LLM Inference on HPC

Turing HPC training 2025

25-26 November – Rosie Wood

Plan for this session

1. Login and get set up on Baskerville
2. Run LLM inference on Baskerville using a batch job (sbatch)
3. Run LLM inference on Baskerville using an interactive job (srun)
4. Understand how this process is different on Isambard-AI

Where can I find the code?

- https://github.com/alan-turing-institute/hpc-training-nov-2025.git

- We will be running

` 2-Inference/transformers_examples` which use 1 node with 4 GPUs.

- You can also have a go at the `vllm_examples` at a later date, these use 2 nodes with 4 GPUs per node (please don’t run them now or we will run out of available GPUs).

Set up on Baskerville

1. Login to Baskerville
   ```
   ssh uname@login.baskerville.ac.uk
   ```
2. Change directory to ` /bask/projects/v/vjgo8416-hpc2511`
   ```
   cd ~/vjgo8416-hpc2511
   ```
3. Change to your directory
   ```
   cd $USER
   ```
4. Change directory to
   ```
   cd hpc-training-nov-2025/2-Inference/transformers_examples
   ```

Using batch jobs on Baskerville

View the `qwen.py` script

1. View the file in your terminal using `less`:

   ```
   less qwen.py
   ```

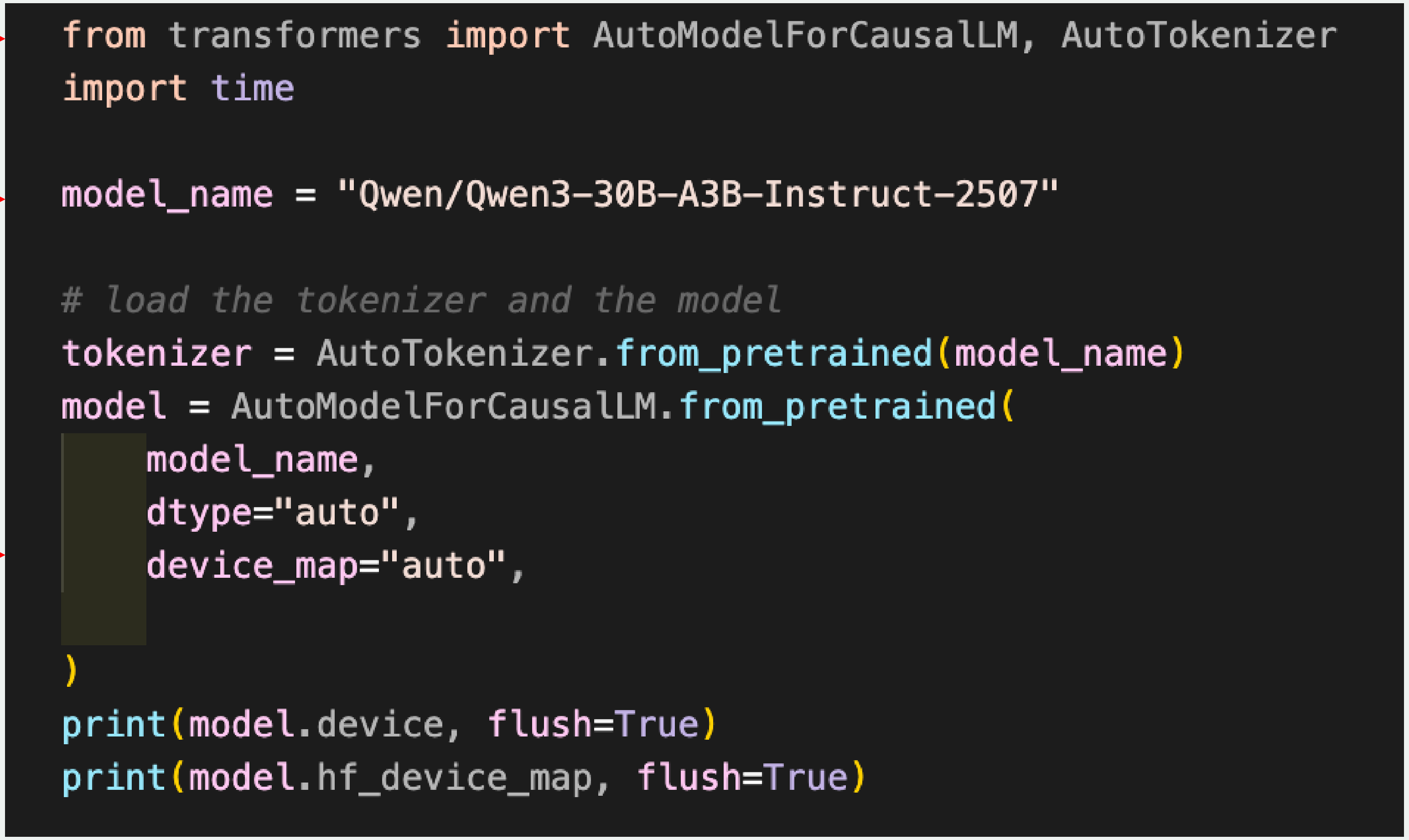
Or go here to view on GitHub: https://github.com/rwood-97/multi_node_inference/blob/main/transformers_examples/qwen.py

