData\Local\ms-playwright\webkit-1944

added 340 packages, and audited 341 packages in 52s

81 packages are looking for funding
  run `npm fund` for details

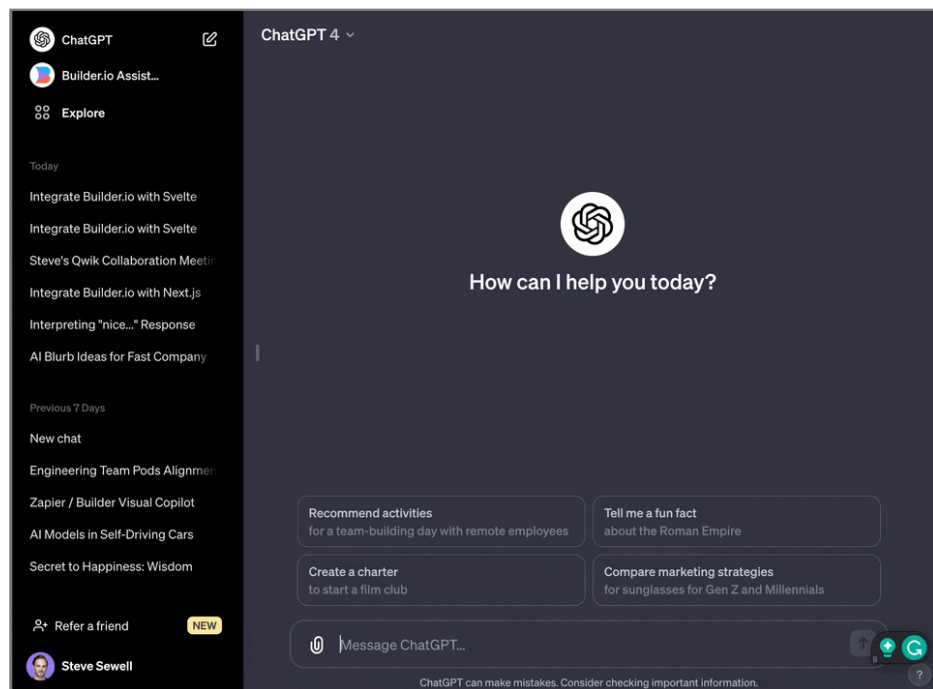
found 0 vulnerabilities

(base) C:\Users\satyc\gpt-crawler>
```

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 custom GPT :)

From builder.io's GitHub page:

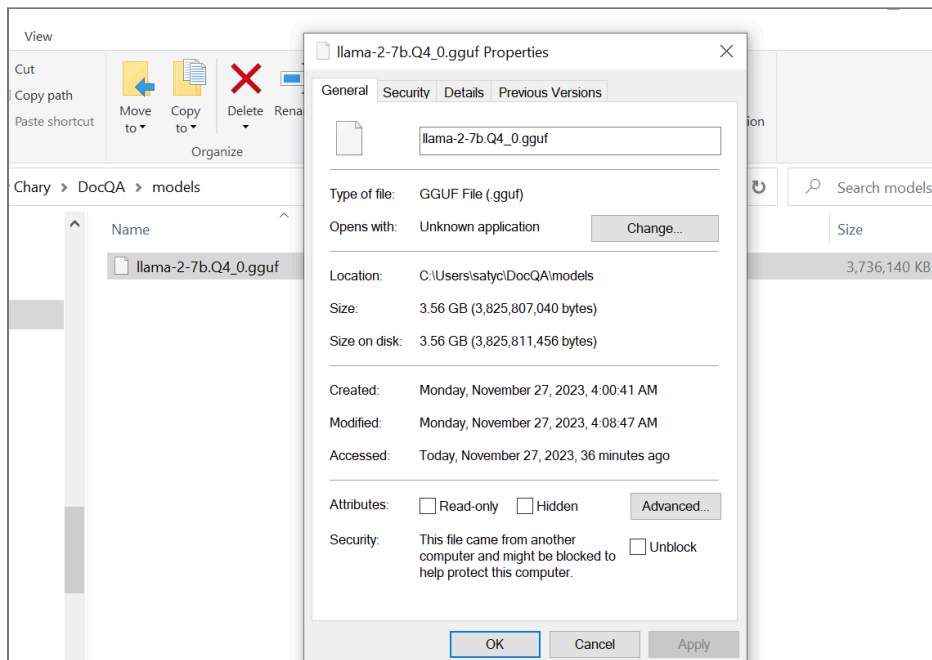


Amazing! You can use this to create all sorts of SMEs [subject matter experts] in the future, by simply scraping **existing** docs on the web.

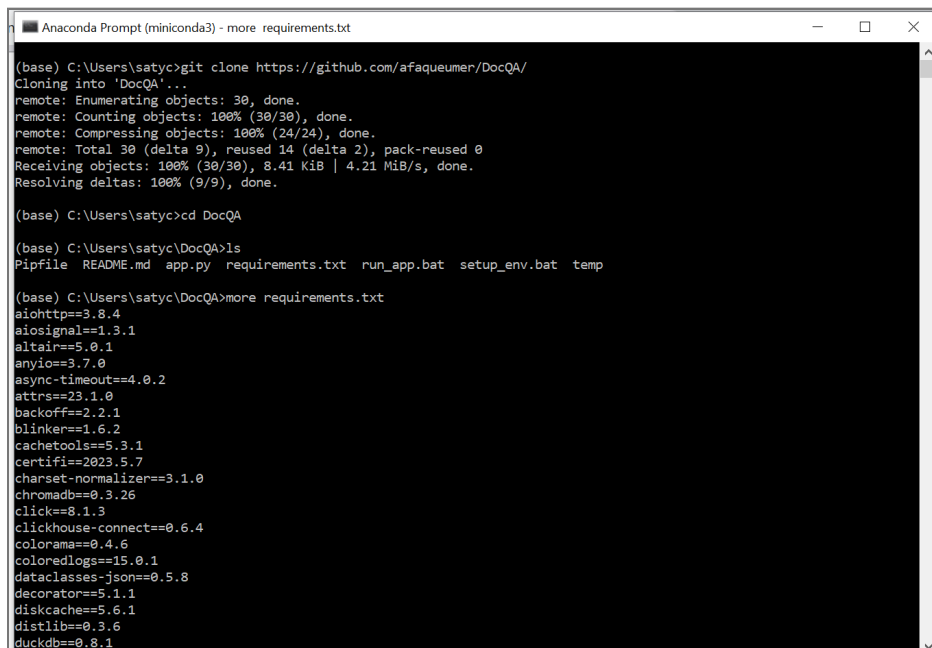
Q3.

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:


```
Anaconda Prompt (miniconda3)
```

```
(base) C:\Users\satyc\DocQA>
(base) C:\Users\satyc\DocQA>
(base) C:\Users\satyc\DocQA>
(base) C:\Users\satyc\DocQA>
(base) C:\Users\satyc\DocQA>
(base) C:\Users\satyc\DocQA>pipenv install
'pipenv' is not recognized as an internal or external command,
operable program or batch file.

(base) C:\Users\satyc\DocQA>pip install --user pipenv
Collecting pipenv
  Downloading pipenv-2023.11.15-py3-none-any.whl (3.2 MB)
    ----- 3.2/3.2 MB 1.9 MB/s eta 0:00:00
Requirement already satisfied: certifi in c:\users\satyc\miniconda3\lib\site-packages (from pipenv) (2023.5.7)
Requirement already satisfied: setuptools>=67 in c:\users\satyc\miniconda3\lib\site-packages (from pipenv) (67.8.0)
