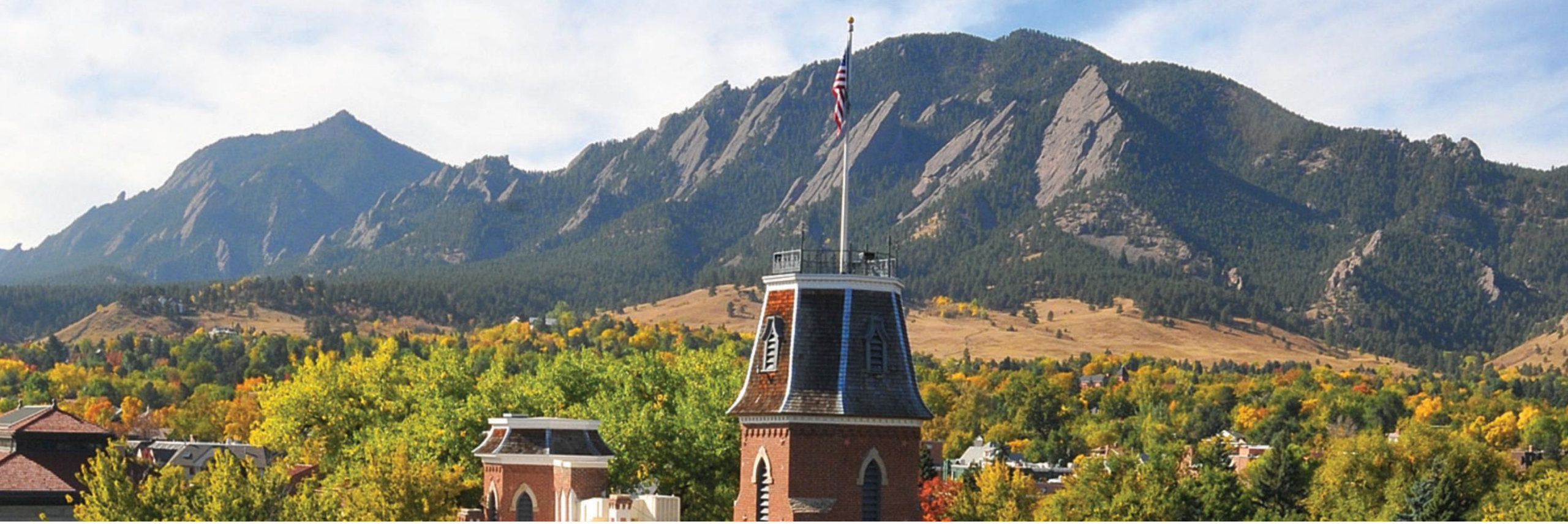


Setting up LLMs on CURC Resources

Research Computing
UNIVERSITY OF COLORADO BOULDER

Setting up LLMs on CURC Resources

Instructor: Brandon Reyes

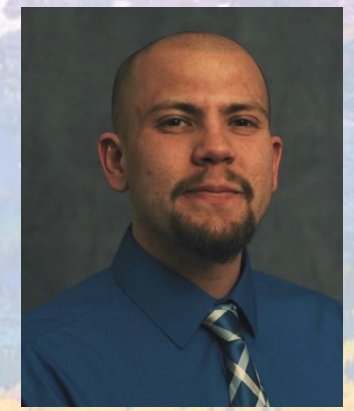
- Research Computing

- Website: www.colorado.edu/rc
- Documentation: https://curc.readthedocs.io
- Helpdesk: support request form
- Survey: http://tinyurl.com/curc-survey18

