LLM supporting tools that make the ecosystem

Paul Rodriguez
San Diego Supercomputer Center

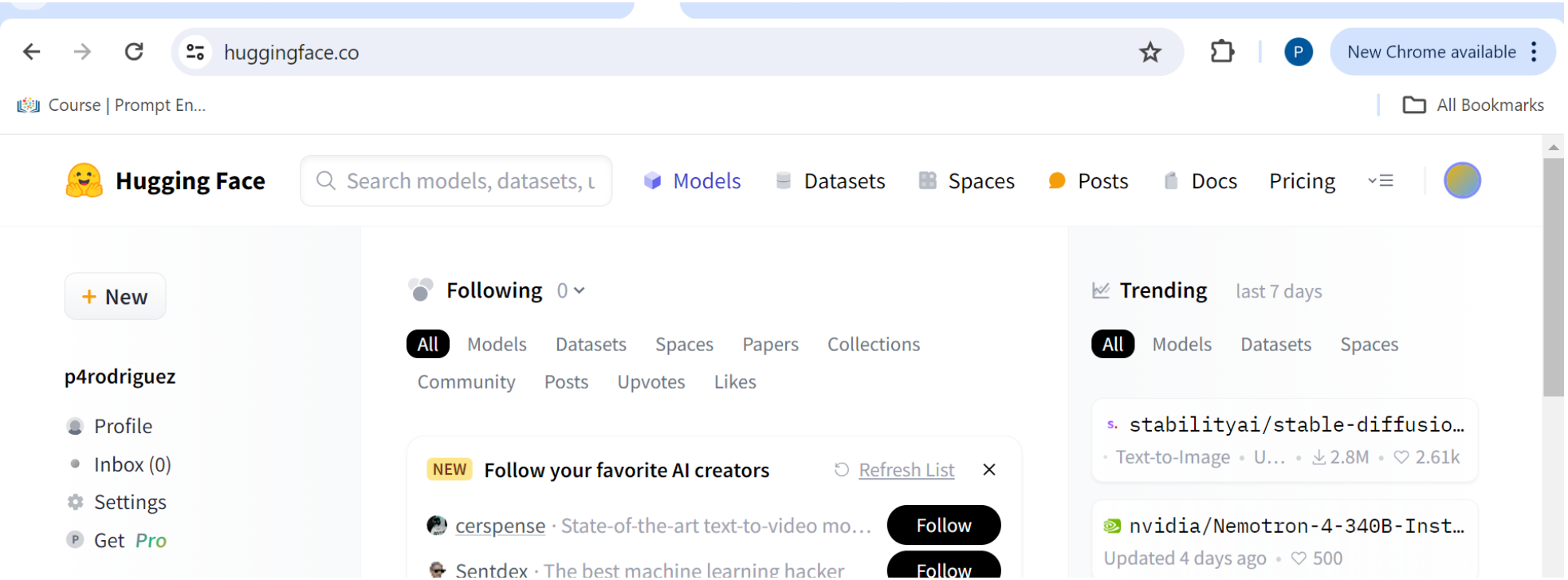
2024 CIML Summer Institute

Outline

- **The Hugging Face ecosystem**
- **Langchain**
- **Hugging Face and Langchain tutorial discussion/demo**

Hugging Face Hub

- **huggingface.com is a hub of models, data, tutorials for using AI models**



Using Hugging Face

- huggingface provides python packages to make it (relatively) easy to run models

- Using models, data require an authentification token

- Some of the main packages are:

pipeline:	to run inferencing
diffusers:	for diffusion models
transformers:	for LLMs
accelerate:	for efficient and/or parallel execution
datasets:	to access data from hub

Hugging Face Abstractions

- The pipeline function is built on more basic functions:

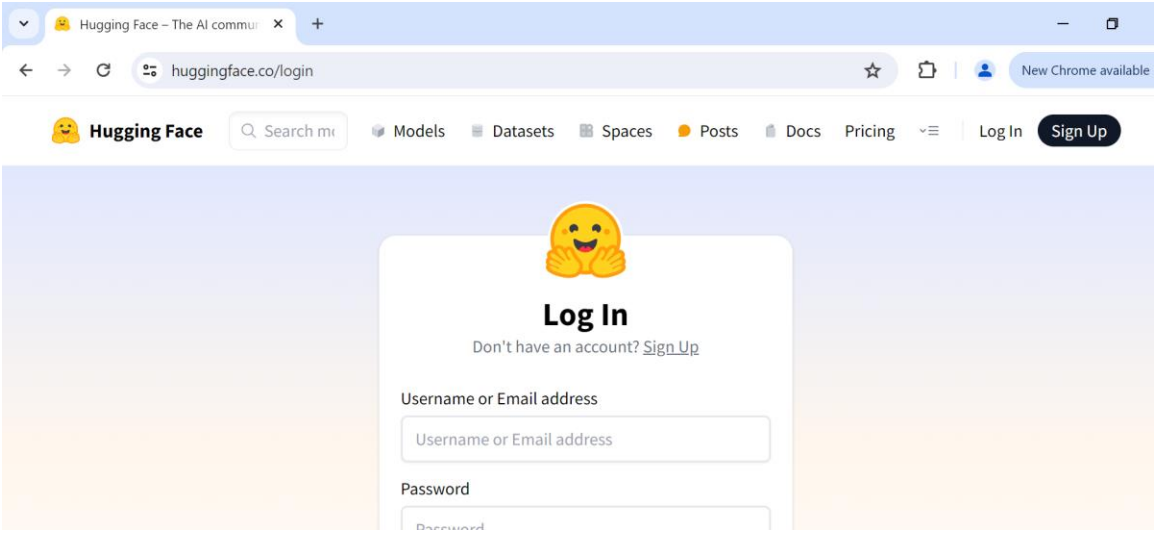
get “config” (to get the model architecture)
get “model” (to download a pretrained model)
get “tokenizer” (to get the appropriate tokenizer)

- These can be invoked more directly if you want to do a pre-training or fine-tuning implementation, or experimentation