Collecting virtualenv>=20.24.2 (from pipenv)
  Downloading virtualenv-20.24.7-py3-none-any.whl (3.8 MB)
    ----- 3.8/3.8 MB 4.0 MB/s eta 0:00:00
Collecting distlib1,>=0.3.7 (from virtualenv>=20.24.2->pipenv)
  Downloading distlib-0.3.7-py2.py3-none-any.whl (468 kB)
    ----- 468.9/468.9 kB 4.2 MB/s eta 0:00:00
Collecting filelock4,>=3.12.2 (from virtualenv>=20.24.2->pipenv)
  Downloading filelock-3.13.1-py3-none-any.whl (11 kB)
Collecting platformdirs<5,>=3.9.1 (from virtualenv>=20.24.2->pipenv)
  Downloading platformdirs-4.0.0-py3-none-any.whl (17 kB)
Installing collected packages: distlib, platformdirs, filelock, virtualenv, pipenv
WARNING: The script virtualenv.exe is installed in 'C:\Users\satyc\AppData\Roaming\Python\Python310\Scripts' which is
not on PATH.
  Consider adding this directory to PATH or, if you prefer to suppress this warning, use --no-warn-script-location.
WARNING: The scripts pipenv-resolver.exe and pipenv.exe are installed in 'C:\Users\satyc\AppData\Roaming\Python\Python
310\Scripts' which is not on PATH.
  Consider adding this directory to PATH or, if you prefer to suppress this warning, use --no-warn-script-location.
Successfully installed distlib-0.3.7 filelock-3.13.1 pipenv-2023.11.15 platformdirs-4.0.0 virtualenv-20.24.7