Research Computing
UNIVERSITY OF COLORADO **BOULDER**

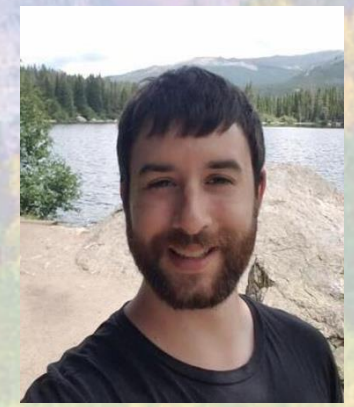
Meet the User Support Team

Layla
Freeborn

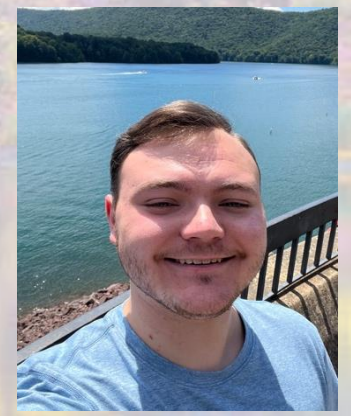


Brandon
Reyes

Andy
Monaghan

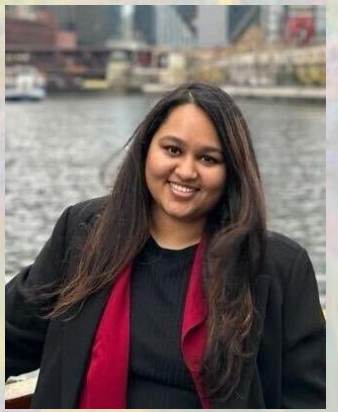


Michael
Schneider

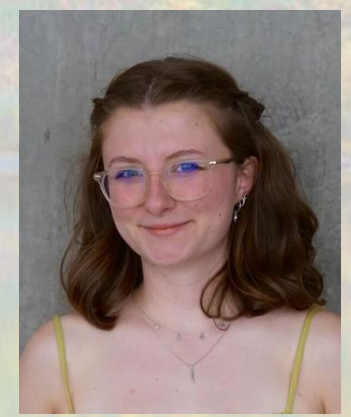


John
Reiland

Dylan
Gottlieb

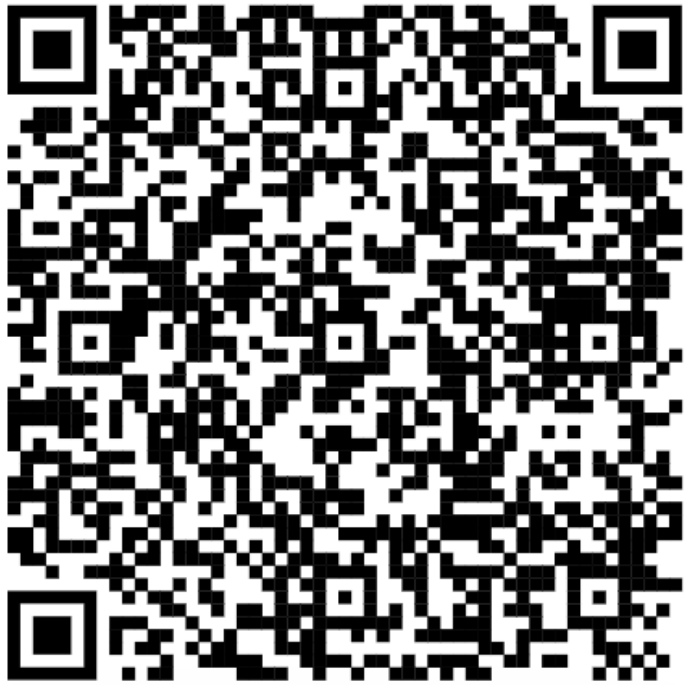


Mohal
Khandelwal



Ragan
Lee

Slides:
https://github.com/ResearchComputing/setting_up_llms_on_curc_resources_rc_short_course



Session Overview

- Background on LLMs
- Accessing CURC hosted LLMs
- Running Ollama
- Running Transformers by Hugging Face

What are LLMs?

Large Language Models (LLMs) are types of artificial intelligence models that are trained on large amounts of data, which enables them to “understand” and generate natural language. They are great at:

- Summarizing material
- Constructing code (if they are trained for coding tasks)
- Conversational tasks
- Generating text

When you break it down, LLMs are made possible by multiplication, addition, and non-linear transformations. This allows them to run very quickly on GPUs.

Important Considerations for LLMs

- Not all LLMs are equal, some are trained for specific tasks, such as code generation
- The type of data the LLMs are trained on determines what they can do
- General models (ChatGPT, Gemini) are great for common knowledge tasks, but can fall short in certain technical areas
- Bigger is not always better
  - You can have small LLMs that are extremely performant for a certain task
- LLMs can “hallucinate”. Put simply, this means they can generate incorrect answers or content that may sound plausible
  - Don’t treat LLMs as if they are all-knowing! They are not … yet

Proprietary vs Open-Source LLMs

- Models such as ChatGPT are proprietary and require that you run the model on their dedicated hardware
  - Pros: these companies have a lot of resources, allowing for quicker access and their models are usually state-of-the-art
  - Cons: **Data privacy and cost**
- Open-source LLMs can be ran anywhere
  - Pros: you can run them for free on our resources, your data stays on our system, and there are some very capable open-source models
  - Cons: proprietary models are not going to be available and complex LLMs can consume a lot of GPU memory

What is an LLM framework?

