Fine-tuning an LLM model with H2O LLM Studio to generate Cypher statements

Avoid depending on external and ever changing APIs for your knowledge graph based chatbot

Tomaz Bratanic · Follow
Published in Towards Data Science
8 min read · Apr 24

Photo by Mike Hindle on Unsplash

Large language models like ChatGPT have a knowledge cutoff date beyond which they are not aware of any events that happened later. Instead of fine-tuning models with later information, the trend is to provide additional external context to LLM at query time. I have written a couple of blog posts on implementing a context-aware knowledge graph-based bot to a bot that can read through the company's resources to answer questions. However, I have used OpenAI's large language models in all of the examples so far

While OpenAI's official position is that they don't use users' data to improve their models, there are stories like how Samsung employees leaked top secret data by inputting it into ChatGPT. If I were dealing with top-secret, proprietary information, I would stay on the safe side and not share that information with OpenAI. Luckily, new open-source LLM models are popping up every day.

I have tested many open-source LLM models on their ability to generate Cypher statements. Some of them have a basic understanding of Cypher syntax. However, I haven't found any models reliably generating Cypher statements based on provided examples or graph schema. So, the only solution was to fine-tune an open-sourced LLM model to generate Cypher statements reliably.

I have never fine-tuned any NLP model, let alone an LLM. Therefore, I had to find a simple way to get started without first obtaining a Ph.D. in machine learning. Luckily, I stumbled upon H2O's LLM Studio tool, released just a couple of days ago, which provides a graphical interface for fine-tuning LLM models. I was delighted to discover that fine-tuning an LLM no longer required me to write any code or long bash commands. With just a few mouse clicks, I would be able to complete the task.

GitHub - h2oai/h2o-llmstudio: H2O LLM Studio - a framework and no-code GUI for fine-tuning LLMs

Welcome to H2O LLM Studio, a framework and no-code GUI designed for fine-tuning state-of-the-art large language models...

github.com

All the code of this blog post is available on GitHub.

Preparing a training dataset

First, I had to learn how the training dataset should be structured. I examined their tutorial notebook and discovered that the tool could handle training data provided as a CSV file, where the first column includes user prompts, and the second column contains desired LLM responses.

Search this file...

	instruction	output
2	Who directed The Matrix?	MATCH (m:Movie {title: 'The Matrix'})<-[:DIRECTED]-(d:Person) RETURN {director: d.nar

llm_train.csv hosted with ❤ by **GitHub** **view raw**

Ok, that's easy enough. Now I just had to produce the training examples. I decided that 200 is a good number of training examples. However, I am way too lazy to write 200 Cypher statements manually. Therefore, I employed GPT-4 to do the job for me. The code can be found here:

blogs/LLM_train_dataset.ipynb at master · tomasonjo/blogs

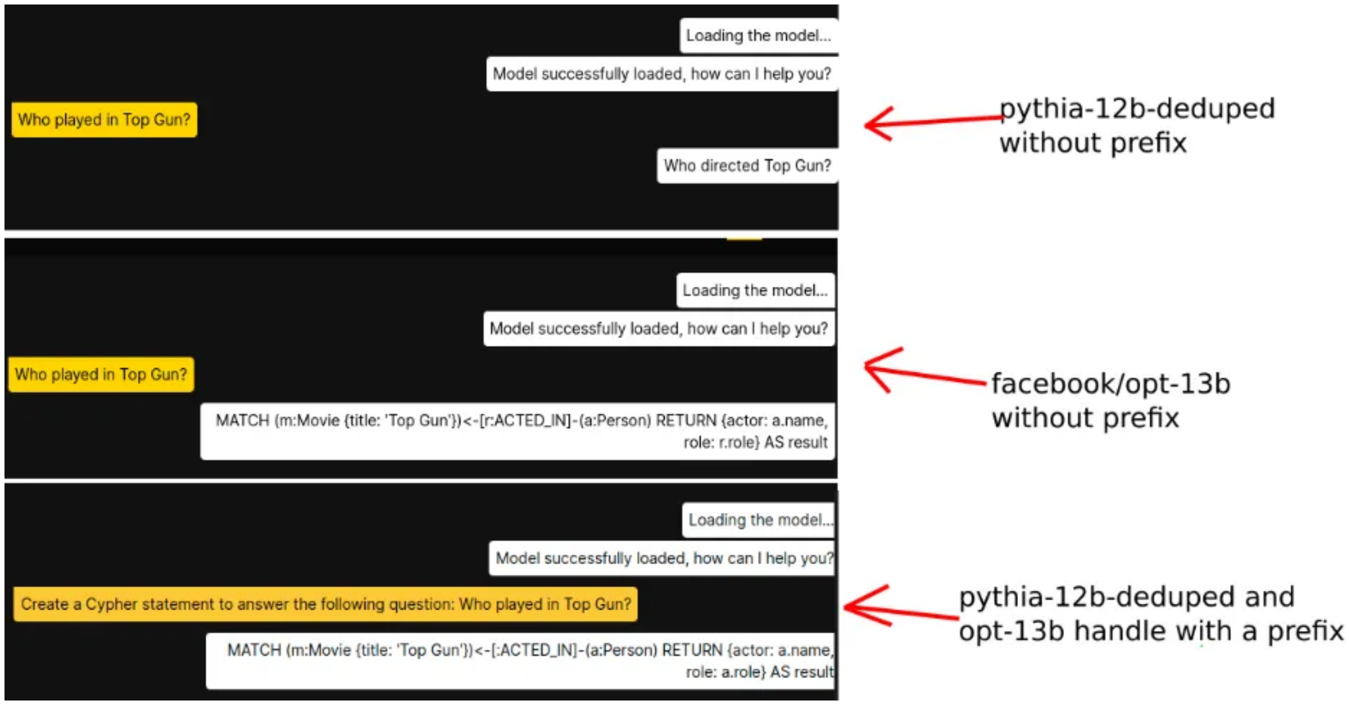
You can't perform that action at this time. You signed in with another tab or window. You signed out in another tab or…