[notice] A new release of pip is available: 23.1.2 -> 23.3.1
[notice] To update, run: python.exe -m pip install --upgrade pip

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

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

```
(base) C:\Users\satyc\DocQA> pipenv --python ../miniconda3/python.exe install
Creating a virtualenv for this project...
Pipfile: C:\Users\satyc\DocQA\Pipfile
Using ../miniconda3/python.exe (3.10.10) to create virtualenv...
created virtual environment CPython3.10.10.Final.0-64 in 4049ms
creator CPython3\Windows(dest=C:\Users\satyc\.virtualenvs\DocQA-B8gy-j-T, clear=False, no_vcs_ignore=False, global=False)
seeder FromAppData(download=False, pip=bundle, setuptools=bundle, wheel=bundle, via=copy, app_data_dir=C:\Users\satyc\AppData\Local\pypa\virtualenv)
added seed packages: pip==23.3.1, setuptools==68.2.2, wheel==0.41.3
activators BashActivator,BatchActivator,FishActivator,PowerShellActivator,PythonActivator
[== ]
Creating virtual environment...Successfully created virtual environment!
Virtualenv location: C:\Users\satyc\.virtualenvs\DocQA-B8gy-j-T
Warning: Your Pipfile requires python_version 3.9, but you are using 3.10.10 (C:\Users\satyc\.\.\.python.exe).
$ pipenv --rm and rebuilding the virtual environment may resolve the issue.
$ pipenv check will surely fail.
Pipfile.lock not found, creating...
Locking [packages] dependencies...
Building requirements...
Resolving dependencies...
[== ] Locking...
```