qwen.py

auto detect which model and tokenizer to use

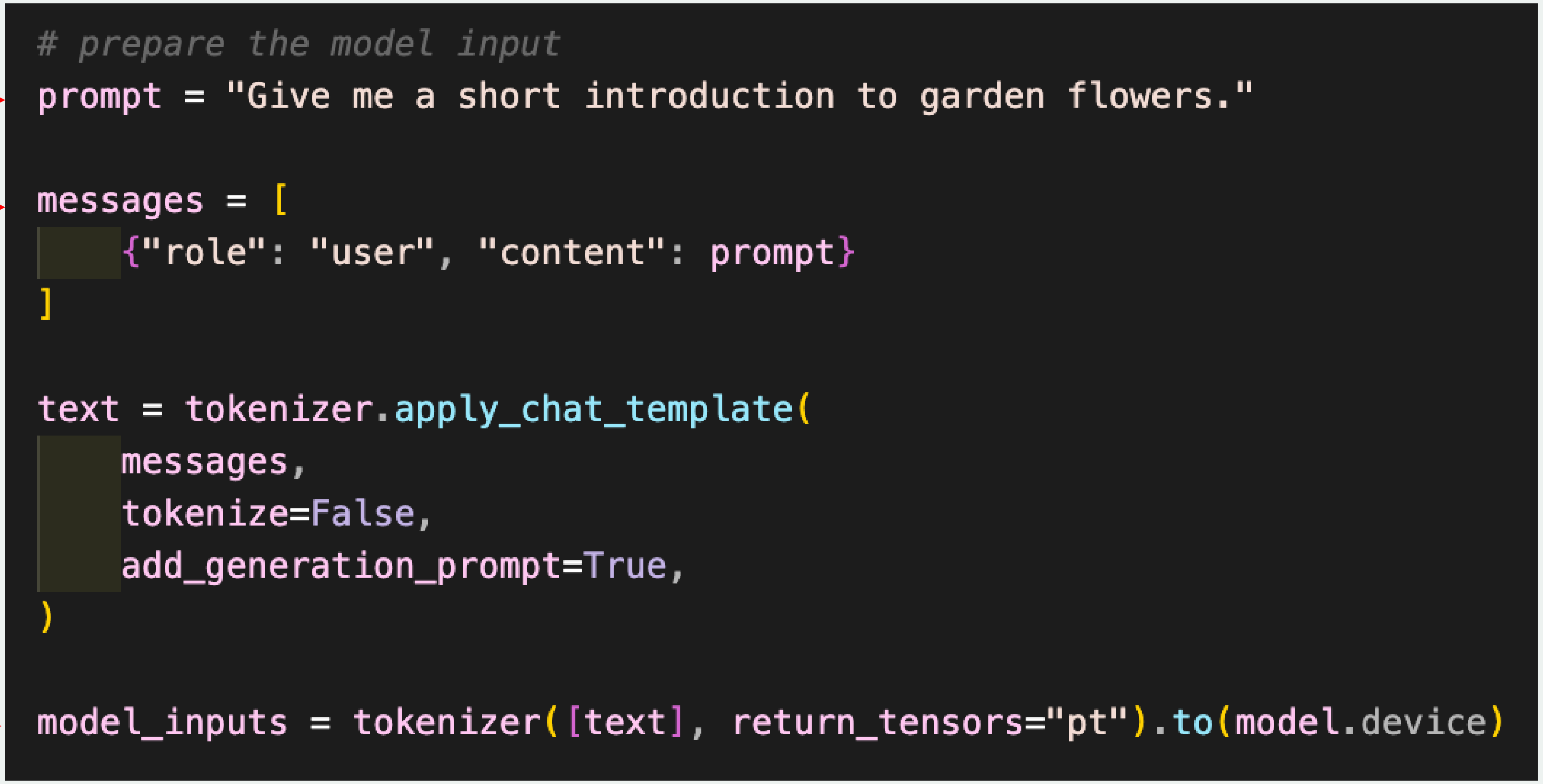
we choose Qwen3-30B

auto pick which device(s)



```python
from transformers import AutoModelForCausalLM, AutoTokenizer
import time

model_name = "Qwen/Qwen3-30B-A3B-Instruct-2507"

# load the tokenizer and the model
tokenizer = AutoTokenizer.from_pretrained(model_name)
model = AutoModelForCausalLM.from_pretrained(
    model_name,
    dtype="auto",
    device_map="auto",

)
print(model.device, flush=True)
print(model.hf_device_map, flush=True)
```

qwen.py



```python
# prepare the model input
prompt = "Give me a short introduction to garden flowers."

messages = [
    {"role": "user", "content": prompt}
]

text = tokenizer.apply_chat_template(
    messages,
    tokenize=False,
    add_generation_prompt=True,
)

model_inputs = tokenizer([text], return_tensors="pt").to(model.device)
```

define your prompt

define messages

convert to input tensor and move to GPU(s)

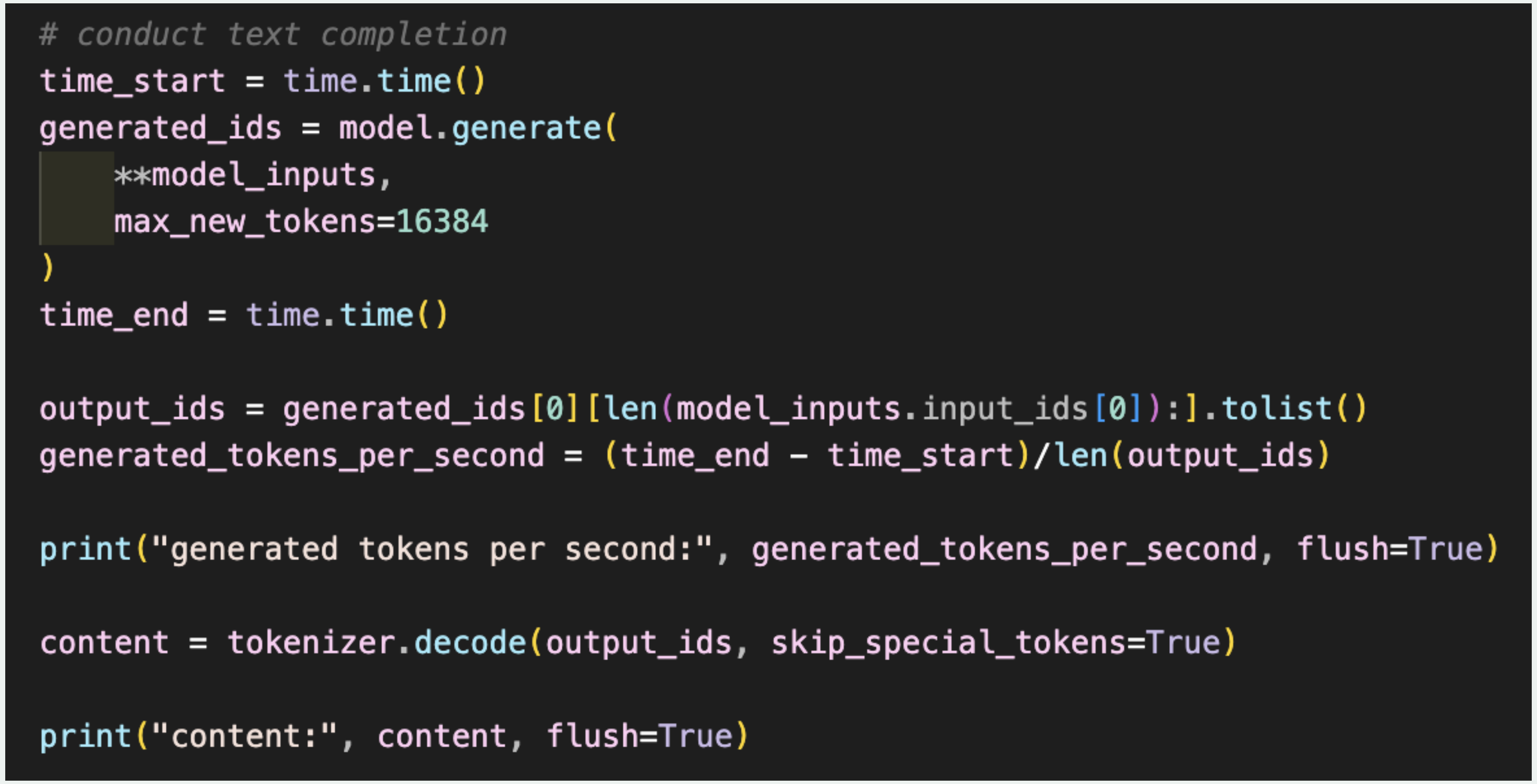
qwen.py

call the model

get the response (as tokens)

convert to text

print response



```
# conduct text completion
time_start = time.time()
generated_ids = model.generate(
    **model_inputs,
    max_new_tokens=16384
)
time_end = time.time()

output_ids = generated_ids[0][len(model_inputs.input_ids[0]):].tolist()
generated_tokens_per_second = (time_end - time_start)/len(output_ids)

print("generated tokens per second:", generated_tokens_per_second, flush=True)

content = tokenizer.decode(output_ids, skip_special_tokens=True)

print("content:", content, flush=True)
```

View the batch script

1. Change directory to the ` baskerville ` directory
   ```
   cd baskerville
   ```
2. View the file in your terminal using ` less ` :
   ```
   less batch_transformers_run_1node.sh
   ```

Or go here to view on GitHub: https://github.com/rwood-97/multi_node_inference/blob/main/transformers_examples/baskerville/batch_transformers_run_1node.sh