github.com

The movie recommendation dataset is baked into GPT-4, so it can generate good enough examples. However, some examples are slightly off and don't fit the graph schema. So, if I were fine-tuning an LLM for commercial use, I would use GPT-4 to generate Cypher statements and then walk through manually to validate them. Additionally, I would want to ensure that the validation set contains no examples from the training set.

I have also tested if a prefix "Create a Cypher statement for the following question" is needed for instructions. It seems that some models like **EleutherAI/pythia-12b-**

deduped need the prefix, otherwise they fail miserably. On the other hand, **facebook/opt-13b** did a solid job even without the prefix.


Loading the model...
Model successfully loaded, how can I help you?
Who played in Top Gun?
Who directed Top Gun?

Loading the model...
Model successfully loaded, how can I help you?
Who played in Top Gun?
MATCH (m:Movie {title: 'Top Gun'})<-[r:ACTED_IN]-(a:Person) RETURN {actor: a.name, role: r.role} AS result

Loading the model...
Model successfully loaded, how can I help you?
Create a Cypher statement to answer the following question: Who played in Top Gun?
MATCH (m:Movie {title: 'Top Gun'})<-[:ACTED_IN]-(a:Person) RETURN {actor: a.name, role: a.role} AS result

Models trained without or with a prefix in instructions. Image by the author.

To be able to compare all models with the same dataset, I used a dataset that adds a prefix “Create a Cypher statement for the following question:” to the instructions section of the dataset.

H2O LLM Studio installation

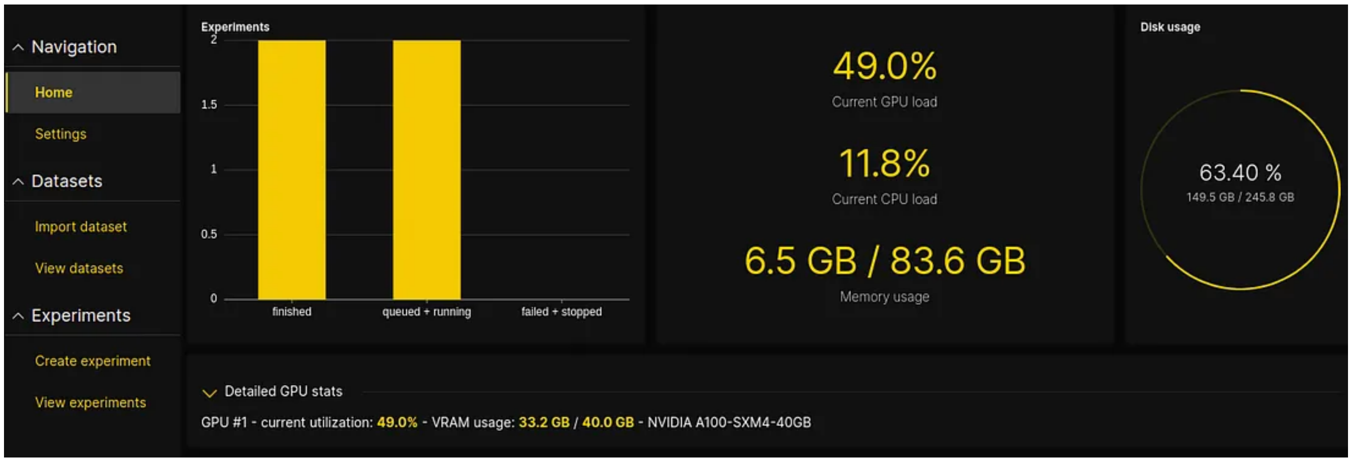
H2O LLM Studio can be installed in two simple steps. In the first step, we have to install Python 3.10 environment if it is missing. The steps to install Python 3.10 are described in their GitHub repository.