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

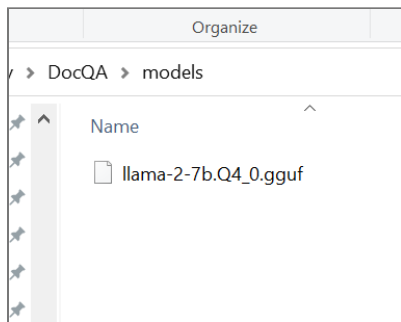
```

(base) C:\Users\satyc\DocQ>pip install llama-cpp-python==0.1.65 --force-reinstall --upgrade --no-cache-dir
Collecting llama-cpp-python==0.1.65
  Downloading llama_cpp_python-0.1.65.tar.gz (1.5 MB)
----- 1.5/1.5 MB 3.2 MB/s eta 0:00:00
Installing build dependencies ... done
Getting requirements to build wheel ... done
Preparing metadata (pyproject.toml) ... done
Collecting typing-extensions>=4.5.0 (from llama-cpp-python==0.1.65)
  Downloading typing_extensions-4.8.0-py3-none-any.whl.metadata (3.0 kB)
Collecting numpy>=1.20.0 (from llama-cpp-python==0.1.65)
  Downloading numpy-1.26.2-cp310-cp310-win_amd64.whl.metadata (61 kB)
----- 61.2/61.2 kB ? eta 0:00:00
Collecting diskcache>=5.6.1 (from llama-cpp-python==0.1.65)
  Downloading diskcache-5.6.3-py3-none-any.whl.metadata (20 kB)
Download diskcache-5.6.3-py3-none-any.whl (45 kB)
----- 45.5/45.5 kB ? eta 0:00:00
Download numpy-1.26.2-cp310-cp310-win_amd64.whl (15.8 MB)
----- 15.8/15.8 MB 8.8 MB/s eta 0:00:00
Download typing_extensions-4.8.0-py3-none-any.whl (31 kB)
Building wheels for collected packages: llama-cpp-python
  Building wheel for llama-cpp-python (pyproject.toml) ... done
  Created wheel for llama-cpp-python: filename=llama_cpp_python-0.1.65-cp310-cp310-win_amd64.whl size=474543 sha256=ec9f0e00d43581259db0a5e2b02e5d2def2965658d46935fec474ae758f360d
  Stored in directory: C:\Users\satyc\AppData\Local\Temp\pip-ephem-wheel-cache-1mcvt6t\wheels\e0\62\84\21f820209ad725e813c2dd41eeda1a0dcb9184af7022d55c0d
Successfully built llama-cpp-python
Installing collected packages: typing-extensions, numpy, diskcache, llama-cpp-python
  Attempting uninstall: typing-extensions
    Found existing installation: typing_extensions 4.7.1
    Uninstalling typing_extensions-4.7.1:
      Successfully uninstalled typing_extensions-4.7.1
  Attempting uninstall: numpy
    Found existing installation: numpy 1.25.1
    Uninstalling numpy-1.25.1:
      Successfully uninstalled numpy-1.25.1
  Attempting uninstall: llama-cpp-python
    Found existing installation: llama-cpp-python 0.1.53
    Uninstalling llama-cpp-python-0.1.53:
      Successfully uninstalled llama-cpp-python-0.1.53
ERROR: pip's dependency resolver does not currently take into account all the packages that are installed. This behavior
is the source of the following dependency conflicts.
gensim 4.3.0 requires FuzzyTMO==0.4.0, which is not installed.
tables 3.8.0 requires blosc2==2.0.0, which is not installed.
tables 3.8.0 requires cython>=0.29.21, which is not installed.
langchain 0.0.340 requires jsonpatch<2.0,>=1.33, but you have jsonpatch 1.32 which is incompatible.
numba 0.58.0 requires numpy<1.26,>=1.21, but you have numpy 1.26.2 which is incompatible.
Successfully installed diskcache-5.6.3 llama-cpp-python-0.1.65 numpy-1.25.2 typing-extensions-4.8.0

```

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:

```
File Edit Search Tools Favourites Clips Options Help
16
17 ##specify model path's full pathname
18 mp = "C:/Users/satyc/DocQA/models/llama-2-7b.Q4_0.gguf"
19
20
21 # customize the layout
22 st.set_page_config(page_title="DOCAI", page_icon="🤖", layout="wide", )
23 st.markdown(f"""
24 <style>
25 .stApp {{background-image: url("https://images.unsplash.com/photo-1509537257950-20f875
26 4.0.3&ixid=M3wxMjA3fDB8MHxwaG90byl1YWdlfHx8fGVuLWVufDB8fHx8fA%3D%3D&auto=format&fit=crop&w=1469&q=80")}
27 background-attachment: fixed;
28 background-size: cover}}
29 </style>
30 """, unsafe_allow_html=True)
31
32 # function for writing uploaded file in temp
33 def write_text_file(content, file_path):
34     try:
35         with open(file_path, 'w') as file:
36             file.write(content)
37         return True
38     except Exception as e:
39         print(f"Error occurred while writing the file: {e}")
40         return False
41
42 # set prompt template
43 prompt_template = """Use the following pieces of context to answer the question at the end. If you
44 that you don't know, don't try to make up an answer.
45 {context}
46 Question: {question}
47 Answer: ""
48 prompt = PromptTemplate(template=prompt_template, input_variables=["context", "question"])
49
50 # initialize the LLM & Embeddings
51 llm = LlamaCpp(model_path=mp)
52 embeddings = LlamaCppEmbeddings(model_path=mp)
53 llm_chain = LLMChain(llm=llm, prompt=prompt)
54
55 st.title("📄 Document Conversation 🤖")
56 uploaded_file = st.file_uploader("Upload an article", type="txt")
57
58 if uploaded_file is not None:
59     content = uploaded_file.read().decode('utf-8')
```