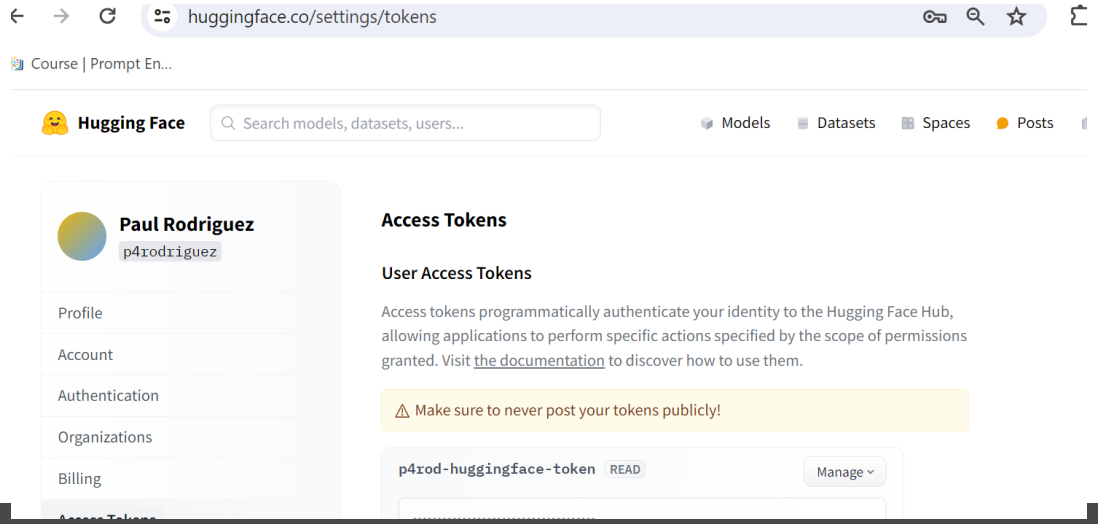
Getting account set up

- **Create an account on huggingface and get authentication token**

huggingface.co/login

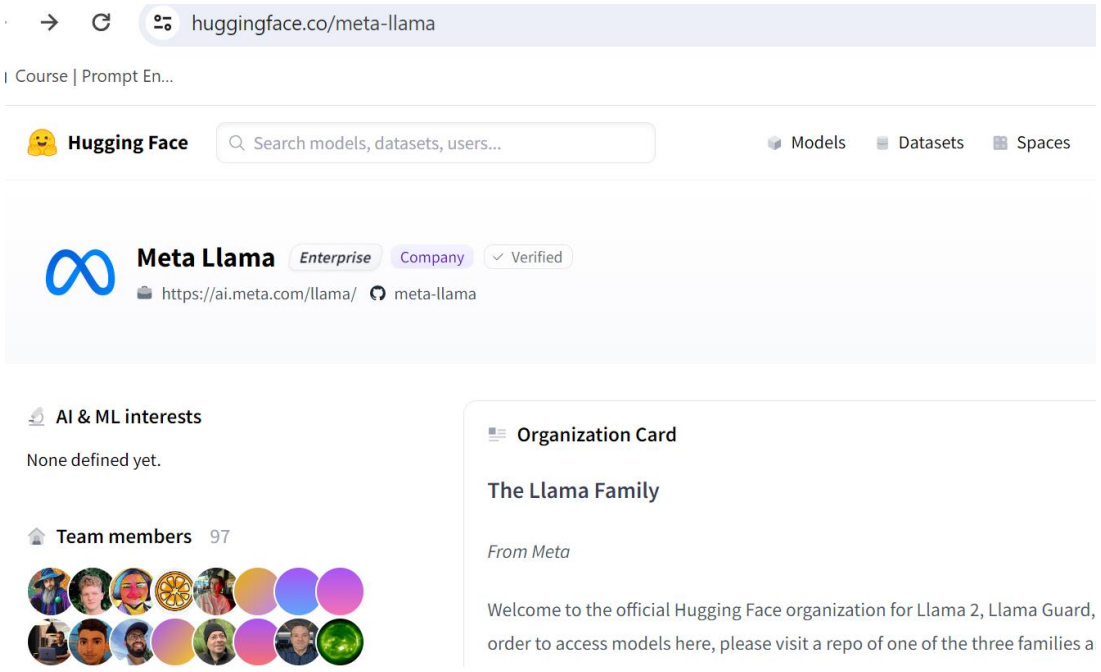


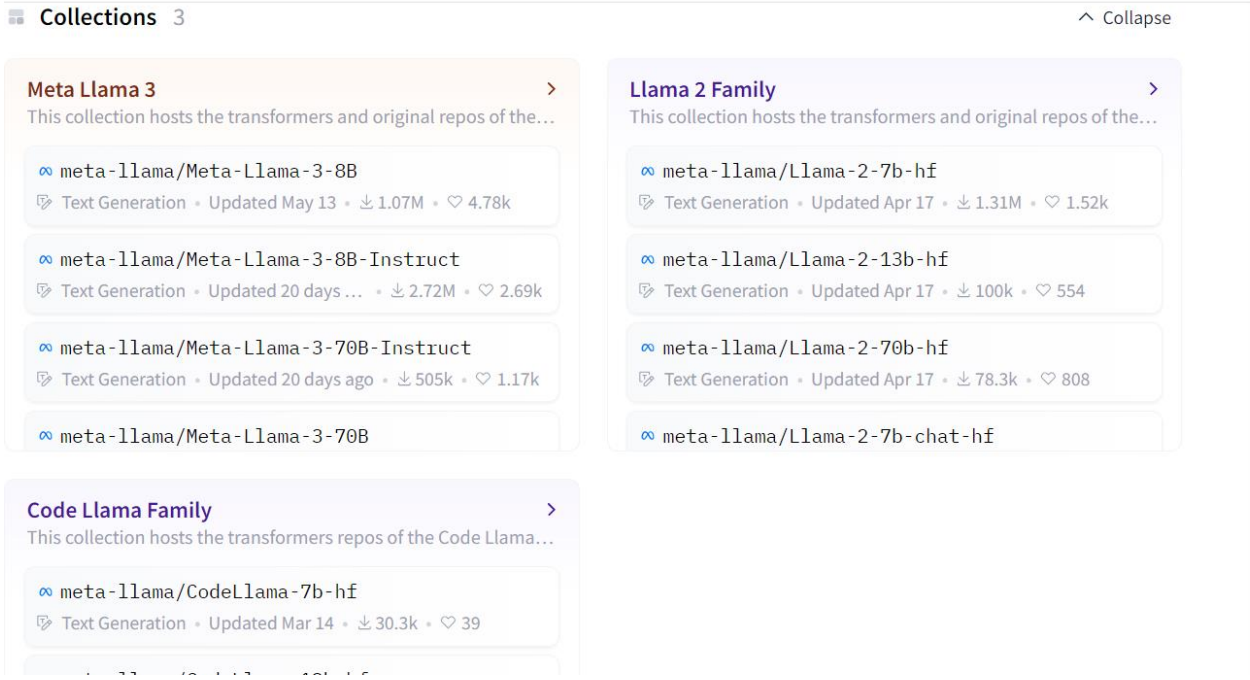