Batch script

use the Turing's allocation

which project to charge time to

estimated time in HH:MM:SS

1 node

4 gpus

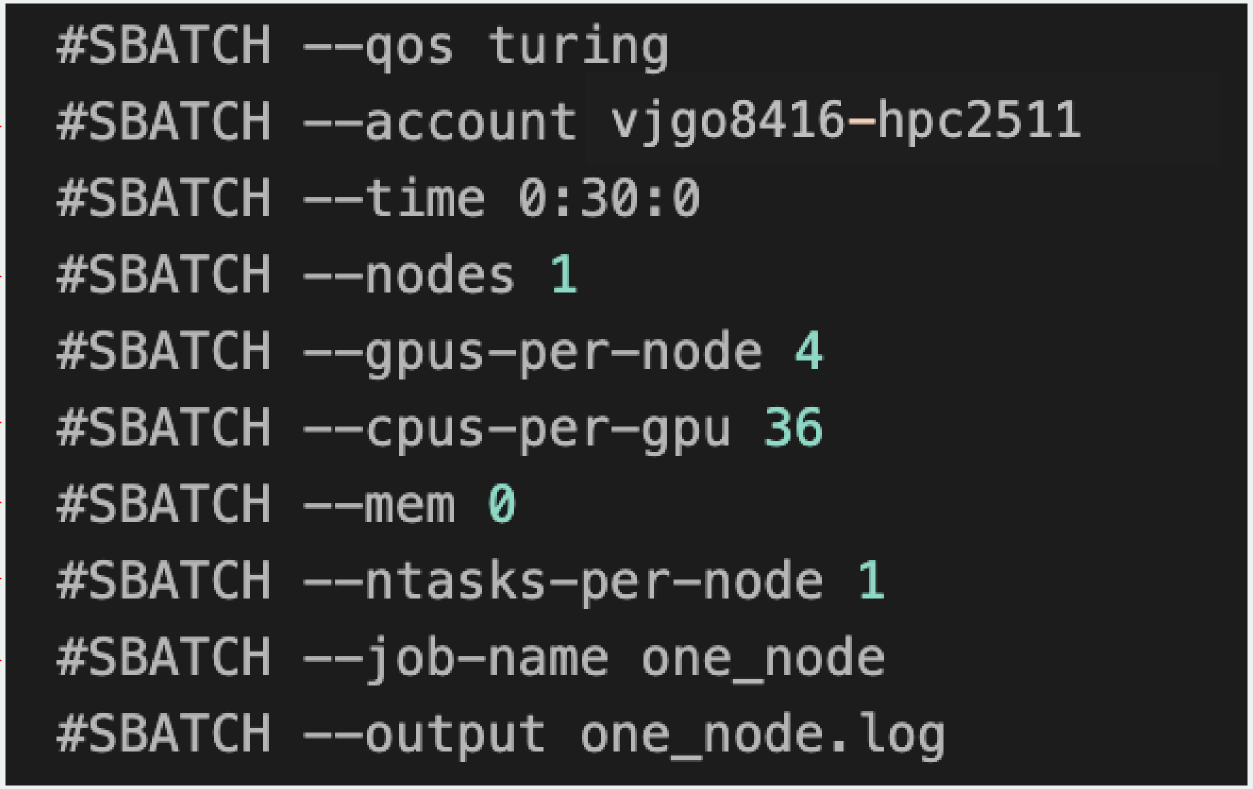
36 cpus

"give me all the memory for this node"

1 task (python)

the job name

the name for the log file



```
#SBATCH --qos turing
#SBATCH --account vjgo8416-hpc2511
#SBATCH --time 0:30:0
#SBATCH --nodes 1
#SBATCH --gpus-per-node 4
#SBATCH --cpus-per-gpu 36
#SBATCH --mem 0
#SBATCH --ntasks-per-node 1
#SBATCH --job-name one_node
#SBATCH --output one_node.log
```