GitHub - h2oai/h2o-llmstudio: H2O LLM Studio - a framework and no-code GUI for fine-tuning LLMs

Welcome to H2O LLM Studio, a framework and no-code GUI designed for fine-tuning state-of-the-art large language models...

github.com

After we ensure a Python 3.10 environment, we simply clone the repository and install dependencies with the `make install` command. After the installation, we can run the LLM studio with the `make wave` command. Now we can open the graphical interface in your favourite browser by opening the `localhost:10101` website.

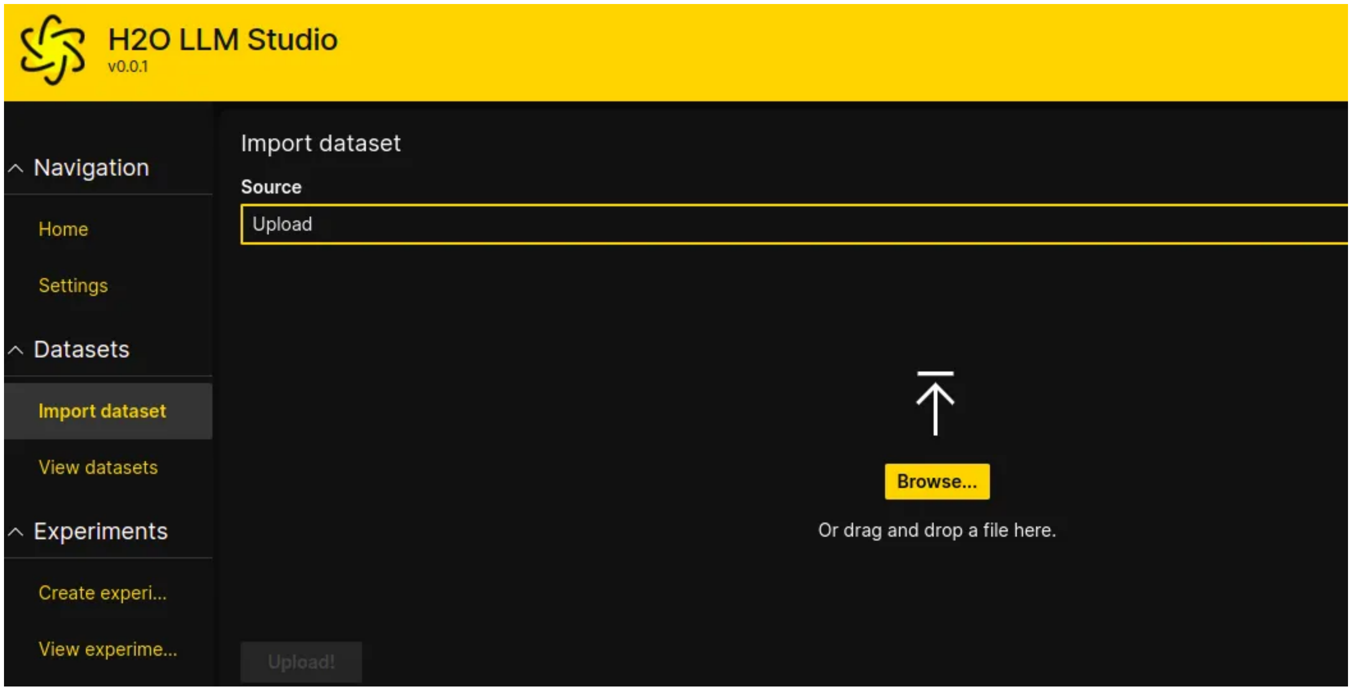


GPU #1 - current utilization: 49.0% - VRAM usage: 33.2 GB / 40.0 GB - NVIDIA A100-SXM4-40GB

H2O LLM Studio home page. Image by the author.