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]:

```
Anaconda Prompt (miniconda3) - pipenv run run_app.bat
(base) C:\Users\satyc>cd DocQA

(base) C:\Users\satyc\DocQA>pipenv install
Installing dependencies from Pipfile.lock (ba35ae)...
To activate this project's virtualenv, run pipenv shell.
Alternatively, run a command inside the virtualenv with pipenv run.

(base) C:\Users\satyc\DocQA>pipenv run
Usage: pipenv run [OPTIONS] COMMAND [ARGS]...
Try 'pipenv run -h' for help.

Error: Missing argument 'COMMAND'.

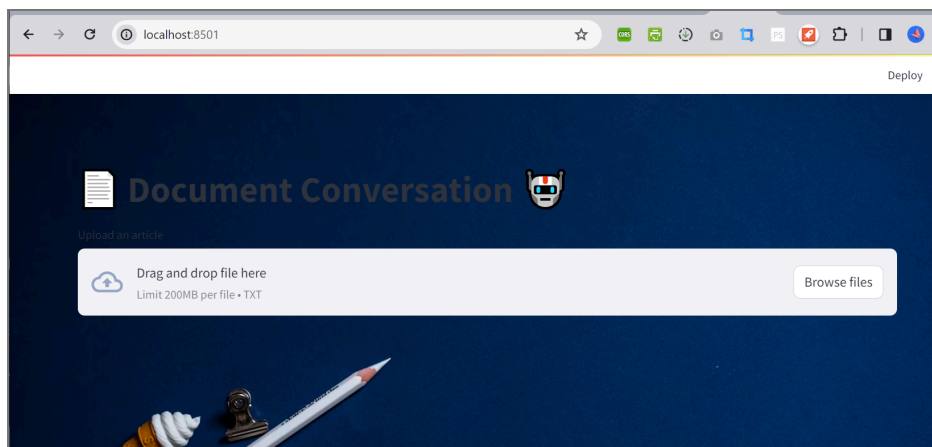
(base) C:\Users\satyc\DocQA>pipenv run run_app.bat

(base) C:\Users\satyc\DocQA>pipenv run streamlit run app.py

You can now view your Streamlit app in your browser.

Local URL: http://localhost:8501
Network URL: http://192.168.1.7:8501
```