Batch script

load the required modules

set the pip cache directory

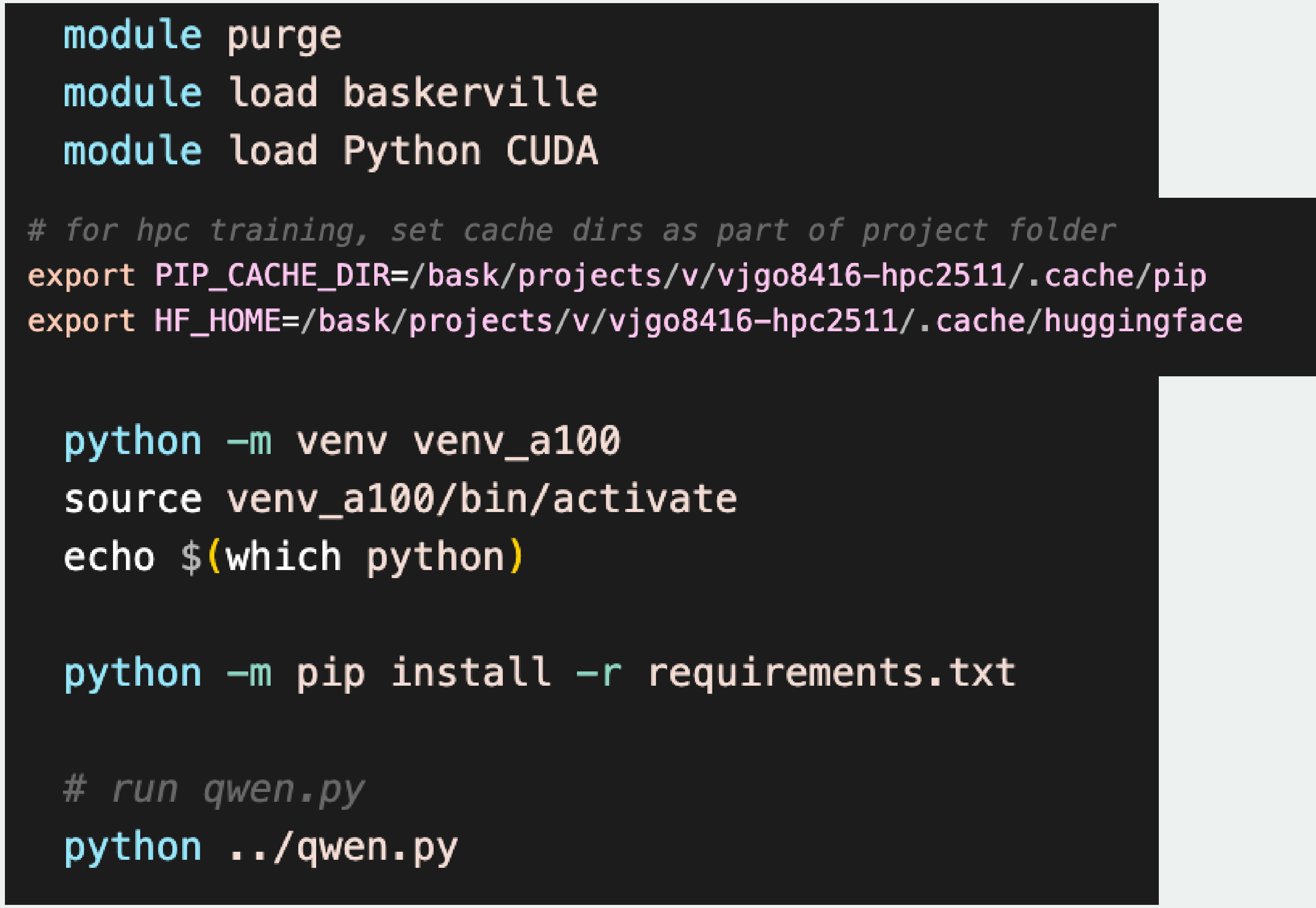
set the huggingface cache directory

create a venv

activate your venv

install python requirements into venv

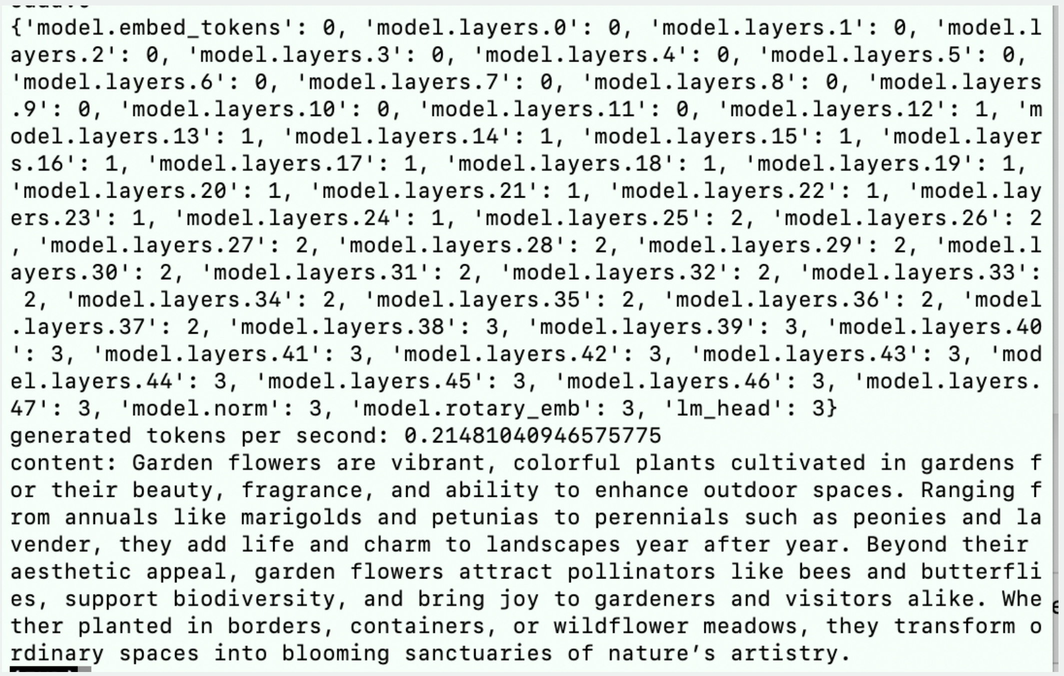
run qwen.py



```
  module purge
  module load baskerville
  module load Python CUDA

# for hpc training, set cache dirs as part of project folder
export PIP_CACHE_DIR=/bask/projects/v/vjgo8416-hpc2511/.cache/pip
export HF_HOME=/bask/projects/v/vjgo8416-hpc2511/.cache/huggingface

  python -m venv venv_a100
  source venv_a100/bin/activate
  echo $(which python)

  python -m pip install -r requirements.txt

  # run qwen.py
  python ../qwen.py
```

Run the `qwen.py` script as a batch job

1. Run the batch script using `sbatch`
   ```
   sbatch batch_transformers_run_1node.sh
   ```
2. Check your job status using `squeue`
   ```
   squeue --me
   ```
3. Once run, view the job logs
   ```
   less one_node.log
   less one_node.log.stats
   ```



```
{'model.embed_tokens': 0, 'model.layers.0': 0, 'model.layers.1': 0, 'model.l
ayers.2': 0, 'model.layers.3': 0, 'model.layers.4': 0, 'model.layers.5': 0,
'model.layers.6': 0, 'model.layers.7': 0, 'model.layers.8': 0, 'model.layers
.9': 0, 'model.layers.10': 0, 'model.layers.11': 0, 'model.layers.12': 1, 'm
odel.layers.13': 1, 'model.layers.14': 1, 'model.layers.15': 1, 'model.layer
s.16': 1, 'model.layers.17': 1, 'model.layers.18': 1, 'model.layers.19': 1,
'model.layers.20': 1, 'model.layers.21': 1, 'model.layers.22': 1, 'model.lay
ers.23': 1, 'model.layers.24': 1, 'model.layers.25': 2, 'model.layers.26': 2
, 'model.layers.27': 2, 'model.layers.28': 2, 'model.layers.29': 2, 'model.l
ayers.30': 2, 'model.layers.31': 2, 'model.layers.32': 2, 'model.layers.33':
 2, 'model.layers.34': 2, 'model.layers.35': 2, 'model.layers.36': 2, 'model
.layers.37': 2, 'model.layers.38': 3, 'model.layers.39': 3, 'model.layers.40
': 3, 'model.layers.41': 3, 'model.layers.42': 3, 'model.layers.43': 3, 'mod
el.layers.44': 3, 'model.layers.45': 3, 'model.layers.46': 3, 'model.layers.
47': 3, 'model.norm': 3, 'model.rotary_emb': 3, 'lm_head': 3}
generated tokens per second: 0.21481040946575775
content: Garden flowers are vibrant, colorful plants cultivated in gardens f
or their beauty, fragrance, and ability to enhance outdoor spaces. Ranging f
rom annuals like marigolds and petunias to perennials such as peonies and la
vender, they add life and charm to landscapes year after year. Beyond their
aesthetic appeal, garden flowers attract pollinators like bees and butterfli
es, support biodiversity, and bring joy to gardeners and visitors alike. Whe
ther planted in borders, containers, or wildflower meadows, they transform o
rdinary spaces into blooming sanctuaries of nature's artistry.
```

Using interactive jobs on Baskerville

View the `qwen_chat.py` script

1. Change directory back to `transformers_examples`:
   ```
   cd ..
   ```
2. View the file in your terminal using `less`:
   ```
   less qwen_chat.py
   ```

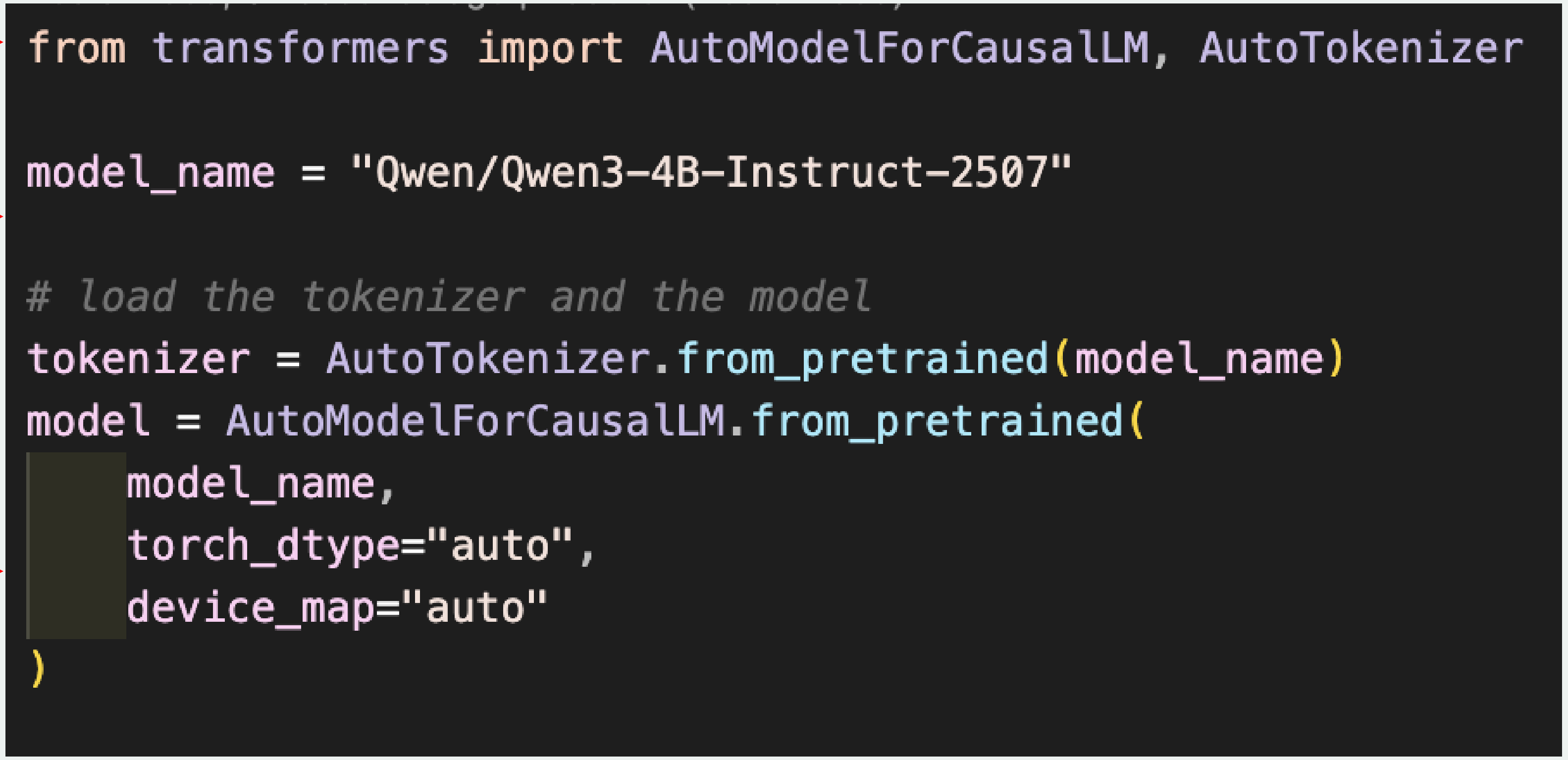
Or go here to view on GitHub: https://github.com/rwood-97/multi_node_inference/blob/main/transformers_examples/qwen_chat.py