Aacount -> settings -> get token



Hugging Face model repo

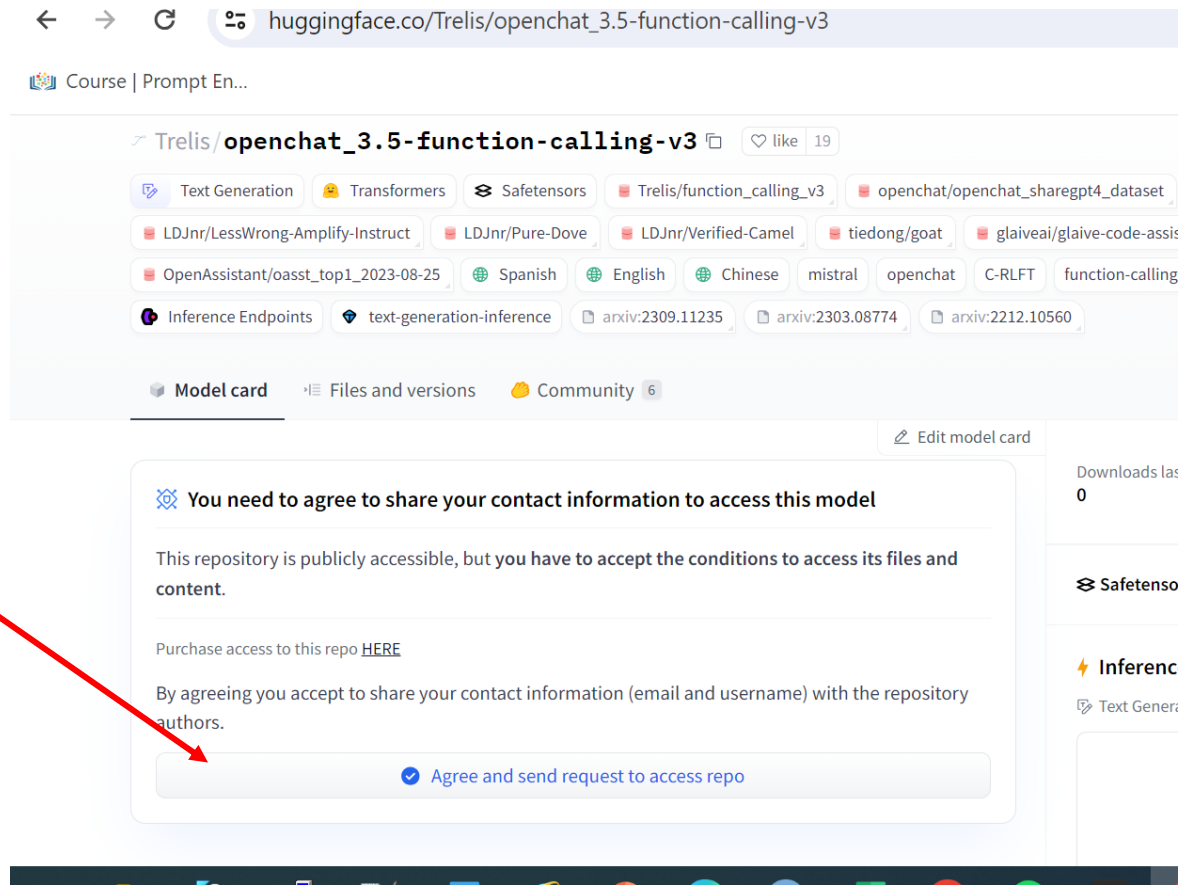
- **Example: Meta has a family of Llama models that vary by size, response training, ‘hf’ format, release date, etc..**