Now that we know why we would like to run an LLM locally, how do we do that? In the past, running an LLM locally took quite a bit of effort. However, now there are several LLM frameworks available.

An LLM framework allows for a user-friendly way to access LLMs and utilize them

- Can enable access to proprietary or local LLMs
- Can enable running the LLM from the command line or within a script

Common LLM Frameworks

There are many LLM frameworks out there, some with their own unique functionality. However, in this talk we will focus on two extremely popular LLM frameworks that enable you to run models locally:

- Ollama
- Transformers by Hugging Face

Ollama

Ollama is an LLM framework maintained by Meta

- It is open-source and extremely beginner friendly
- Enables users to run LLMs locally and retrieve models
- Easily installable models are curated by Ollama
  - Available models can be seen at https://ollama.com/search
  - You can use other models, but you need to convert them to a compatible format
- With Ollama, you establish a server and then use this server to install, manage, and run LLMs

Transformers by Hugging Face

Transformers is an LLM framework maintained by Hugging Face and is extremely popular

- It is open-source and contains additional functionality that Ollama does not have, such as the ability to train models
- Enables users to run LLMs locally and retrieve models
- Does not require you to setup a server like Ollama

What is Quantization and Instruct?

- Quantization
  - When training, most LLMs use either single or double precision. This can make those models consume a lot of memory.
  - Quantization allows you to reformat the model into a lower-precision format, which makes the model consume less memory.

- Instruct
  - Instruct or instruction models are ideal for specifying tasks for the LLM to perform and are the models used in common chat interfaces.

Before we jump into things …

Before running or working with LLMs on CURC resources, please be sure that you are adhering to all CURC User Policies and CU's policies and guidance around Artificial Intelligence outlined in the pages Artificial Intelligence at CU Boulder and Resources & Guidance.

What resources should you run on?

- Most popular LLMs need a GPU to run in a timely manner
- On our system, we have both AMD and NVIDIA GPUs
  - The modules we have on our system for Ollama and Transformers are only compatible with CPU and NVIDIA GPU nodes
- We recommend that you run on our aa100, al40, or atesting_a100 partitions
  - The atesting_a100 partition is only for testing, debugging, and compiling programs
  - The al40 partition has L40 GPUs that can be great for inference (evaluating LLMs)
  - The aa100 partition has A100 GPUs that can be better for training LLMs

atesting_a100 interactive session example:

```
$ sinteractive --partition=atesting_a100 --qos=testing --nodes=1 --ntasks=10 --gres=gpu --time=60
```

Accessing CURC hosted LLMs

To streamline access and reduce redundant storage of LLMs, CURC has created a shared space for common LLMs

- This is a CURC managed space, you cannot store models in this space
  - If you do think a model should be here, reach out to us!
- All models can be accessed using the environment variable CURC_LLM_DIR
  - You must be on a compute node to get access to this variable and the provided LLMs

CURC hosted Ollama Compatible LLMs

- gpt-oss:20b
  - An open-source model from OpenAI (ChatGPT folks)
  - Requires about 14 GB of GPU memory
- gemma3:12b
  - An open-source model from Google (a derivative of Gemini)
  - Requires around 8 GB of GPU memory
- llama3.1:8b
  - An open-source model from Meta
  - Requires about 5 GB of GPU memory
- embeddinggemma:latest
  - An open-source embedding model from Google (used for generating vector representations, not for chatting)
  - Requires about 1 GB of GPU memory

All of these are quantized, instruct models (only non-embedding models), and available at:

$CURC_LLM_DIR/ollama

CURC hosted Transformer Compatible LLMs

- gpt-oss-20b
  - This model has been quantized
  - Requires about 14 GB of GPU memory
- gemma-3-12b-it
  - This model has not been quantized
  - Without quantization, requires more than 20 GB of GPU memory
- Llama-3.1-8B-Instruct
  - This model has not been quantized
  - Without quantization, requires more than 20 GB of GPU memory