qwen_chat.py

auto detect which model and tokenizer to use

we choose Qwen3-4B (smaller model here, since we are just using 1 GPU)

auto pick which device(s)



```python
from transformers import AutoModelForCausalLM, AutoTokenizer

model_name = "Qwen/Qwen3-4B-Instruct-2507"

# load the tokenizer and the model
tokenizer = AutoTokenizer.from_pretrained(model_name)
model = AutoModelForCausalLM.from_pretrained(
    model_name,
    torch_dtype="auto",
    device_map="auto"
)
```

i.e. so far, the same as the qwen.py

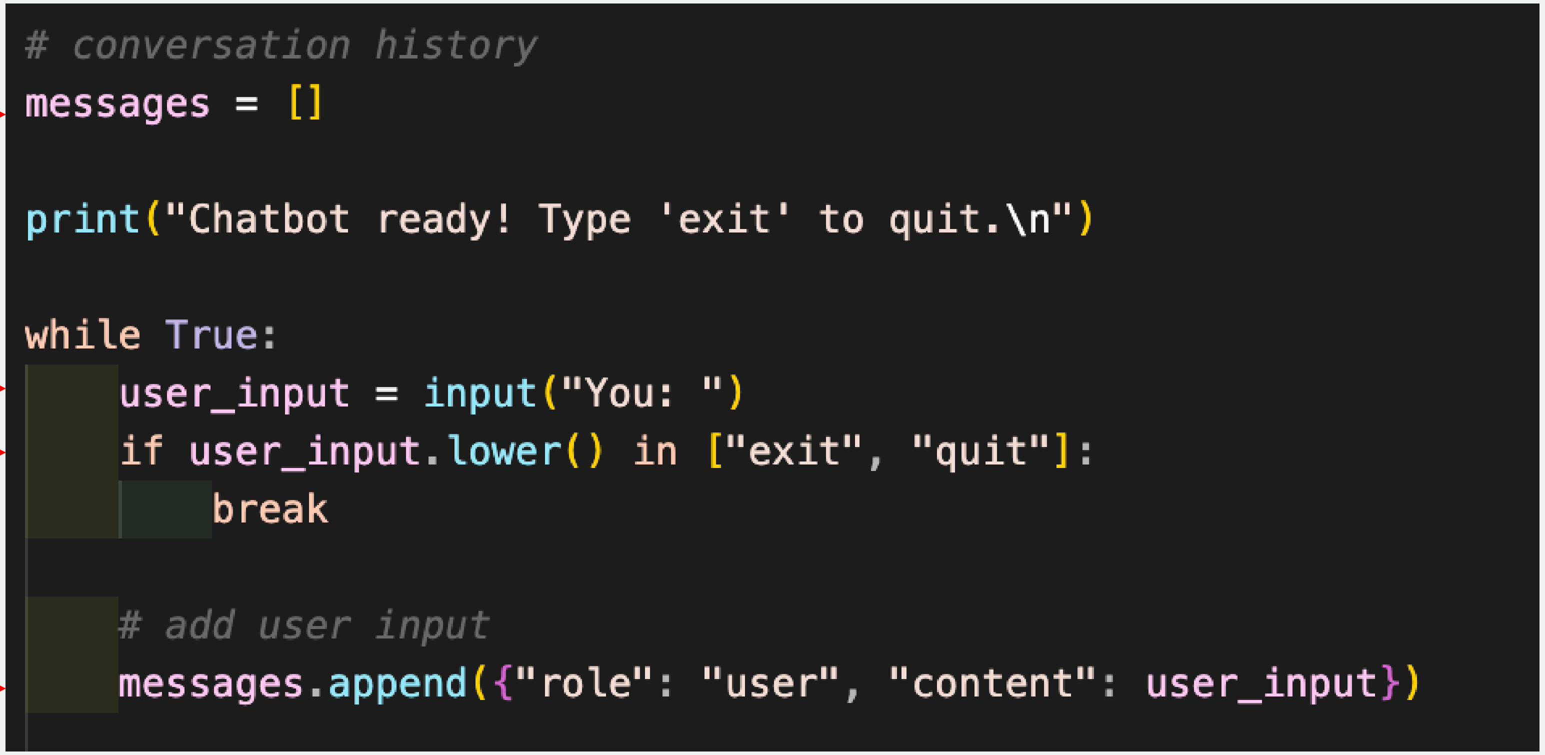
qwen_chat.py

initialize messages as a blank list

get the user to input some text

while loop will run until we type “exit” or “quit”

add input to messages

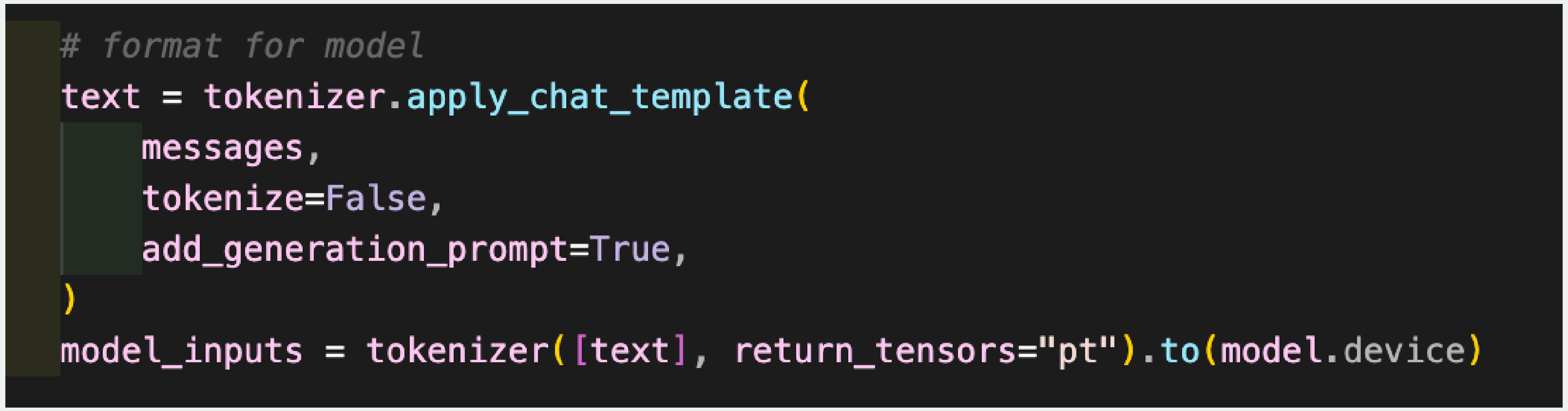


```python
# conversation history
messages = []

print("Chatbot ready! Type 'exit' to quit.\n")

while True:
    user_input = input("You: ")
    if user_input.lower() in ["exit", "quit"]:
        break

    # add user input
    messages.append({"role": "user", "content": user_input})
```

unless you input some text, this will hang, i.e. we need interactive

qwen_chat.py

...still in while loop



```
# format for model
text = tokenizer.apply_chat_template(
    messages,
    tokenize=False,
    add_generation_prompt=True,
)
model_inputs = tokenizer([text], return_tensors="pt").to(model.device)
```