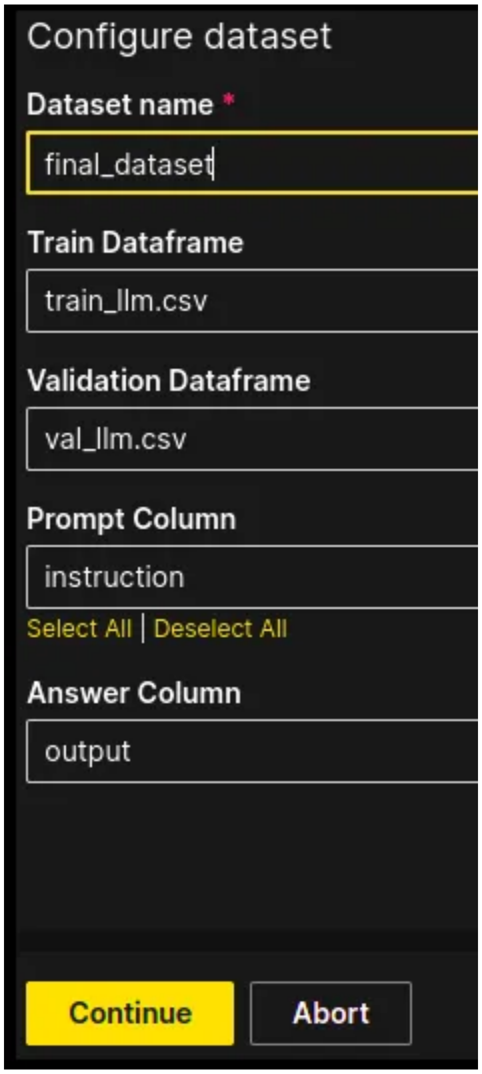
Import dataset

First, we have to import the dataset to be used to fine-tune an LLM. You can download the one I used if you don't want to create your dataset. Note that it is not curated, and some examples do not fit the movie recommendation graph schema. However, it is a great start to getting to know the tool. We can import CSV files using the drag&drop interface.



Upload CSV interface. Image by the author.

It is a bit counter-intuitive, but we have to upload the training and validation sets separately. Let's say we first upload the training set. Then, when we upload the validation set, we have to use the merge datasets option so that we have both the training and validation sets in the same dataset.


Configure dataset

Dataset name *
final_dataset

Train Dataframe
train_llm.csv

Validation Dataframe
val_llm.csv

Prompt Column
instruction

Answer Column
output

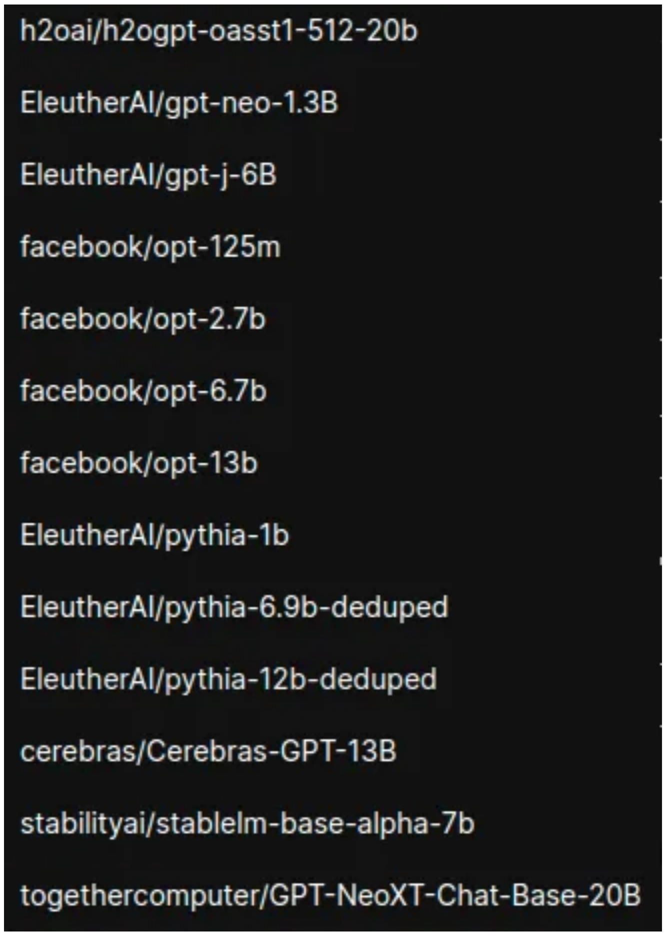
Imported dataset with both train and validation dataframes. Image by the author.

The final dataset should have both training and validation dataframes present.

