

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
```

```
[adgarne@vm-slurm-p-login02 ~]$ python3 -m venv gpt-env
[adgarne@vm-slurm-p-login02 ~]$ source gpt-env/bin/activate
(gpt-env) [adgarne@vm-slurm-p-login02 ~]$ pip install torch transformers datasets
Collecting torch
  Downloading https://files.pythonhosted.org/packages/a4/54/81b1c3c574a1ffde54b0c82ed2a37d81395709cdd5f50e59970aedd5d95e/torch-1.10.2-cp36m-manylinux1_x86_64.whl (881.9MB)
    100% |#####| 881.9MB 2.2kB/s
Collecting transformers
  Downloading https://files.pythonhosted.org/packages/8f/e9/c2b4c823b3959d475a570c1bd2df4125478e2e37b96fb967a87933ae7134/transformers-4.18.0-py3-none-any.whl (4.0MB)
    100% |#####| 4.0MB 306kB/s
Collecting datasets
  Downloading https://files.pythonhosted.org/packages/13/68/8f123cf1b84fc32d749357b2c7ed6e9e61c06246965ba7f6f7a78cba54e9/datasets-2.4.0-py3-none-any.whl (365kB)
    100% |#####| 368kB 504kB/s
Collecting dataclasses; python_version < "3.7" (from torch)
  Downloading https://files.pythonhosted.org/packages/fe/ca/75fac5856ab5cfa51bbbcfa250182e50441074f3dc3f803f6e76451fab43/dataclasses-0.8-py3-none-any.whl
Collecting typing-extensions (from torch)
  Downloading https://files.pythonhosted.org/packages/45/6b/44f7f8f1e110027cf88956b59f2fad776cca7e1704396d043f89effd3a0e/typing_extensions-4.1.1-py3-none-any.whl
Collecting tokenizers!=0.11.3,<0.13,>=0.11.1 (from transformers)
  Downloading https://files.pythonhosted.org/packages/12/57/da0cb8e40437f88630769164a66afec8af294ff686661497b6c88bc08556/tokenizers-0.12.1.tar.gz (220kB)
```

- **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
```

```
Command "python setup.py egg_info" failed with error code 1 in /tmp/pip-build-uoi5wg6j/tokenizers/
You are using pip version 9.0.3, however version 24.3.1 is available.
You should consider upgrading via the 'pip install --upgrade pip' command.
(gpt-env) [adgarne@vm-slurm-p-login02 ~]$ pip install --upgrade pip
Collecting pip
  Downloading https://files.pythonhosted.org/packages/a4/6d/6463d49a933f547439d6b5b98b46af8742cc03ae83543e4d7688c2420f8b/pip-21.3.1-py3-none-any.whl (1.7MB)
    100% |#####| 1.7MB 491kB/s
Installing collected packages: pip
  Found existing installation: pip 9.0.3
  Uninstalling pip-9.0.3:
    Successfully uninstalled pip-9.0.3
  Successfully installed pip-21.3.1
You are using pip version 21.3.1, however version 24.3.1 is available.
You should consider upgrading via the 'pip install --upgrade pip' command.
(gpt-env) [adgarne@vm-slurm-p-login02 ~]$ pip install --upgrade pip
Requirement already satisfied: pip in ./gpt-env/lib/python3.6/site-packages (21.3.1)
(gpt-env) [adgarne@vm-slurm-p-login02 ~]$ pip install torch transformers datasets
Collecting torch
```

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.

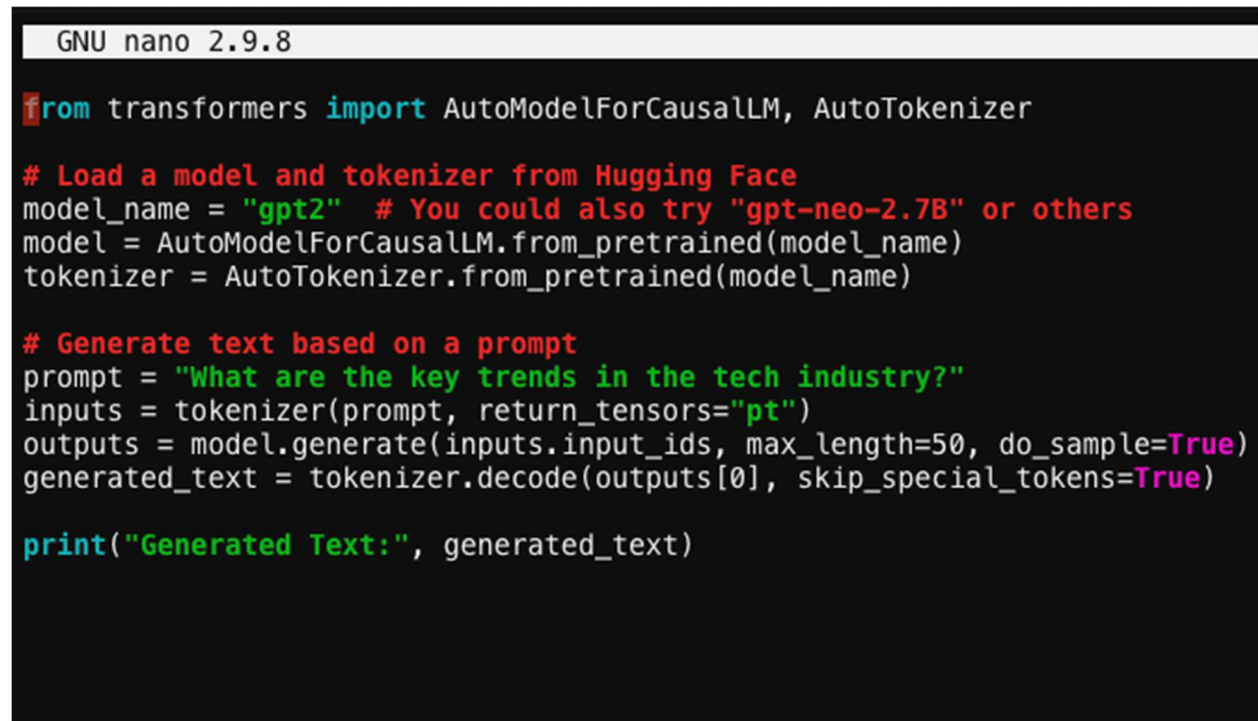
```
from transformers import AutoModelForCausalLM, AutoTokenizer
```

```
# 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)
```



```
GNU nano 2.9.8

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)
```

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.

```

(gpt-env) [adgarne@vm-slurm-p-login02 ~]$ nano gpt_script.py
(gpt-env) [adgarne@vm-slurm-p-login02 ~]$ python gpt_script.py
Downloading: 100% |
Downloading: 100% |
Downloading: 100% |
Downloading: 100% |
Downloading: 100% |
Downloading: 100% |
Setting `pad_token_id` to `eos_token_id`:50256 for open-end generation.
Generated Text: What are the key trends in the tech industry?

In some way, the tech industry is becoming more popular, but it also has some challenges. For example, the growing number of self-d
(gpt-env) [adgarne@vm-slurm-p-login02 ~]$

```

#### 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.