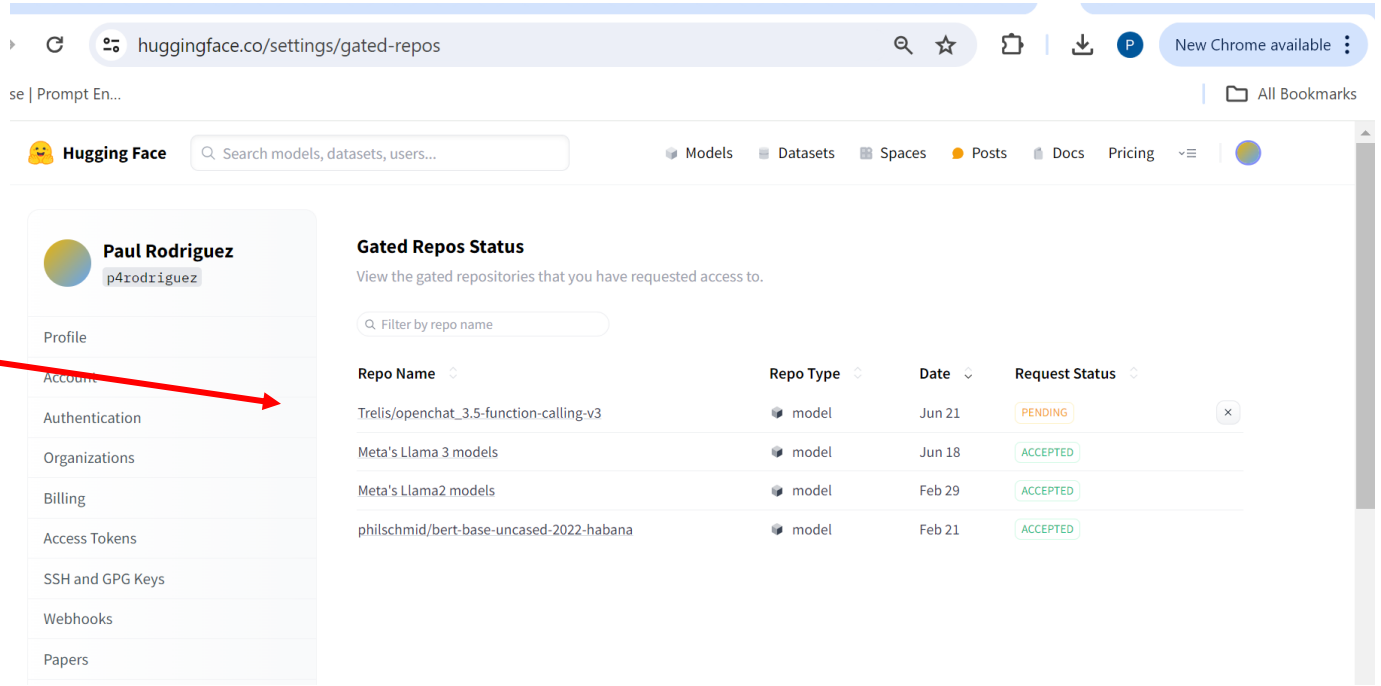
Hugging Face model repo

- **Some models or data require that you request access here**



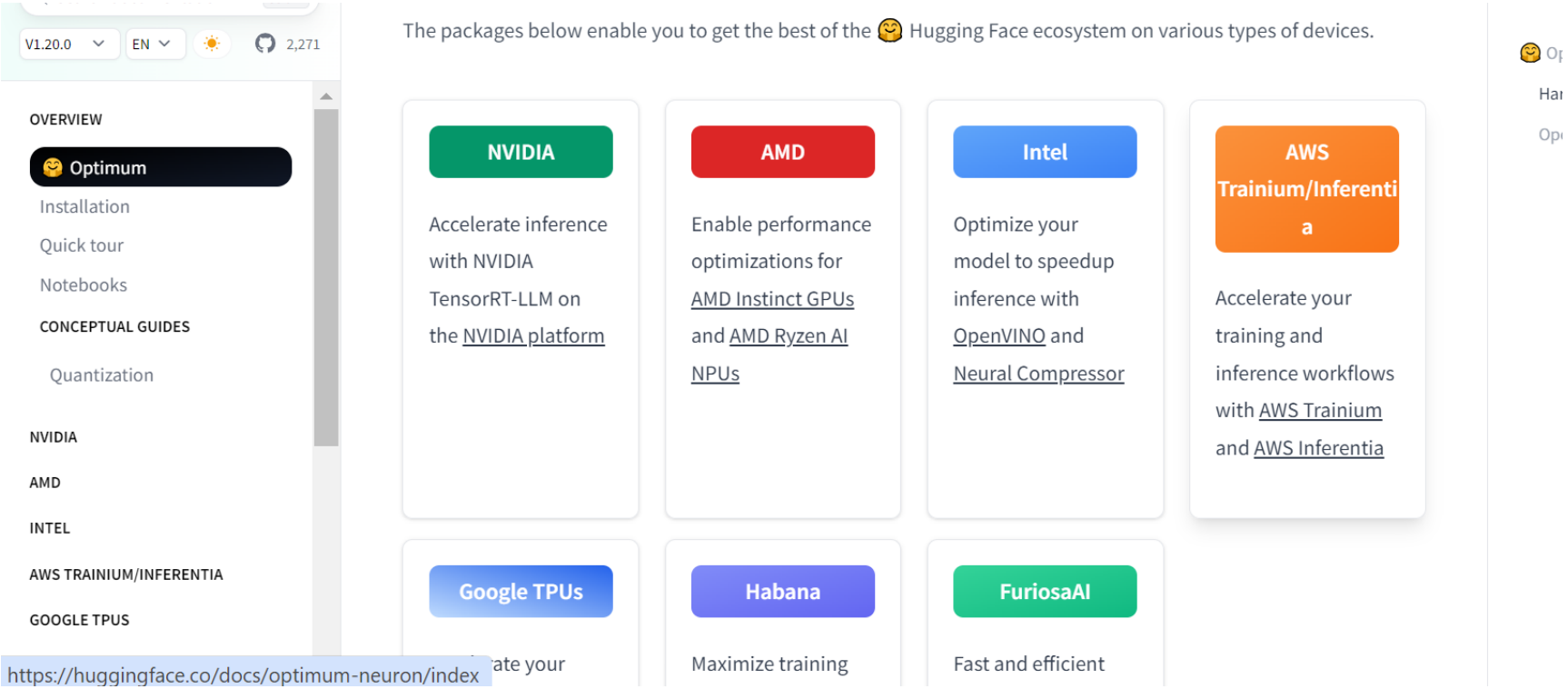
Hugging Face model repo

- Some models or data require that you request access
- Then go to
  account->
  setting->
  gated repo



Hardware Partners

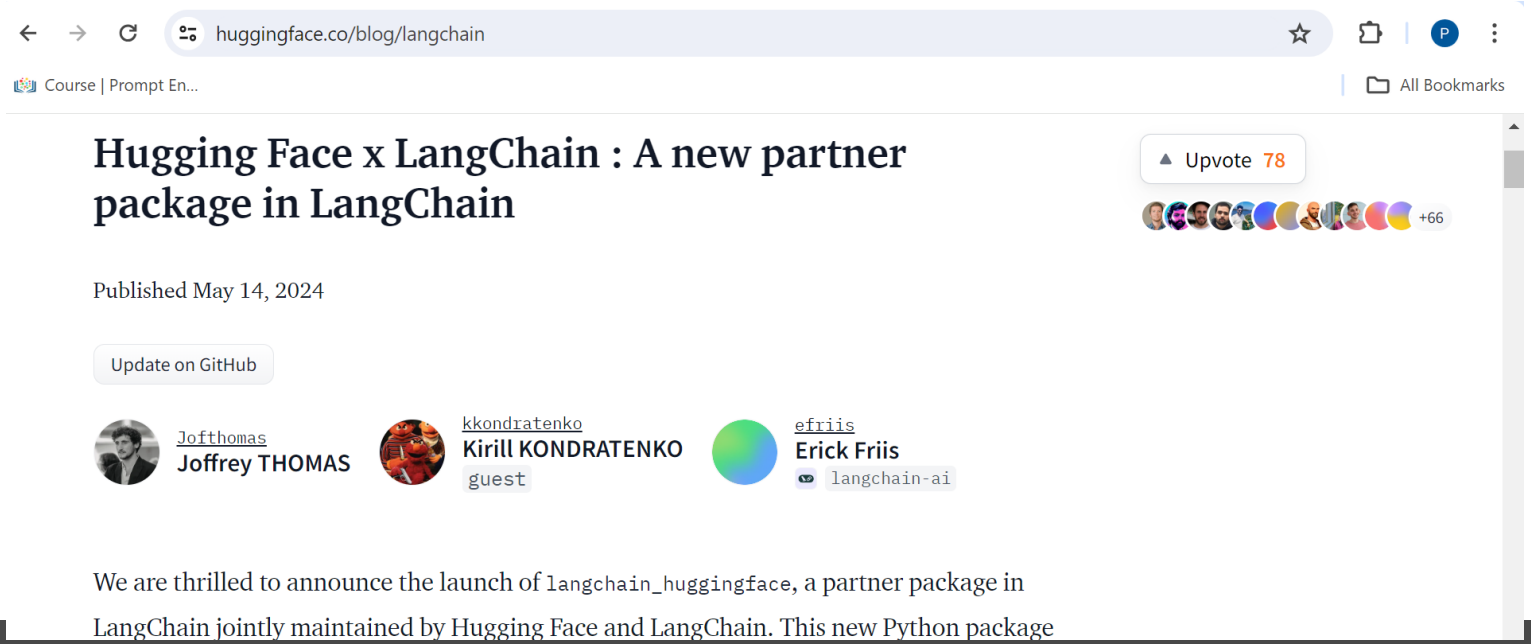
- **Hugging Face also incorporates accelerator libraries into their packages**