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



```
Anaconda Prompt (miniconda3) - pipenv run streamlit run app.py
(base) C:\Users\satyc\DocQA>
(base) C:\Users\satyc\DocQA>pipenv run streamlit run app.py

You can now view your Streamlit app in your browser.

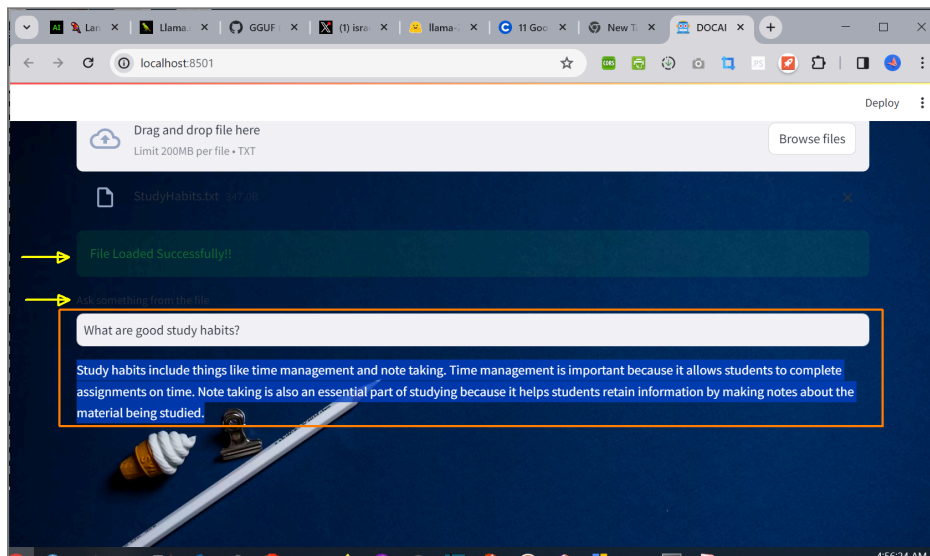
Local URL: http://localhost:8501
Network URL: http://192.168.1.7:8501

llama_model_loader: loaded meta data with 19 key-value pairs and 291 tensors from C:/Users/satyc/DocQA/models/llama-2-7b
.Q4_0.gguf (version GGUF V2)
llama_model_loader: - tensor 0: token_embd.weight q4_0 [ 4096, 32000, 1, 1 ]
llama_model_loader: - tensor 1: blk.0.attn_norm.weight f32 [ 4096, 1, 1, 1 ]
llama_model_loader: - tensor 2: blk.0.ffn_down.weight q4_0 [ 11008, 4096, 1, 1 ]
llama_model_loader: - tensor 3: blk.0.ffn_gate.weight q4_0 [ 4096, 11008, 1, 1 ]
llama_model_loader: - tensor 4: blk.0.ffn_up.weight q4_0 [ 4096, 11008, 1, 1 ]
llama_model_loader: - tensor 5: blk.0.ffn_norm.weight f32 [ 4096, 1, 1, 1 ]
llama_model_loader: - tensor 6: blk.0.attn_k.weight q4_0 [ 4096, 4096, 1, 1 ]
llama_model_loader: - tensor 7: blk.0.attn_output.weight q4_0 [ 4096, 4096, 1, 1 ]
llama_model_loader: - tensor 8: blk.0.attn_q.weight q4_0 [ 4096, 4096, 1, 1 ]
llama_model_loader: - tensor 9: blk.0.attn_v.weight q4_0 [ 4096, 4096, 1, 1 ]
llama_model_loader: - tensor 10: blk.1.attn_norm.weight f32 [ 4096, 1, 1, 1 ]
llama_model_loader: - tensor 11: blk.1.ffn_down.weight q4_0 [ 11008, 4096, 1, 1 ]
llama_model_loader: - tensor 12: blk.1.ffn_gate.weight q4_0 [ 4096, 11008, 1, 1 ]
llama_model_loader: - tensor 13: blk.1.ffn_up.weight q4_0 [ 4096, 11008, 1, 1 ]
llama_model_loader: - tensor 14: blk.1.ffn_norm.weight f32 [ 4096, 1, 1, 1 ]
llama_model_loader: - tensor 15: blk.1.attn_k.weight q4_0 [ 4096, 4096, 1, 1 ]
llama_model_loader: - tensor 16: blk.1.attn_output.weight q4_0 [ 4096, 4096, 1, 1 ]
```

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!