I've learned you can also upload a ZIP file with both training and validation sets to avoid having to separately upload files.

Create experiment

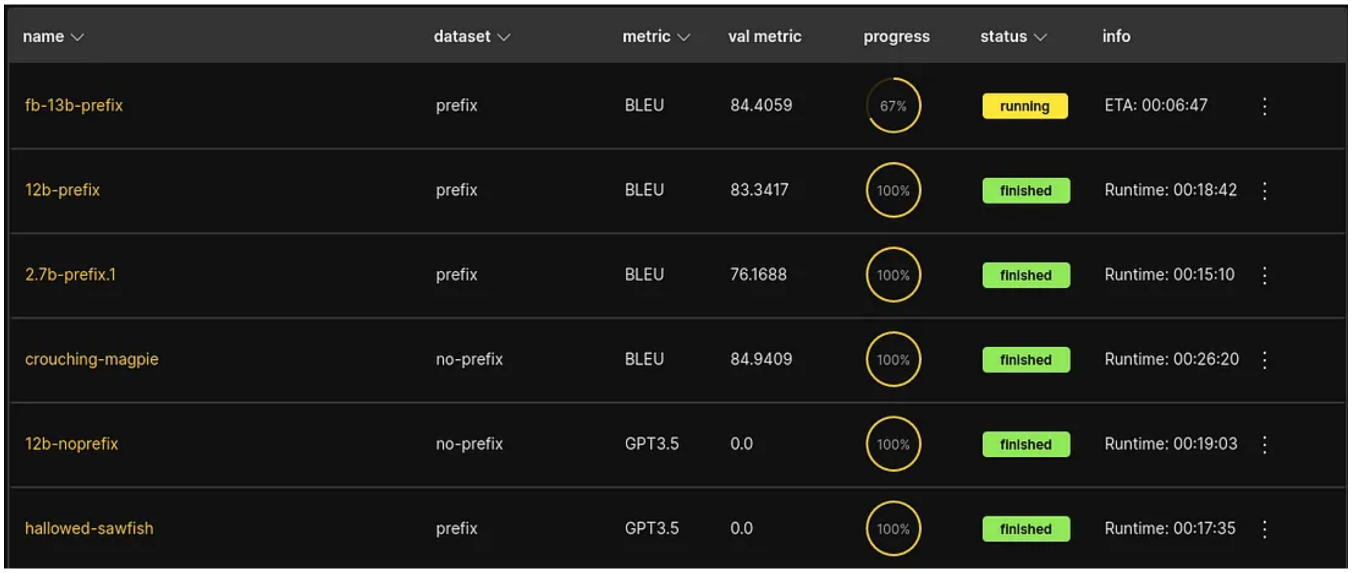
Now that everything is ready, we can go ahead and fine-tune an LLM model. If we click on the **Create Experiment** tab, we will be presented with fine-tuning options. The most important setting to choose are the **dataset** used for training, the **LLM backbone**, and I have also increased the **epochs** count in my experiments. I have left the other parameters default as I have no idea what they do. We can choose from 13 LLM models:



h2oai/h2ogpt-oasst1-512-20b
EleutherAI/gpt-neo-1.3B
EleutherAI/gpt-j-6B
facebook/opt-125m
facebook/opt-2.7b
facebook/opt-6.7b
facebook/opt-13b
EleutherAI/pythia-1b
EleutherAI/pythia-6.9b-deduped
EleutherAI/pythia-12b-deduped
cerebras/Cerebras-GPT-13B
stabilityai/stablelm-base-alpha-7b
togethercomputer/GPT-NeoXT-Chat-Base-20B

Available LLM models. Image by the author.

