If you have access to a Linux-based supercomputer cluster, you’re in a great position to work with large language models like GPT! Here’s a guide to get you started on using and potentially fine-tuning a GPT model on a supercomputer cluster.

**1. Set Up Your Environment**

• **Access the Cluster**: Log in via SSH if you’re accessing it remotely.

• **Set Up a Virtual Environment**: Keep things organized and avoid conflicts with system-wide packages.

**python3 -m venv gpt-env**

**source gpt-env/bin/activate**

A black screen with white text

Description automatically generated

• **Install Necessary Libraries**:

You’ll need libraries like transformers from Hugging Face, torch for deep learning, and datasets if you’ll be handling large datasets.

**pip install torch transformers datasets**

A screen shot of a computer

Description automatically generated

Note: this may hang up and seem like nothing is happening, if this happens just ctrl+c and run the pip command again and it will pick up where it left off

**2. Choose Your GPT Model**

If you’re just getting started, consider using a smaller model (like GPT-2) to get a feel for it. Hugging Face hosts several pre-trained models, so you can load a model with just a few lines of code.

from transformers import AutoModelForCausalLM, AutoTokenizer

*# Load a model and tokenizer from Hugging Face*

model\_name = "gpt2" *# You could also try "gpt-neo-2.7B" or others*

model = AutoModelForCausalLM.from\_pretrained(model\_name)

tokenizer = AutoTokenizer.from\_pretrained(model\_name)

**3. Run Simple Inference (Testing the Model)**

You can start by feeding the model some basic prompts to see how it generates responses. This is a good way to verify that everything is working properly.

prompt = "What are the key trends in the tech industry?"

inputs = tokenizer(prompt, return\_tensors="pt")

outputs = model.generate(inputs.input\_ids, max\_length=50, do\_sample=True)

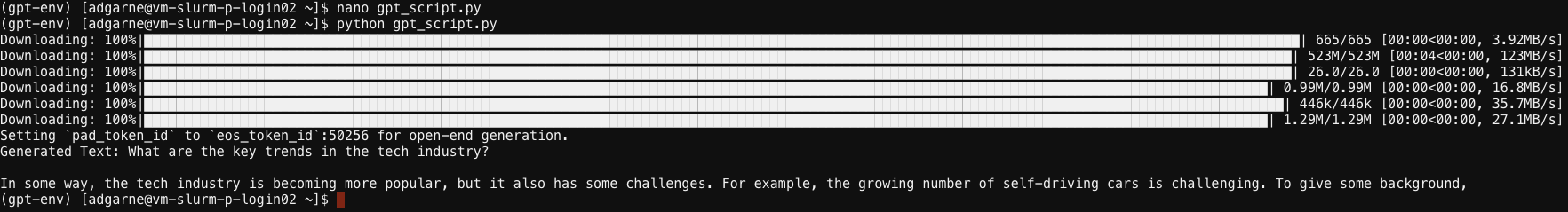
generated\_text = tokenizer.decode(outputs[0], skip\_special\_tokens=True)

print(generated\_text)

A computer screen with text

Description automatically generated

Running this will give you some initial output. You may want to explore adjusting parameters like max\_length and do\_sample to control response length and variability.



**4. Prepare for Fine-Tuning (Optional)**

If you need the model to specialize in a particular area, like analyzing financial data or generating business insights, you can fine-tune it.

• **Create/Obtain a Dataset**: Gather text documents or labeled data (e.g., business reports, transcripts, etc.) that reflect what you want the model to learn.

• **Tokenize Your Dataset**:

from datasets import load\_dataset

*# Load a dataset (you can also create your own custom dataset)*

data = load\_dataset("your\_dataset\_name")

def tokenize\_function(example):

return tokenizer(example['text'], truncation=True)

tokenized\_data = data.map(tokenize\_function, batched=True)

• **Set Up Fine-Tuning**: Hugging Face’s Trainer class can help with this.

from transformers import Trainer, TrainingArguments

training\_args = TrainingArguments(

output\_dir="./results",

evaluation\_strategy="epoch",

per\_device\_train\_batch\_size=4,

num\_train\_epochs=3,

save\_steps=10\_000,

save\_total\_limit=2,

)

trainer = Trainer(

model=model,

args=training\_args,

train\_dataset=tokenized\_data["train"],

eval\_dataset=tokenized\_data["validation"],

)

*# Train the model*

trainer.train()

**5. Optimize for the Supercomputer (If Necessary)**

• **Distributed Training**: If your model and dataset are large, look into distributed training. Hugging Face provides tools to help, like deepspeed and accelerate, which make it easier to leverage multiple GPUs on supercomputer clusters.

pip install deepspeed accelerate

Then, modify the Trainer settings to leverage deepspeed.

• **Batch Sizes and Checkpoints**: Supercomputer clusters have more memory, so you can increase batch sizes for faster training. Be sure to save model checkpoints frequently to avoid losing progress.

**6. Deploy and Access Your Model**

Once trained, you can set up a service to use it:

• **Use it Directly**: Run inference jobs on the cluster with specific prompts as needed.