https://huggingface.co/docs/optimum-neuron/index

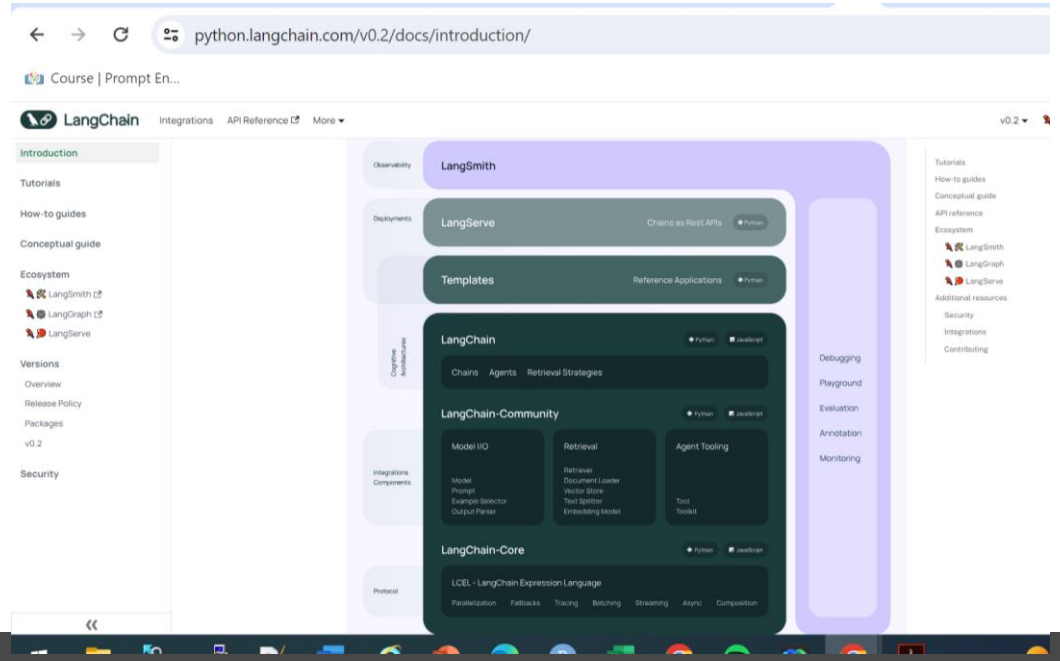
Partner packages

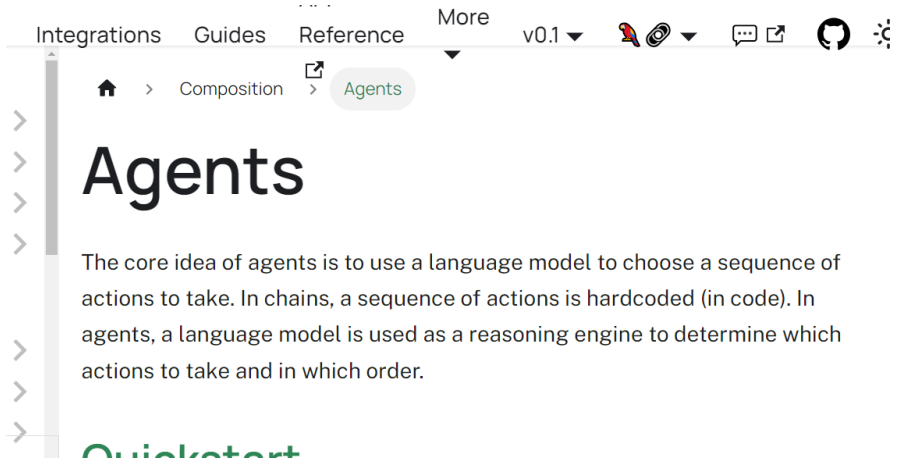
- **Hugging Face incorporates other packages**
- **For example, Langchain helps run RAG applications by:**
  - accessing PDF files or URL text as raw documents
  - creating database of documents (split into chunks and vectorized)
  - Setting up prompts that include relevant contexts



Langchain:

- **Not an LLM provider (like openAI, or huggingface) but provides a standard interface**
- **Has ecosystem of tools beyond RAG – e.g. Agents:** “a language model is used as a reasoning engine to determine which actions to take and in which order.”

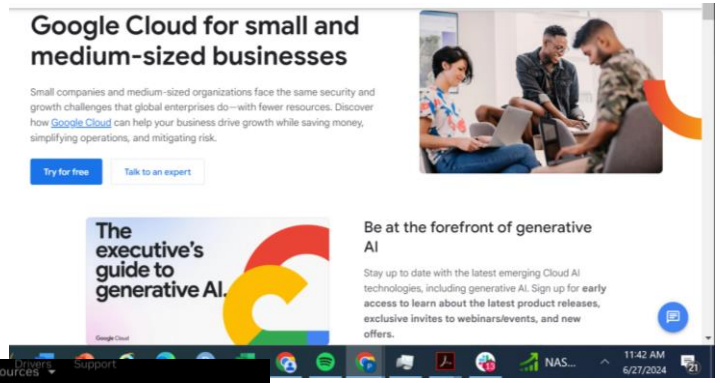


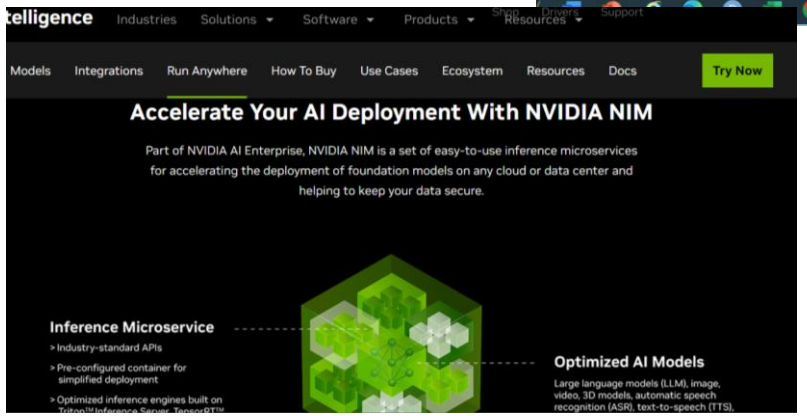


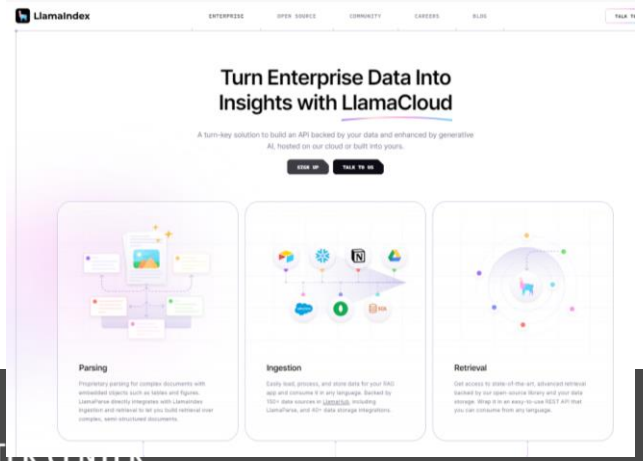
Other ecosystems to run LLMs

Many Big Tech firms have LLM services that provide ‘turn key’ solutions/tools for RAG, with APIs, open models, interfaces, deployment., etc.. geared for businesses.