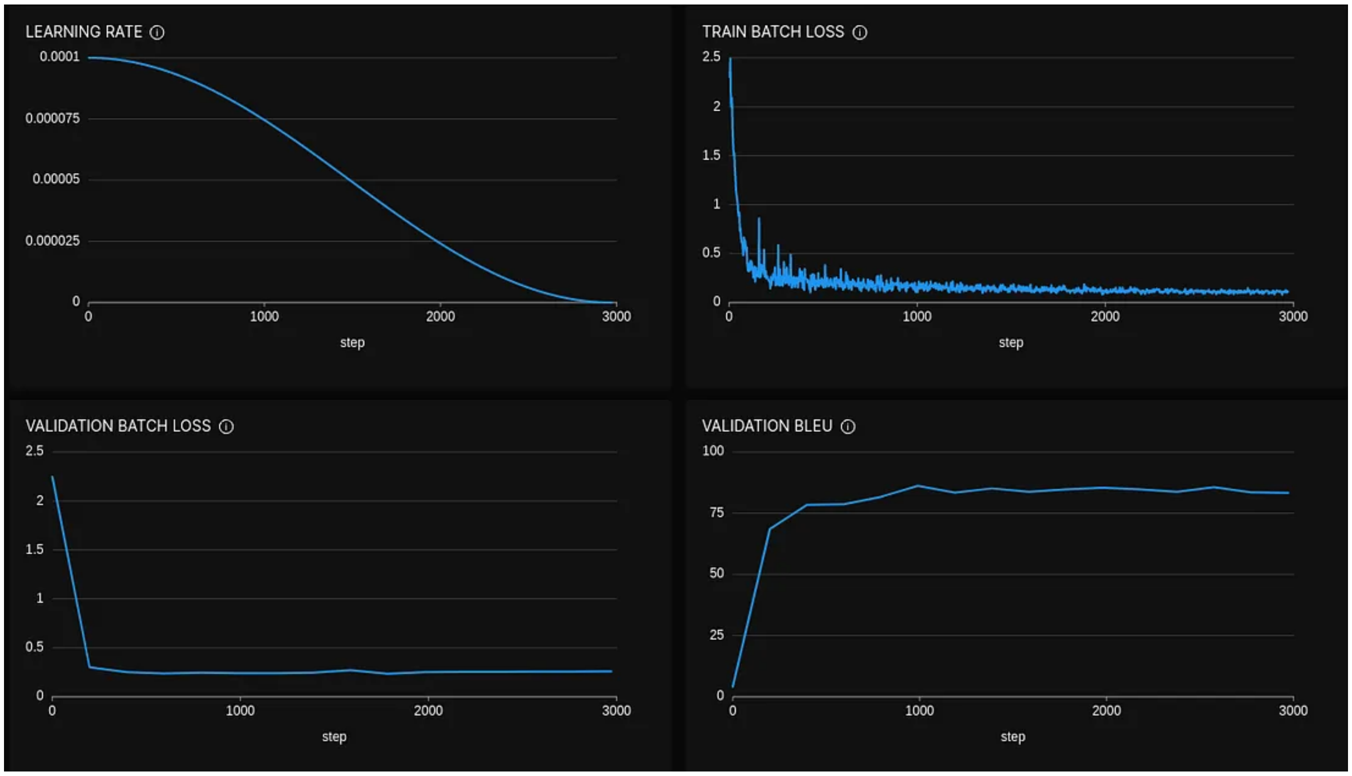
Note that the higher the parameter count, the more GPU RAM we require for finetuning and inference. For example, I ran out of memory using a 40GB GPU when trying to finetune an LLM model with 20B parameters. On the other hand, we expect that the higher the parameter count of an LLM, the better the results. I would say that we require about 5GB of GPU RAM for smaller LLMs like pythia-1b and up to 40GB GPU for opt-13b models. Once we set the desired parameters, we can run the experiment with a single click. For the most part, the finetuning process was relatively fast using an Nvidia A100 40GB.



name	dataset	metric	val metric	progress	status	info
fb-13b-prefix	prefix	BLEU	84.4059	67%	running	ETA: 00:06:47
12b-prefix	prefix	BLEU	83.3417	100%	finished	Runtime: 00:18:42
2.7b-prefix.1	prefix	BLEU	76.1688	100%	finished	Runtime: 00:15:10
crouching-magpie	no-prefix	BLEU	84.9409	100%	finished	Runtime: 00:26:20
12b-noprefix	no-prefix	GPT3.5	0.0	100%	finished	Runtime: 00:19:03
hallowed-sawfish	prefix	GPT3.5	0.0	100%	finished	Runtime: 00:17:35