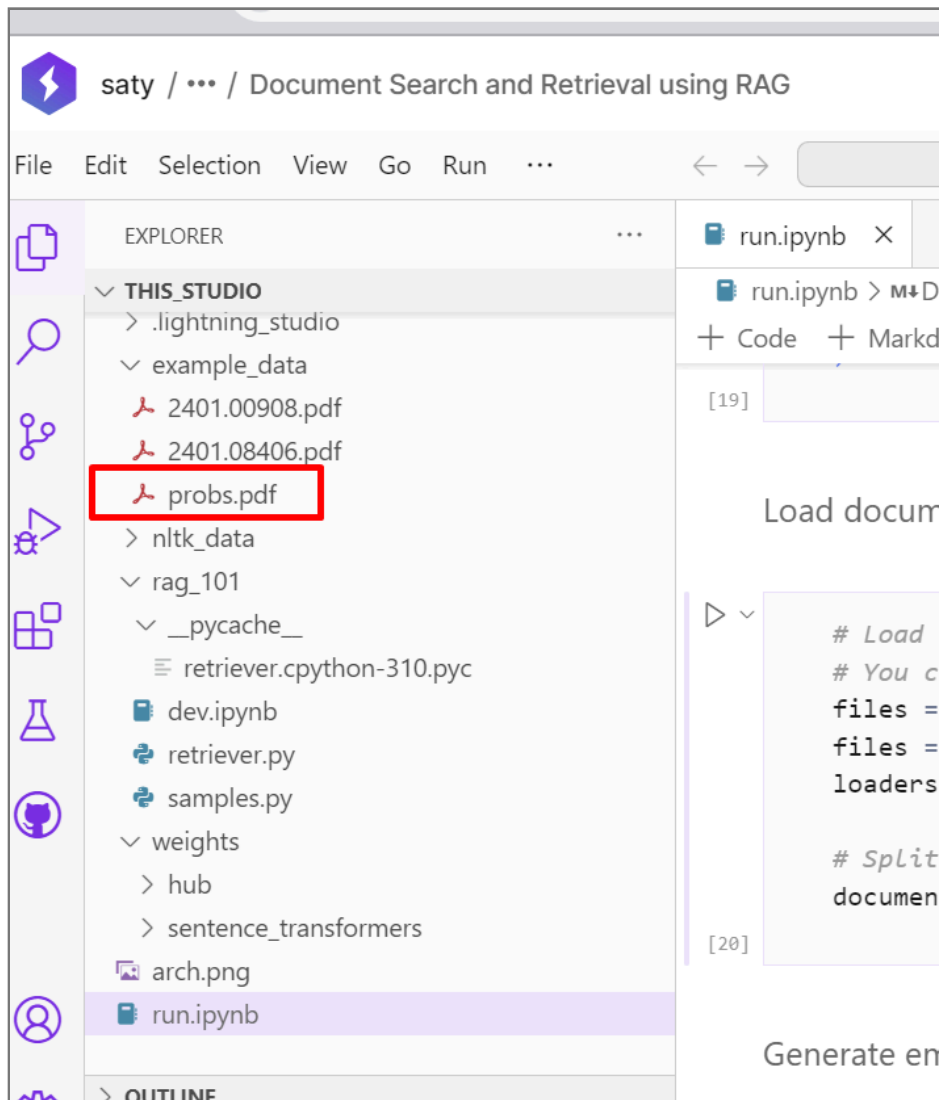
Go to <https://lightning.ai/> and sign up for a free account. Then read these: <https://lightning.ai/docs/overview/getting-started> and <https://lightning.ai/docs/overview/getting-started/studios-in-10-minutes>

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).

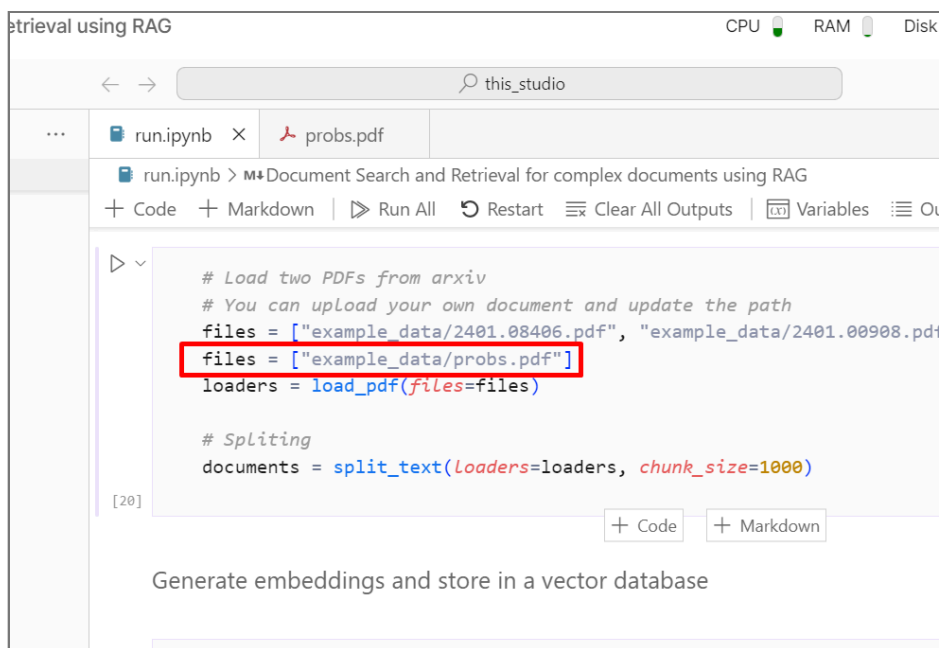
Create this Studio: <https://lightning.ai/lightning-ai/studios/document-search-and-retrieval-using-rag> - you are going to use this to do RAG using your own PDF :)