All of these models are instruct models and available at:

$CURC_LLM_DIR/hf-transformers

Using CURC's Ollama install

The latest version of Ollama is available via our module stack. Once on a compatible compute node, you can load the module using:

```
module load ollama
```

- Starts an Ollama server for you and sets all necessary environment variables
- Uses CURC's hosted LLMs by default

If you would like to point to your own directory that contains LLMs, this can be done with:

```
export OLLAMA_MODELS=/projects/$USER/my_ollama_models; module load ollama
```

This module is only available on CPU nodes and NVIDIA GPU nodes. We only provide the most up-to-date version, and these versions will be updated during each planned maintenance

Useful Ollama Commands

- ollama list
  - Lists all available models
- ollama pull <model>
  - Downloads a model if it does not exist, otherwise updates the model
- ollama run <model>
  - Runs the model from the command line and downloads the model if it does not exist
- ollama rm <model>
  - Removes the specified model

Accessing Ollama in Python

To use Ollama within Python, you must install the appropriate packages. We have created a minimal uv environment for this. You can obtain access to this environment using:

```
module load uv
source $CURC_UV_ENV_DIR/ollama-python-api-env/bin/activate
```

Note: The Ollama server needs to be running before you run your code.

Example Ollama Python script

ollama_test.py

```
from ollama import chat
from ollama import ChatResponse

response: ChatResponse = chat(model='llama3.1:8b', messages=[
  {
    'role': 'user',
    'content': 'In one sentence, how cool is CU Research Computing?',
  },
])

print(response.message.content)
```

To run this script from the command line:

(ollama-python-api-env) $ python ollama_test.py

Installing Ollama

As noted, our modules only install the newest version of Ollama that is available during our planned maintenance. Sometimes you need an older or newer version. We provide extremely detailed instructions for installing both Ollama and setting up the Ollama Python package via uv in the “Self-install instructions” tab:

https://curc.readthedocs.io/en/latest/ai-ml/llms.html#ollama

Transformers by Hugging Face

Models are not as easy to install as Ollama

- Pros:
  - Provides access to cutting-edge LLMs and datasets: https://huggingface.co/
  - Provides access to community driven models and models without quantization
- Cons:
  - Most well-known models require a Hugging Face account and the acceptance of terms of use to download
  - If the model uses a different library to run, you have to install those dependencies
  - Community driven models and datasets are sometimes not vetted

Let’s take a look at a model card!

https://huggingface.co/google/gemma-3-12b-it

Using CURC's Transformers install

The latest version of Transformers is available via our module stack. Once on a compatible compute node, you can load the module using:

```
module load hf-transformers
```

- Loads base packages using uv and sets general environment variables
- Unlike Ollama, Transformers isn't connected to CURC's LLM directory (more on this later)

This module is only available on CPU nodes and NVIDIA GPU nodes. We only provide the most up-to-date version, and these versions will be updated during each planned maintenance

Installing Transformers

As noted, our modules only install the newest version of Transformers that is available during our planned maintenance. Sometimes you need an older or newer version. We provide extremely detailed instructions for installing Transformers via uv in the “Self-install instructions” tab:

https://curc.readthedocs.io/en/latest/ai-ml/llms.html#transformers-by-hugging-face

Downloading models

- You must have a Hugging Face account
- You need to generate a user access token

Once your account is setup, you can install models using:

hf download --local-dir <install-directory> <hf-model>

For in-depth instructions, see https://curc.readthedocs.io/en/latest/ai-ml/llms.html#downloading-transformers-compatible-models

Example Transformers Python script

```
from transformers import AutoTokenizer, AutoModelForCausalLM
import os

# Get our system-defined path to CURC's shared LLMs
CURC_LLM_DIR = os.getenv('CURC_LLM_DIR')

# Specify that we want to use the gpt-oss-20b model that is in CURC's shared LLM directory
path_to_model=f"{CURC_LLM_DIR}/hf-transformers/gpt-oss-20b"

# Obtain the model-defined tokenizer
tokenizer = AutoTokenizer.from_pretrained(path_to_model)

# Load the model onto the GPU and do not apply quantization
model = AutoModelForCausalLM.from_pretrained(path_to_model, device_map="cuda")

# Specify the task we want the LLM to perform
messages = [{"role": "user", "content": "Write a paragraph about cookies."}]

# Format the input we will provide to the LLM and load this onto the GPU
inputs = tokenizer.apply_chat_template(messages, add_generation_prompt=True, tokenize=True, return_dict=True,
return_tensors="pt").to(model.device)

# Generate the LLM response
outputs = model.generate(**inputs, max_new_tokens=200)

# Print the provided LLM response
print(tokenizer.decode(outputs[0][inputs["input_ids"].shape[-1]:]))
```