convert to input tensor
and move to GPU

qwen_chat.py

…still in while loop

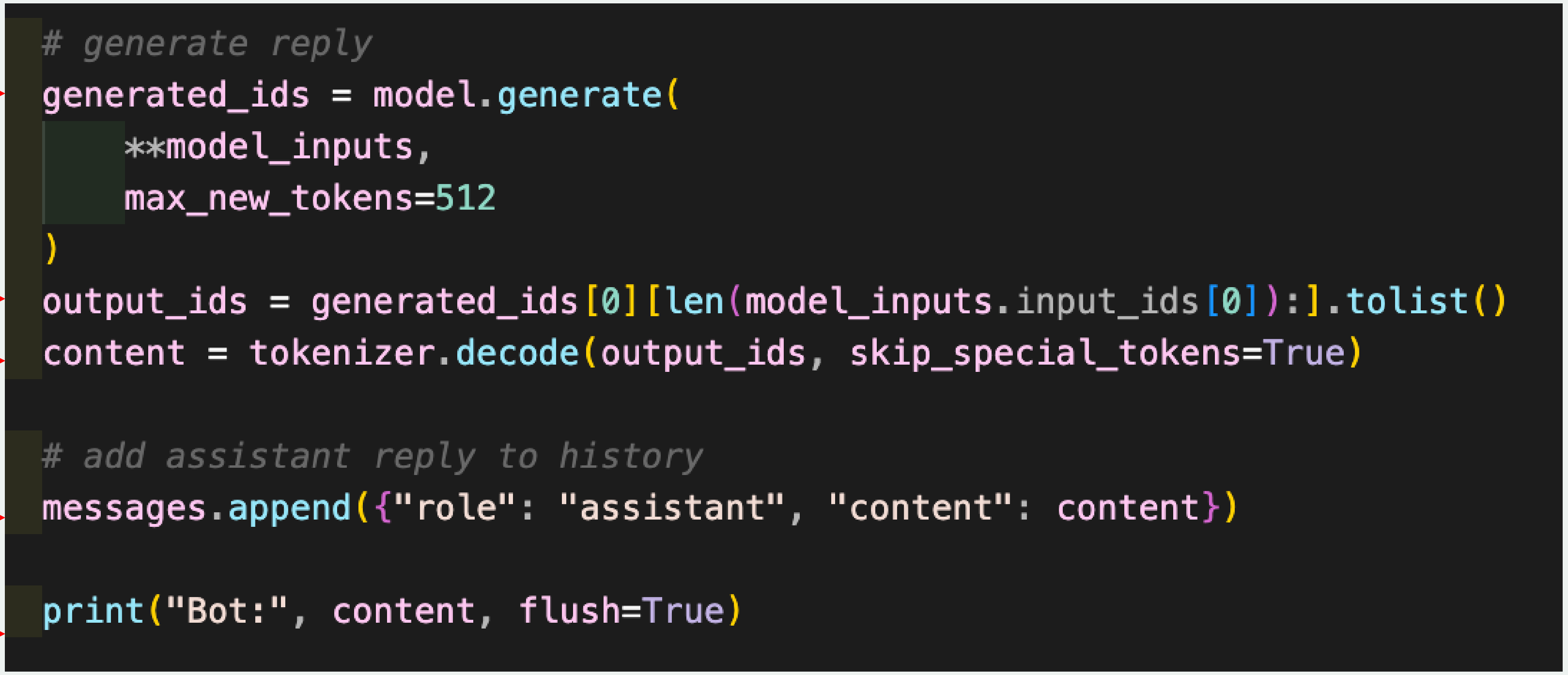
call the model

get the response (as tokens)

convert to text

append the response to our list of message

print response



```python
# generate reply
generated_ids = model.generate(
    **model_inputs,
    max_new_tokens=512
)
output_ids = generated_ids[0][len(model_inputs.input_ids[0]):].tolist()
content = tokenizer.decode(output_ids, skip_special_tokens=True)

# add assistant reply to history
messages.append({"role": "assistant", "content": content})

print("Bot:", content, flush=True)
```

Start an interactive job

1. Change directory to the `baskerville` directory:

```
cd baskerville
```

2. Use `srun` to request an interactive job

```
srun --qos turing --account vjgo8416-hpc2511 --time 
00:10:00 --nodes 1 --gpus-per-node 1 --reservation 
vjgo8416-hpc2511 --pty /bin/bash
```

^ Note this is basically the same as the sbatch commands we used

Run the `qwen_chat.py` script from a compute node

1. Once you are allocated resources, run the following:

```
module purge
module load baskerville
module load Python CUDA

export PIP_CACHE_DIR=/bask/projects/v/vjgo8416-hpc2511/.cache/pip
export HF_HOME=/bask/projects/v/vjgo8416-hpc2511/.cache/huggingface

python -m venv venv_a100
source venv_a100/bin/activate
echo $(which python)

python -m pip install -r requirements.txt

python ../qwen_chat.py
```

2. Chat with the model

alternatively, just run ./run_chat_local.sh
which runs these for you

Using Isambard-AI

Isambard-AI

- ✅ Uses slurm (i.e. `sbatch` and `srun` commands)
- 🆚 Preference for using containers instead of modules
- Isambard-AI uses aarch64 CPUs vs the more common amd64 CPUs which means binaries aren't compatible, sometimes you get weird errors.

The main difference is you need to set up your container!

Set up on Isambard-AI

1. To login to Isambard-AI
   ```
   ssh proj.aip2.isambard
   ```
2. Change directory to your project directory
   ```
   cd $PROJECTDIR
   ```
3. Clone the training repo
   ```
   git clone --recursive https://github.com/alan-turing-
   institute/hpc-training-nov-2025.git
   ```
4. Change directory to `transformers_examples/isambard_ai`
   ```
   cd hpc-training-nov-2025/2-
   Inference/transformers_examples/isambard_ai
   ```

Building a container using apptainer/singularity

View the container definition file

1. View the file in your terminal using `less`:

```
less container/container.def
```

Or go here to view on GitHub: https://github.com/rwood-97/multi_node_inference/blob/main/transformers_examples/isambard_ai/container/container.def