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

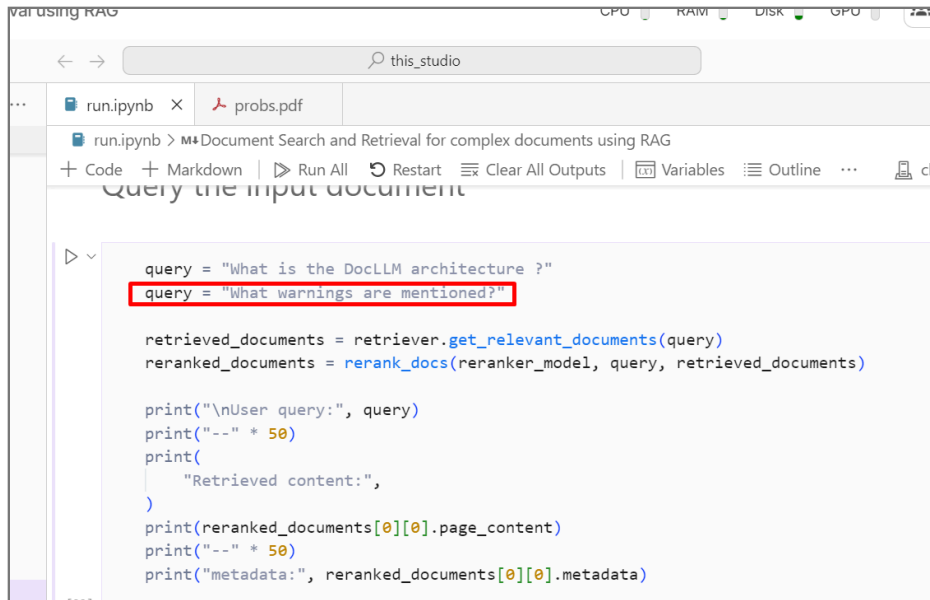
Eg. this shows the pdf I uploaded:



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



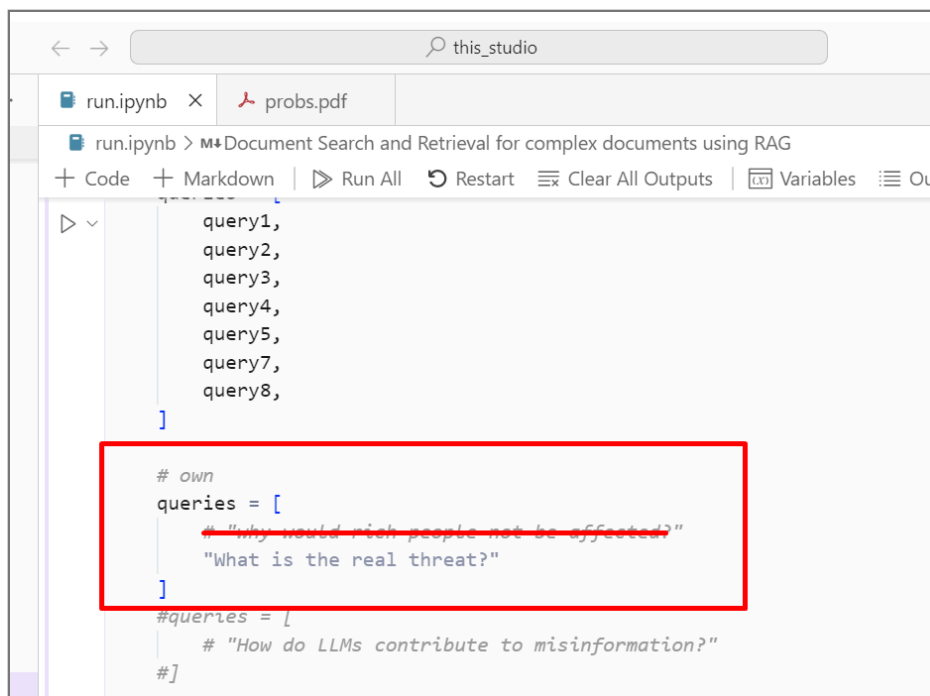
Modify the 'query' var, and the 'queries' var, to each contain a QUESTION on which you'd like to do RAG, ie. get the answers from the pdf you uploaded! THIS is the cool part - to be able to **ask questions in natural language**, rather than search by keyword, or look up words in the index [if the pdf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



```
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 (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/>