- **Nvidia (NeMo)**
- **Google (cloud) AI**
- **LlamaIndex (meta)**
- **Etc…**







Keras extension for NLP

- **KerasNLP also has models, data, functions to run different LLMs (no tokens)**

- **Includes pre-trained LLMs base or full model, for example:**

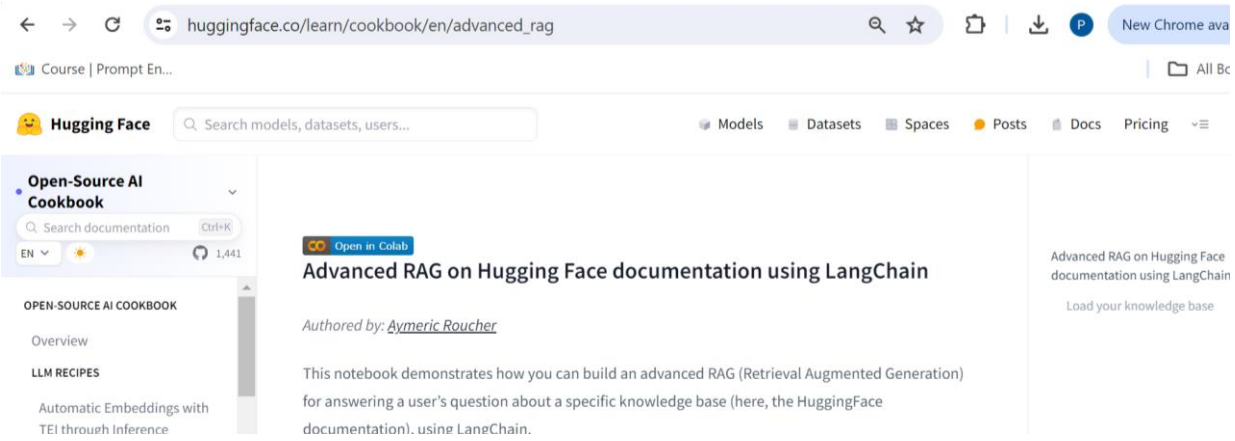
  - GPT2Backbone the model without task specific output layers
  - GPT2CausalLM the model with output predictions
  - GPT2CausalLMPreprocessor the preprocessor that feeds model.fit

Expanse Demo/Exercise setting up HugginFace and Langchain

Let’s combine 2 Jupyter notebook LLM tutorials

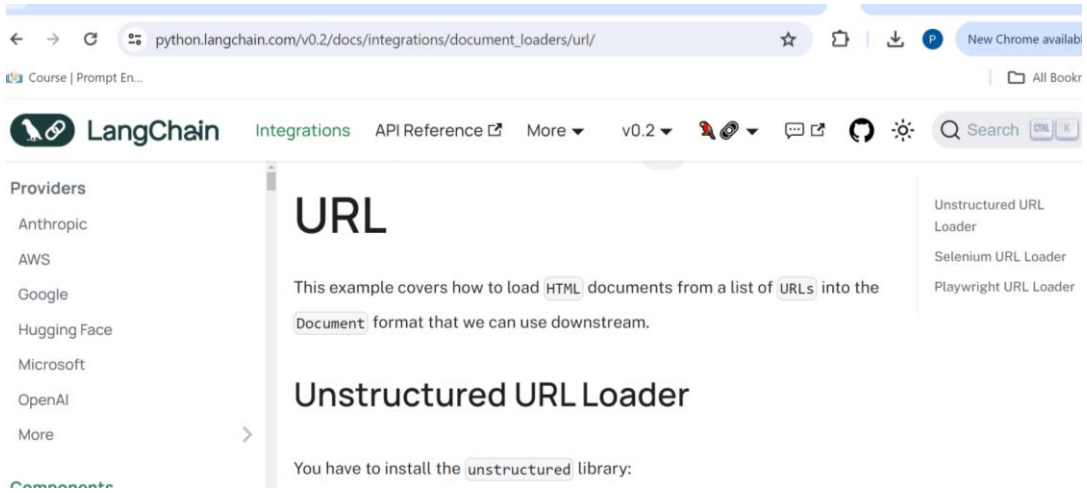
- Start with Huggingface/Langchain for RAG

https://huggingface.co/learn/cookbook/en/advanced_rag

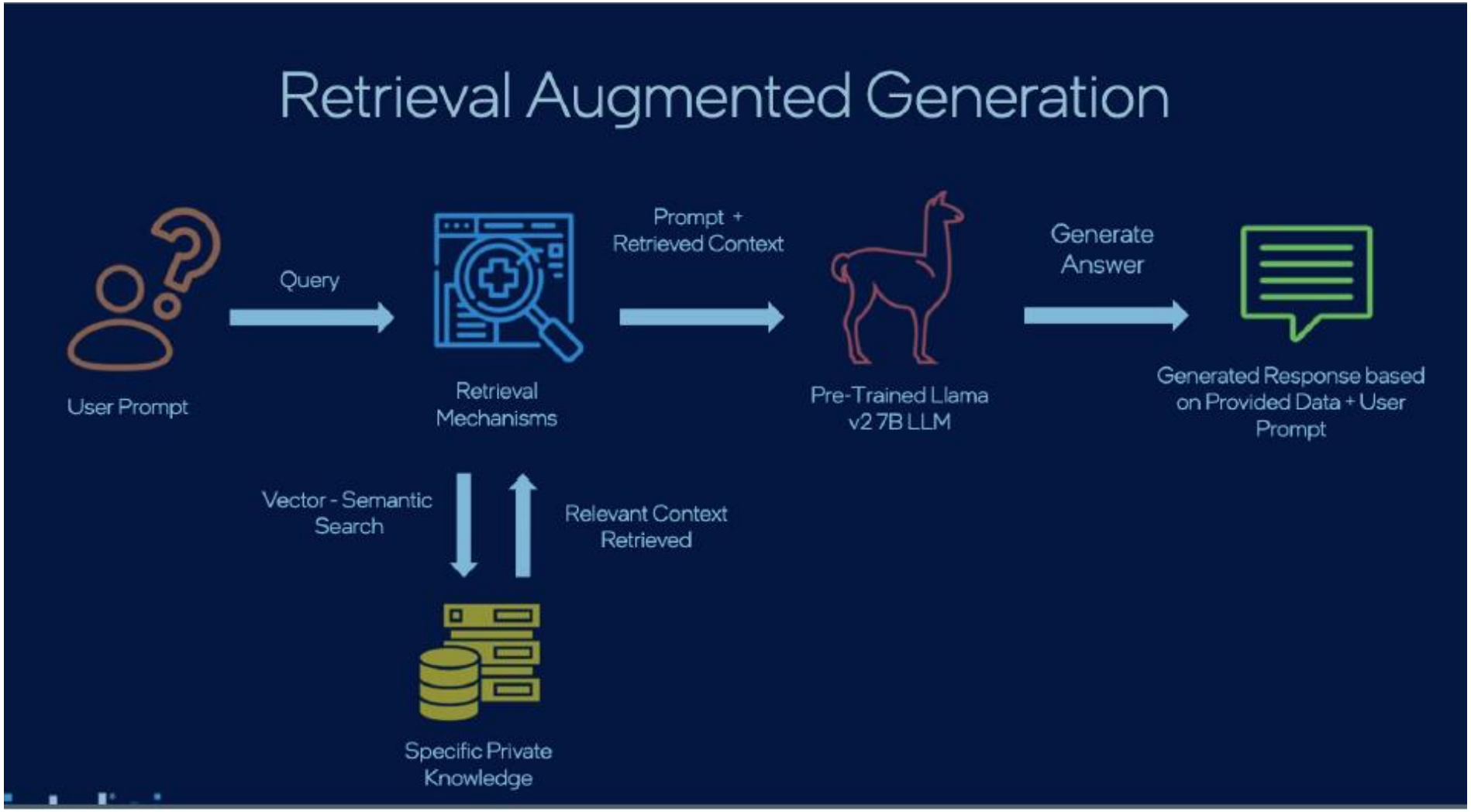


- But use a URL about SLURM for the raw documents

https://python.langchain.com/v0.2/docs/integrations/document_loaders/url/



RAG overview (from the Mai's slide)