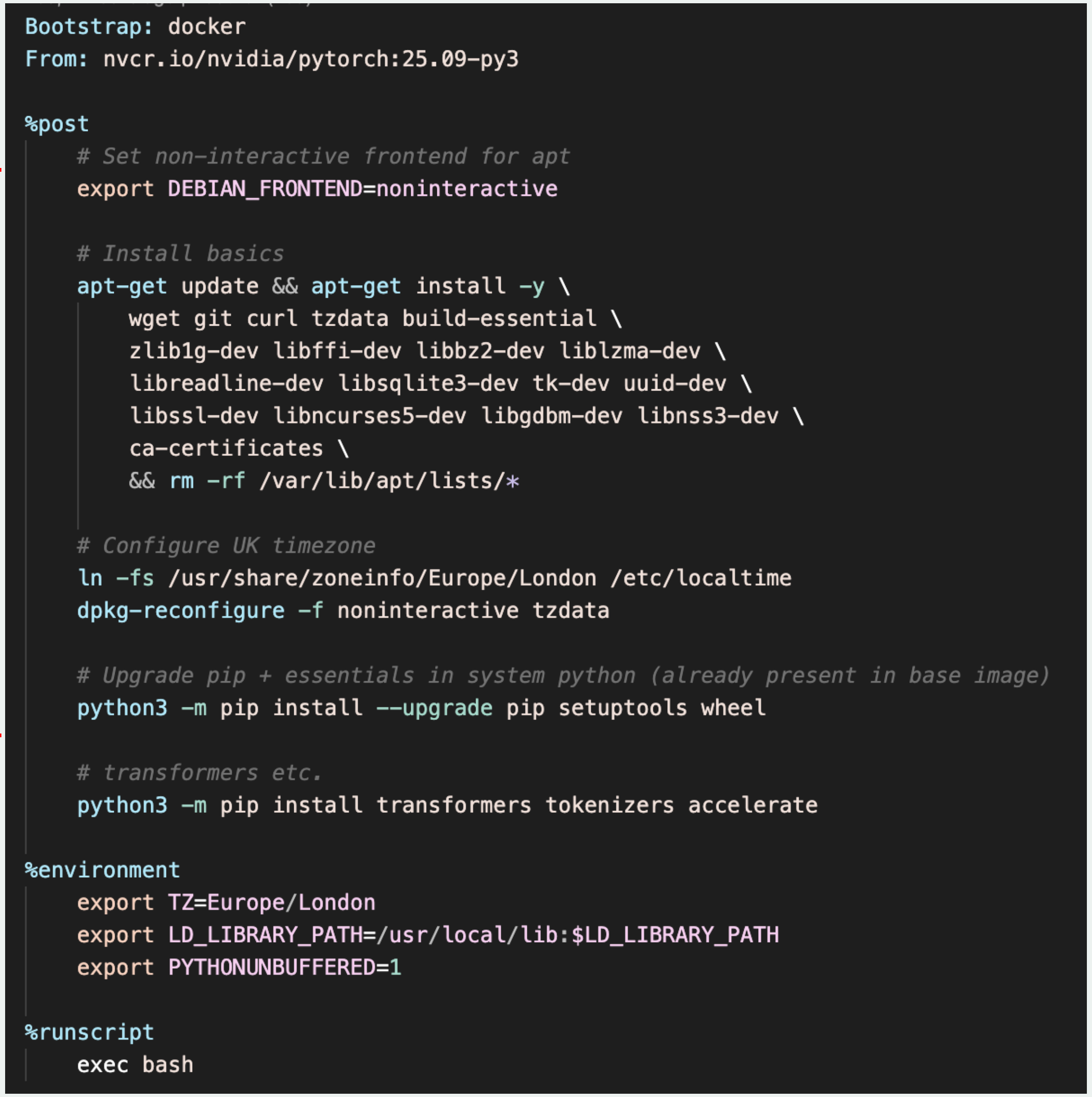
Container.def

which image to use as base image, i.e. https://catalog.ngc.nvidia.com/orgs/nvidia/containers/pytorch?version=25.09-py3

updates

install additional dependencies

run bash when we start the image



```
Bootstrap: docker
From: nvcr.io/nvidia/pytorch:25.09-py3

%post
    # Set non-interactive frontend for apt
    export DEBIAN_FRONTEND=noninteractive

    # Install basics
    apt-get update && apt-get install -y \
        wget git curl tzdata build-essential \
        zlib1g-dev libffi-dev libbz2-dev liblzma-dev \
        libreadline-dev libsqlite3-dev tk-dev uuid-dev \
        libssl-dev libncurses5-dev libgdbm-dev libnss3-dev \
        ca-certificates \
        && rm -rf /var/lib/apt/lists/*

    # Configure UK timezone
    ln -fs /usr/share/zoneinfo/Europe/London /etc/localtime
    dpkg-reconfigure -f noninteractive tzdata

    # Upgrade pip + essentials in system python (already present in base image)
    python3 -m pip install --upgrade pip setuptools wheel

    # transformers etc.
    python3 -m pip install transformers tokenizers accelerate

%environment
    export TZ=Europe/London
    export LD_LIBRARY_PATH=/usr/local/lib:$LD_LIBRARY_PATH
    export PYTHONUNBUFFERED=1

%runscript
    exec bash
```

Building a container

1. Use `srun` to request an interactive job (1 GPU)
```
srun --time 00:10:00 --nodes 1 --gpus-per-node 1 --pty /bin/bash
```
2. Change directory to the container directory
```
cd container
```
3. Build the container with apptainer
```
apptainer build container.sif container.def
```
4. Once this is done, exit the interactive job by typing `exit`

Using batch jobs on Isambard-AI

View the batch file

1. View the file in your terminal using `less`:
   ```
   less batch_transformers_run_gh_1node.sh
   ```

Or go here to view on GitHub: https://github.com/rwood-97/multi_node_inference/blob/main/transformers_examples/isambard_ai/batch_transformers_run_gh_1node.sh

Batch file

estimated time in HH:MM:SS

1 node

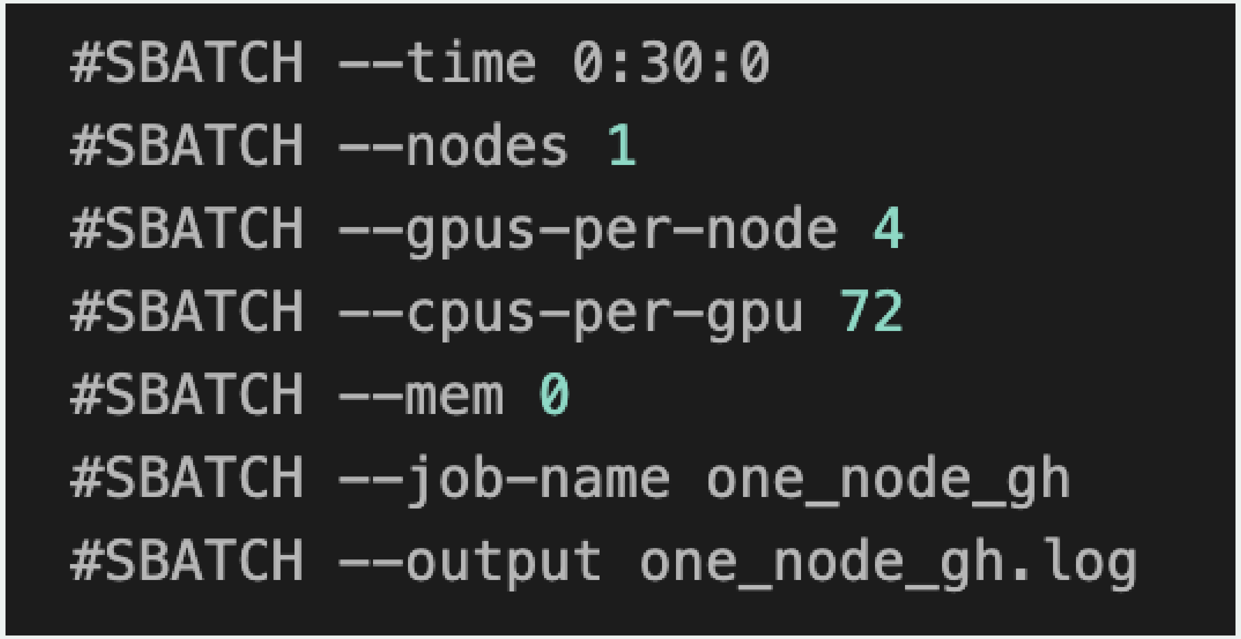
4 gpus

72 cpus

”give me all the memory for this node”

the job name

the name for the log file

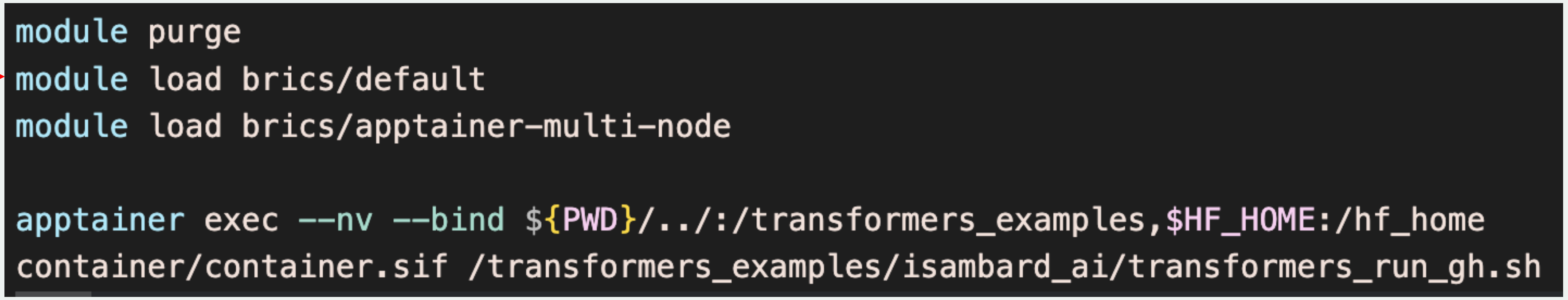


```
#SBATCH --time 0:30:0
#SBATCH --nodes 1
#SBATCH --gpus-per-node 4
#SBATCH --cpus-per-gpu 72
#SBATCH --mem 0
#SBATCH --job-name one_node_gh
#SBATCH --output one_node_gh.log
```

i.e. similar to Baskerville

Batch file

load the required modules →



```
module purge
module load brics/default
module load brics/apptainer-multi-node

apptainer exec --nv --bind ${PWD}/../:/transformers_examples,$HF_HOME:/hf_home
container/container.sif /transformers_examples/isambard_ai/transformers_run_gh.sh
```