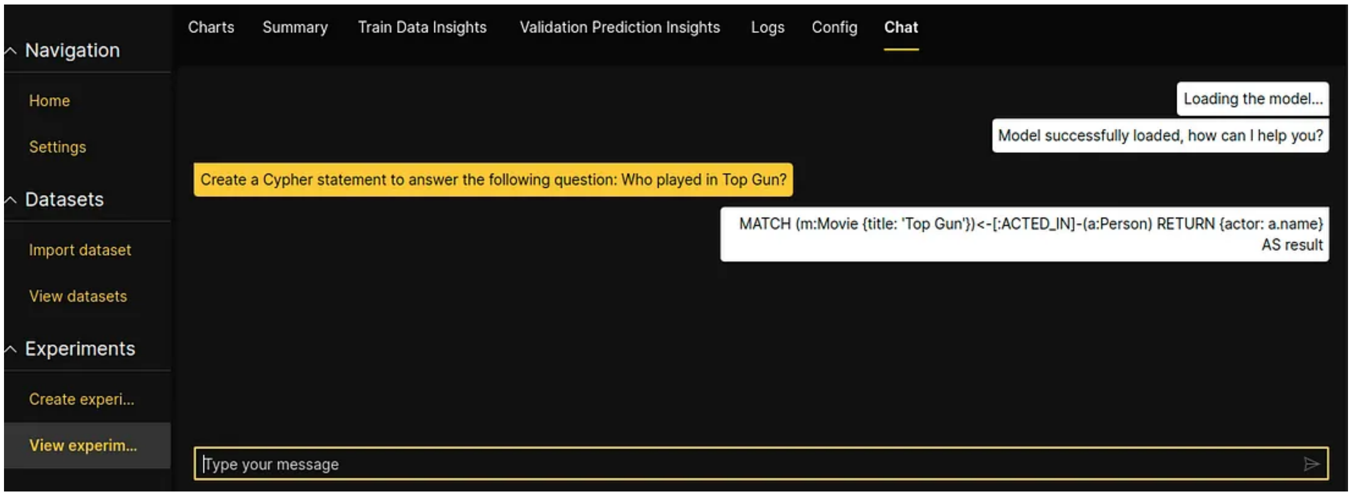
Experiments page. Image by the author.

Most models were trained in less than 30 minutes using 15 epochs. The nice thing about the LLM Studio is that it produces a dashboard to inspect the training results.



LLM finetuning metrics. Image by the author.

Not only that, but we can also chat with the model in the graphical interface.



Loading the model...

Model successfully loaded, how can I help you?

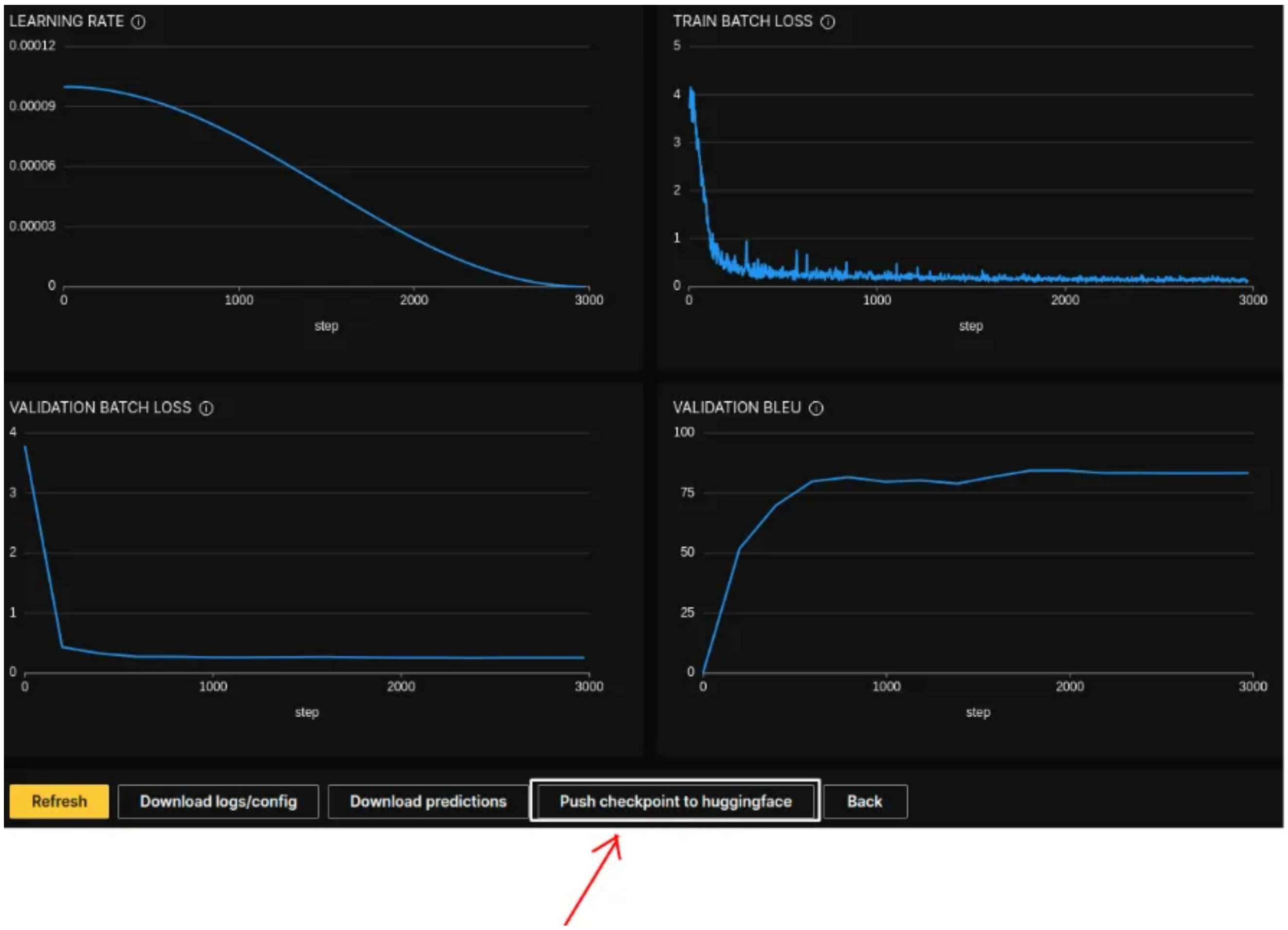
Create a Cypher statement to answer the following question: Who played in Top Gun?

```
MATCH (m:Movie {title: 'Top Gun'})<-[:ACTED_IN]-(a:Person) RETURN {actor: a.name}
AS result
```