• **Deploy as an API**: Use tools like FastAPI or Flask to deploy the model on a server accessible to other applications.

**Summary**

In summary:

1. Set up your environment and install necessary packages.

2. Load a pre-trained GPT model for testing.

3. Optionally, fine-tune with a dataset for specialized tasks.

4. Optimize for supercomputer resources if needed, using distributed training or larger batch sizes.

5. Deploy or run inference jobs directly on the cluster.

**Here’s how you can use nano to set up and run Python code on the cluster.**

**1. Open nano to Create a Python File**

Start by creating a new Python file using nano:

nano gpt\_script.py

**2. Write the Python Code in nano**

In nano, type (or paste) the following code. This script loads a GPT model (like GPT-2) and runs a test prompt to generate text.

from transformers import AutoModelForCausalLM, AutoTokenizer

*# Load a model and tokenizer from Hugging Face*

model\_name = "gpt2" *# You could also try "gpt-neo-2.7B" or others*

model = AutoModelForCausalLM.from\_pretrained(model\_name)

tokenizer = AutoTokenizer.from\_pretrained(model\_name)

*# Generate text based on a prompt*

prompt = "What are the key trends in the tech industry?"

inputs = tokenizer(prompt, return\_tensors="pt")

outputs = model.generate(inputs.input\_ids, max\_length=50, do\_sample=True)

generated\_text = tokenizer.decode(outputs[0], skip\_special\_tokens=True)

print("Generated Text:", generated\_text)

3. **Save and Exit** nano

• Press CTRL + O to save the file. You’ll see a prompt at the bottom; just press **Enter** to confirm.

• Press CTRL + X to exit nano.

**3. Run the Python Script**

Now that your script is saved, you can execute it with Python.

python gpt\_script.py

This will run the script, which loads the model, processes the prompt, and generates text. You should see the output printed directly in your terminal.

**Additional Tips**

• **Edit the Script Again**: If you want to make changes, simply reopen the file in nano:

nano gpt\_script.py

• **Install Additional Packages**: If you haven’t installed the transformers library yet, install it with:

pip install transformers

**How to do everything at once**

I’ll guide you through adding steps 3 and 4 to your Python code in nano for testing, tokenizing data, and fine-tuning the model.

**1. Reopen gpt\_script.py in nano**

If you’ve already created gpt\_script.py with the initial code, open it again:

nano gpt\_script.py

**2. Add Tokenization and Fine-Tuning Code in nano**

Here’s the full code, including steps 2, 3, and 4, so you can add everything at once. I’ll explain each section after.

from transformers import AutoModelForCausalLM, AutoTokenizer, Trainer, TrainingArguments

import torch

from datasets import load\_dataset

*# Load a model and tokenizer from Hugging Face*

model\_name = "gpt2" *# You could also try "gpt-neo-2.7B" or others*

model = AutoModelForCausalLM.from\_pretrained(model\_name)

tokenizer = AutoTokenizer.from\_pretrained(model\_name)

*# Step 3: Test the model with a prompt*

prompt = "What are the key trends in the tech industry?"

inputs = tokenizer(prompt, return\_tensors="pt")

outputs = model.generate(inputs.input\_ids, max\_length=50, do\_sample=True)

generated\_text = tokenizer.decode(outputs[0], skip\_special\_tokens=True)

print("Generated Text:", generated\_text)

*# Step 4: Prepare Dataset for Fine-Tuning*

*# Loading a dataset. Replace 'your\_dataset\_name' with your actual dataset*

data = load\_dataset("your\_dataset\_name") *# For example, "wikitext" or your own data*

*# Tokenize the dataset*

def tokenize\_function(example):

return tokenizer(example['text'], truncation=True, padding="max\_length", max\_length=128)

tokenized\_data = data.map(tokenize\_function, batched=True)

*# Fine-tuning setup with Trainer*

training\_args = TrainingArguments(

output\_dir="./results",

evaluation\_strategy="epoch",

learning\_rate=2e-5,

per\_device\_train\_batch\_size=2,

num\_train\_epochs=3,

weight\_decay=0.01,

)

trainer = Trainer(

model=model,

args=training\_args,

train\_dataset=tokenized\_data['train'],

eval\_dataset=tokenized\_data['validation']

)

*# Train the model*

trainer.train()

**3. Save and Exit nano**

• Press CTRL + O to save the file and **Enter** to confirm.

• Press CTRL + X to exit nano.

**4. Run the Script**

Now, execute the script to test the model and start fine-tuning it.

python gpt\_script.py

**Explanation of Each Step:**

• **Testing the Model**: The code generates text based on a sample prompt, which helps verify that the model and tokenizer are working correctly.

• **Dataset Preparation**: Loads a dataset, tokenizes it, and prepares it for fine-tuning. Replace "your\_dataset\_name" with an actual dataset, such as "wikitext", or use a custom dataset if you have one.

• **Fine-Tuning**: The Trainer is set up to fine-tune the model, specifying batch size, learning rate, and epochs.

Once the fine-tuning is complete, you should see the model save results to ./results. This structure will allow you to test, tokenize, and fine-tune all in one go.