This command:

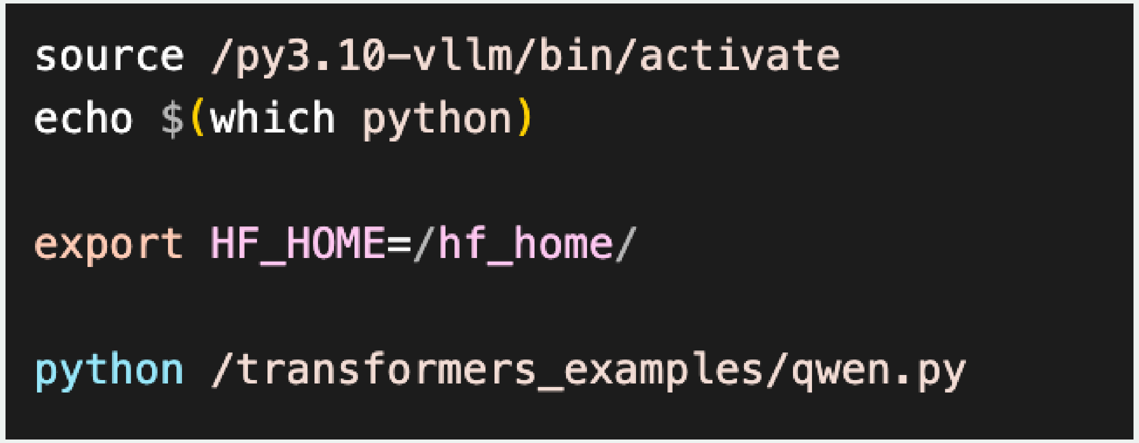
- `apptainer exec` executes a command on a container
- `--nv` says we want to use (NVIDIA) GPUs
- `--bind X:Y` mounts directory X to location Y on the container, i.e. we mount the transformers_examples directory and huggingface cache directory
- `container/container.sif` is our container image
- `/transformers_examples/isambard_ai/transformers_run_gh.sh` is the script we want to run (note: this uses the path on the container)

The transformers_run_gh.sh script

source the python environment (already with everything pre-installed)

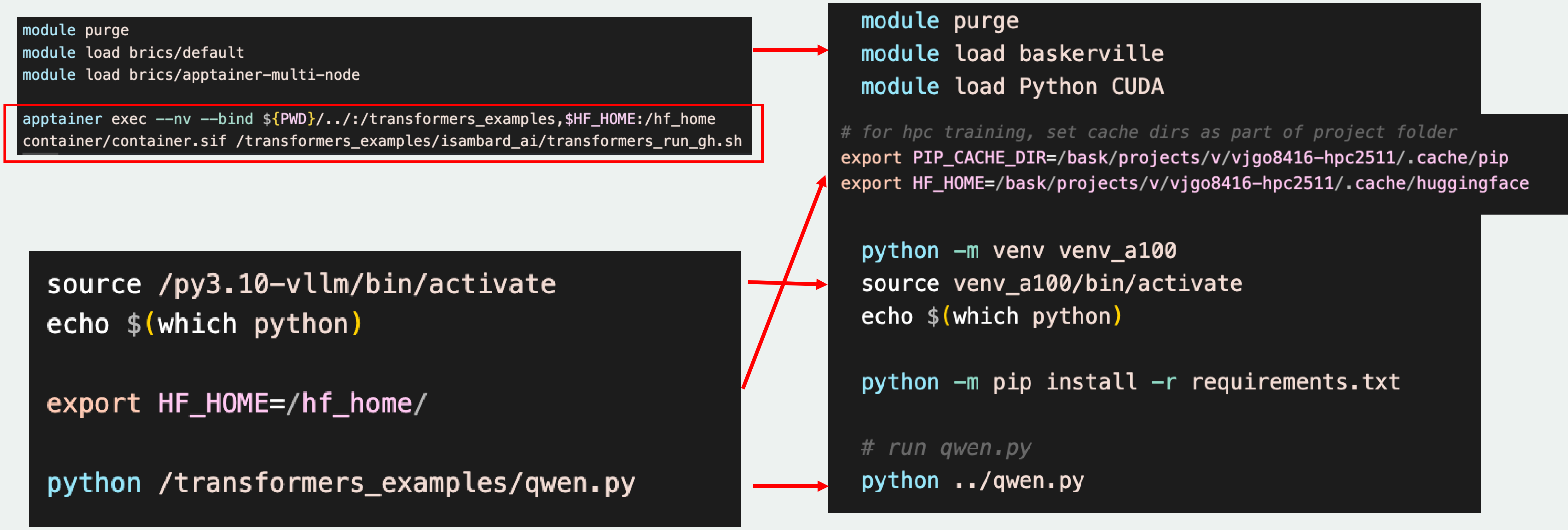
set the huggingface cache directory

run qwen.py



```
source /py3.10-vllm/bin/activate
echo $(which python)

export HF_HOME=/hf_home/

python /transformers_examples/qwen.py
```

Baskerville comparison (right)



```
module purge
module load brics/default
module load brics/apptainer-multi-node

apptainer exec --nv --bind ${PWD}/../:/transformers_examples,$HF_HOME:/hf_home
container/container.sif /transformers_examples/isambard_ai/transformers_run_gh.sh
```

```
source /py3.10-vllm/bin/activate
echo $(which python)

export HF_HOME=/hf_home/

python /transformers_examples/qwen.py
```

```
  module purge
  module load baskerville
  module load Python CUDA

# for hpc training, set cache dirs as part of project folder
export PIP_CACHE_DIR=/bask/projects/v/vjgo8416-hpc2511/.cache/pip
export HF_HOME=/bask/projects/v/vjgo8416-hpc2511/.cache/huggingface

  python -m venv venv_a100
  source venv_a100/bin/activate
  echo $(which python)

  python -m pip install -r requirements.txt

  # run qwen.py
  python ../qwen.py
```

Run the `qwen.py` file as a batch job

1. Change directory to the `isambard_ai` directory
   ```
   cd isambard_ai
   ```
2. Run the batch script using `sbatch`
   ```
   sbatch batch_transformers_run_gh_1node.sh
   ```
3. Check your job status using `squeue`
   ```
   squeue --me
   ```
4. Once run, view the job logs
   ```
   less one_node.log
   less one_node.log.stats
   ```

i.e. general process is the same as Baskerville

Using interactive jobs on Isambard-AI

Run the `qwen_chat.py` file using an interactive job

1. Use `srun` to request an interactive job
   ```
   srun –time 00:10:00 --nodes 1 --gpus-per-node 1 --pty
   /bin/bash
   ```
2. Once you are allocated resources, run the `apptainer run` to start the container:
   ```
   apptainer run --nv --bind
   ${PWD}/../:/transformers_examples,$HF_HOME:/hf_home
   container/container.sif
   ```
3. Set your huggingface cache directory
   ```
   export HF_HOME=/hf_home
   ```
4. Run the `qwen_chat.py` script
   ```
   python /transformers_examples/qwen_chat.py
   ```
5. Chat with the model

Summary

Summary

- Batch jobs – Run scripts, queue multiple jobs at the same time, non-interactive, reproducible!
- Interactive jobs – Good for experimenting and debugging, no logs (you will forget what you did)

- Baskerville – Use modules to access different software versions
- Isambard-AI – Use containers to access different software versions + run within container