Running notebook tutorials on Expanse

- On Expanse we can run a singularity container and start up Jupyter notebooks on a GPU

  Pro: easy to follow instructions

  Con: some things not obvious

Running notebook tutorials on Expanse

- On Expanse we can run a singularity container and start up Jupyter notebooks on a GPU

  Pro: easy to follow instructions

  Con: some things not obvious

OR

- On Expanse we can get a GPU node and use the command line to run singularity and set up commands

  Pro: you get to see how/where everything is set up

  Con: more typing

Running notebook tutorials on Expanse

- On Expanse we can run a singularity container and start up Jupyter notebooks on a GPU

  Pro: easy to follow instructions

  Con: some things not obvious

OR

- On Expanse we can get a GPU node and use the command line to run singularity and set up commands

  Pro: you get to see how/where everything is set up

  Con: more typing

  added benefit: directly translates to a batch script

Hugging Face set up using a notebook

- **Many tutorials use Jupyter notebooks with “!pip install” commands**

In [ ]:
```
!pip install --upgrade 'huggingface_hub[pytorch,cli]'
!pip install transformers
!pip install accelerate
```

In [ ]:
```
#restart kernel to reload .local modules after instdalling hugging face hub
import huggingface_hub
from transformers import AutoTokenizer
import transformers
import torch
```

Packages will be in /home/userid/.local

After installing (~5 min) restart kernel to reset variables

Hugging Face set up using a notebook

- **Many tutorials use Jupyter notebooks with “!pip install” commands**

In [ ]:
```
!pip install --upgrade 'huggingface_hub[pytorch,cli]'
!pip install transformers
!pip install accelerate
```

In [ ]:
```
#restart kernel to reload .local modules after instdalling hugging face hub
import huggingface_hub
from transformers import AutoTokenizer
import transformers
import torch
```

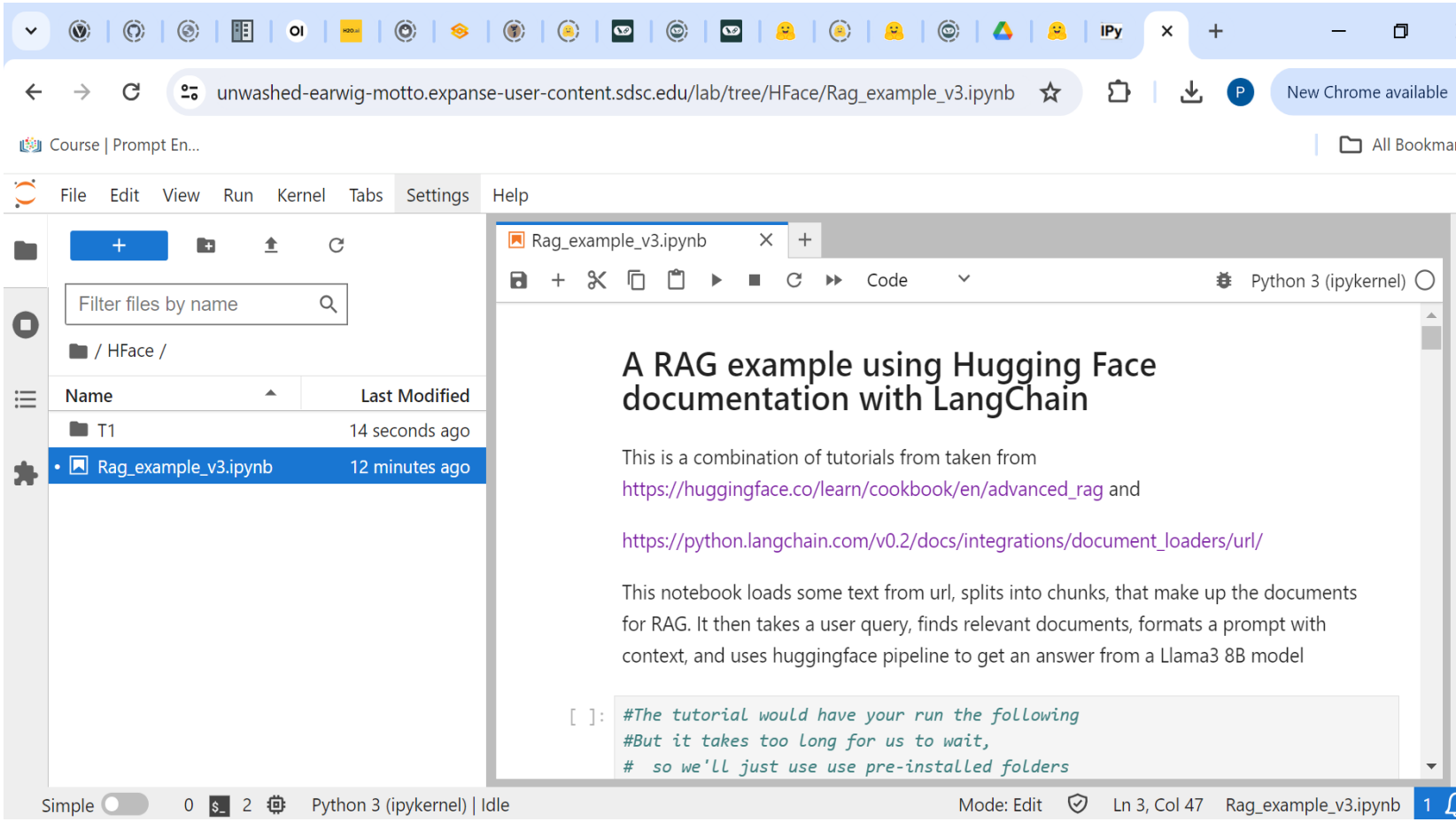
Packages will be in /home/userid/.local

After installing (~5 min) restart kernel to reset variables

BEWARE, you might create conflicts with other set ups using .local

exercise

- jupyter_gpu_shared_pytorch
- Review and run cells
- Try changing chunk size?