Example Transformers Python script

Let's name our previous script no_quant.py. To run this script, we first need to make sure that the model has been installed. Once the model has been installed, we can run this script from the command line as follows:

```
(hf-transformers-env) $ python no_quant.py
```

Adding Quantization

As mentioned before, some models (Llama-3.1-8B-Instruct) will not fit on a smaller GPU. For this reason, you need to Quantize the model. For those models that use PyTorch, this can be done using Bitsandbytes. This is straightforward and requires only the addition/modifications below:

```
from transformers import BitsAndBytesConfig
import torch

# Use QLoRA or 4-bit quantization
bnb_config = BitsAndBytesConfig( load_in_4bit=True,bnb_4bit_quant_type="nf4",
bnb_4bit_use_double_quant=True, bnb_4bit_compute_dtype=torch.bfloat16 )

# Load the model onto the GPU and apply quantization using bitsandbytes
model = AutoModelForCausalLM.from_pretrained(path_to_model, device_map="cuda",
quantization_config=bnb_config, dtype=torch.bfloat16)
```

Streaming to the command line

While in an interactive session, we can stream the output of the LLM in real-time by modifying the “model.generate” call with the following:

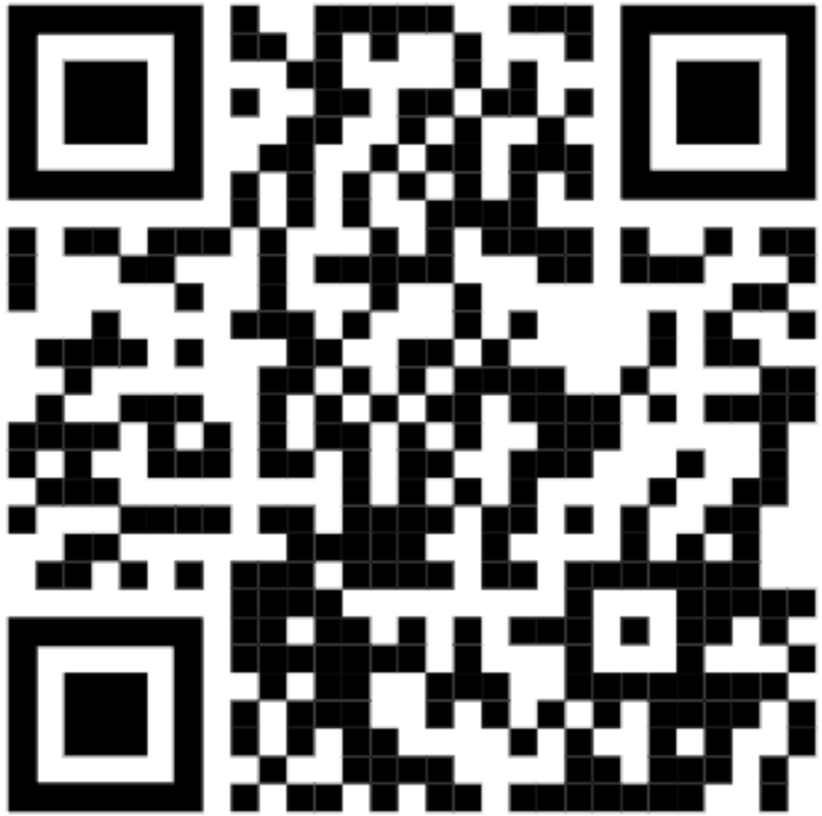
```
from transformers import TextStreamer

streamer = TextStreamer(tokenizer)
output = model.generate(**inputs, max_new_tokens=200, streamer=streamer)
```

Thank you!

Survey and feedback

http://tinyurl.com/curc-survey18



Slides:
https://github.com/ResearchComputing/setting_up_llms_on_curc_resources_rc_short_course