Chat interface in the LLM Studio. Image by the author.

Export models to HuggingFace repository

It's as if the H2O LLM Studio wasn't cool enough, it also allows to export finetuned models to HuggingFace with a single click.



Export models to HuggingFace. Image by the author.

The ability to export a model to the HuggingFace repository with a single click allows us to use the model anywhere in our workflows as easily as possible. I have exported a small finetuned pythia-1b model that can run in Google Colab to demonstrate how to use it with the transformers library.

```
import torch
from transformers import AutoModelForCausalLM, AutoTokenizer

device = "cuda:0" if torch.cuda.is_available() else "cpu"

tokenizer = AutoTokenizer.from_pretrained("tomasonjo/movie-generator-small")
model = AutoModelForCausalLM.from_pretrained("tomasonjo/movie-generator-small")
    device
)

prefix = "\nCreate a Cypher statement to answer the following question:"

def generate_cypher(prompt):
    inputs = tokenizer(
        f"{prefix}{prompt}<|endoftext|>", return_tensors="pt", add_special_toke
    ).to(device)
    tokens = model.generate(
        **inputs,
        max_new_tokens=256,
        temperature=0.3,
        repetition_penalty=1.2,
        num_beams=4,
    )[0]
    tokens = tokens[inputs["input_ids"].shape[1] :]
    return tokenizer.decode(tokens, skip_special_tokens=True)
```

The LLM Studio uses a special `<|endoftext|>` character that must be added to the end of the user prompt in order for the model to work correctly. Therefore, we must do the same when using the finetuned model with the transformers library. Other than that, there is nothing really that needs to be done. We can now use the model to generate Cypher statements.

```
generate_cypher("How many movies did Tom Hanks appear in?")
#MATCH (d:Person {name: 'Tom Hanks'})-[:ACTED_IN]->(m:Movie)
#RETURN {movie: m.title} AS result

generate_cypher("When was Toy Story released?")
#MATCH (m:Movie {title: 'When'})-[:IN_GENRE]->(g:Genre)
#RETURN {genre: g.name} AS result
```

I deliberately showed one valid and one invalid Cypher statement generated to show that the smaller models might be good enough for demos, where the prompts can be predefined. On the other hand, you probably wouldn't want to use them in production. However, using bigger models comes with a price. For example, to run models with 12B parameters, we need at least 24 GB GPU, while the 20B parameter models require GPUs with 48 GB.

Summary

Finetuning open-source LLMs allows us to break free of the OpenAI dependency. Although GPT-4 works better, especially in a conversational setting where follow-up questions could be asked, we can still keep our top-secret data to ourselves. I tested multiple models while writing this blog post, except for 20B models, due to GPU memory issues. I can confidently say that you could finetune a model to generate

Gpt ChatGPT g Llm o Neo4j I Hands On Tutorials One thing to note is that follow-up questions, where the model has to rely on previous dialogue to understand the context of the question, don't seem to be functioning at the moment. Therefore, we are limited to single-step queries, where we need to provide the whole context in a single prompt. However, since the development of open-source LLMs is exploding, I am excited about what's to come next.

Till then, try out the H2O LLM Studio if you want to finetune an LLM to fit your personal or company's needs with only a few mouse clicks.

Follow