Sample output

File | C:/Users/Paul/Documents/Work2019/Expanse/CIML_wkshop_2024/Rag_example_v3.html

```
partitions, which group nodes into logical (possibly overlapping) sets,
jobs, or allocations of resources assigned to a user for
a specified amount of time, and
job steps, which are sets of (possibly parallel) tasks within a job.
The partitions can be considered job queues, each of which has an assortment of
constraints such as job size limit, job time limit, users permitted to use it, etc.
Priority-ordered jobs are allocated nodes within a partition until the resources
(nodes, processors, memory, etc.) within that partition are exhausted. Once
a job is assigned a set of nodes, the user is able to initiate parallel work in
the form of job steps in any configuration within the allocation. For instance,
a single job step may be started that utilizes all nodes allocated to the job,
or several job steps may independently use a portion of the allocation.

Figure 2. Slurm entities

CommandsDocument 4:::
information about Slurm nodes and partitions. slurm Slurm system overview. sprio View the factors that comprise a job's scheduli
ng priority. squeue View information about jobs located in the Slurm scheduling queue. sreport Generate reports from the slurm a
ccounting data. srun Run parallel jobs. sshare Tool for
---
Now here is the question you need to answer.

Question: How to create a slurm job?<|eot_id|><|start_header_id|>assistant<|end_header_id|>

To create a Slurm job, you can use the `sbatch` command, which is used to submit a job script for later execution. The script wi
ll typically contain one or more `srun` commands to launch parallel tasks.
```

In [ ]:

Hugging Face set up using the command line

Main Steps

1. Request a GPU node
2. ssh into that node
3. Load modules
4. Run “singularity shell” command
5. Run “pip install” commands
6. Login to hugging face
7. Run your script

After you this a while, you might prefer to work from command line to see all the dettails

Hugging Face set up using the command line

Main Steps

1. **Request a GPU node**

```
[train113@login02 ciml-summer-institute-2024]$
[train113@login02 ciml-summer-institute-2024]$ jupyter-gpu-shared-pytorch
```

2. **ssh into that node**

```
[p4rodrig@login01 HFace]$ squeue -u p4rodrig
             JOBID PARTITION     NAME     USER ST       TIME  NODES NODELIST(REASON)
          31537549 gpu-debug hface-gp p4rodrig  R       0:01      1 exp-7-59
[p4rodrig@login01 HFace]$
[p4rodrig@login01 HFace]$
[p4rodrig@login01 HFace]$ ssh exp-7-59
```

Hugging Face set up using the command line

Main Steps

3. **Load modules**

```
[p4rodrig@exp-7-59 HFace]$ module purge
[p4rodrig@exp-7-59 HFace]$ module load gpu
[p4rodrig@exp-7-59 HFace]$ module load slurm
[p4rodrig@exp-7-59 HFace]$
[p4rodrig@exp-7-59 HFace]$ module load singularitypro/3.11
```

4. **Run “singularity shell” command**

```
[p4rodrig@exp-7-59 ~]$ singularity shell /cm/shared/apps/containers/singularity/pytorch
h-latest.sif --nv --bind /expanse,/scratch
Singularity>
```

Hugging Face set up using the command line

Main Steps

5. Run "pip install" commands (takes ~5minutes)

```
Singularity> pip install --upgrade huggingface_hub[pytorch,cli] transformers
ts
pip install --upgrade langchain sentence-transformers langchain-community
pip install --upgrade bitsandbytes pypdf faiss-gpu pydantic
```

6. Login to hugging face

```
Singularity> /home/p4rodrig/.local/bin/huggingface-cli login --token hf_cxOE
```

7. Run your script

```
Singularity> python3 Rag_example_v1.py > rag_stdout.txt
```

Hugging Face set up using the command line

Other Steps

Review output:

```
 Now here is the question you need to answer.

 Question: How to create a slurm job?<|eot_id|><|start_header_id|>assistant<|end
_header_id|>

According to the provided documents, to create a Slurm job, you can use the `sbat
ch` command. This command is used to submit a job script for later execution. The
 script will typically contain one or more `srun` commands to launch parallel tas
ks.

Here is an example of how to use `sbatch`:

`sbatch [options] script.sh`

Where `script.sh` is the name of the job script file, and `[options]` are optiona
l parameters that can be used to specify various settings for the job.

For more information on how to use `sbatch` and other Slurm commands, you can ref
er to the Slurm documentation, which is available in the provided documents.
```

Hugging Face set up using the command line

Other Steps

Review cached models in ~/.cach/huggingface/hub

```
[p4rodrig@login02 hub]$ du -sh *
417M    models--bert-base-cased
572K    models--bert-base-uncased
512     models----home--p4rodrig--HW2-tests--Yelp1m--bert-it-
-bert-it-vocab.txt
13G     models--meta-llama--Llama-2-7b-chat-hf
15G     models--meta-llama--Meta-Llama-3-8B-Instruct
512     version.txt
[p4rodrig@login02 hub]$
```

Hugging Face in a batch script

Other Steps

move .local to new folder, and export PYTHONPATH to avoid potential conflicts

Use “singularity exec” to launch commands in a batch script

```
# Set up paths for python to find pacakges
export PYTHONPATH=/home/$USER/Local_HFgpu-latest/lib/python3.1
export PATH=/home/$USER/Local_HFgpu_latest/local/bin:$PATH
echo "--------------- paths -------------------"
echo $PYTHONPATH
echo $PATH

#You can run hugging face login first (it will put the token i
# and also run it within singularity (b/c it was installed tha
singularity exec --nv --bind /expanse,/scratch /cm/shared/apps
pytorch-latest.sif /home/$USER/Local_HFgpu-latest/bin/huggingf


#Now you can run the rag example
singularity exec --nv --bind /expanse,/scratch /cm/shared/apps
pytorch-latest.sif python3 Rag_example_v1.py > rag_stdout.txt
```

Text for command lines

- **The above instructions are in “CommandLineTexts.txt” file**

```
[p4rodrig@login02 HFace]$ more CommandLineTexts.txt
1 Request a GPU node
/cm/shared/apps/sdsc/galyleo/galyleo.sh launch -A use300 -p gpu-debug -n 1 -
 -G 1 -t 00:30:00 -e singularitypro/3.11 --nv --bind /expanse,/scratch -s /c
ared/apps/containers/singularity/pytorch/pytorch-latest.sif

NOTE, change use300 to guest
```

- **Also look at the run-pyt-gpu.. batch script**

```
[p4rodrig@login02 HFace]$ more run-pyt-gpudebug-latest.sh
#!/usr/bin/env bash
#SBATCH --job-name=hface-gpu
#SBATCH --account=use300
# ----------------------------
#SBATCH --partition=gpu-debug
```

Hugging Face set up using the command line

Other details like:

Login to hugging face

```
Singularity> /home/p4rodrig/.local/bin/huggingface-cli login --token hf_cxOE
```

Review cached models in ~/.cach/huggingface/hub

```
[p4rodrig@login02 hub]$ du -sh *
417M    models--bert-base-cased
572K    models--bert-base-uncased
512     models----home--p4rodrig--HW2-tests--Yelp1m--bert-it-
-bert-it-vocab.txt
13G     models--meta-llama--Llama-2-7b-chat-hf
15G     models--meta-llama--Meta-Llama-3-8B-Instruct
512     version.txt
[p4rodrig@login02 hub]$
```

end