The Alan Turing Institute

Tutorial 3: Multi-node training with Lightning

HPC Training – 25-26 November 2025

David Llewellyn-Jones

Motivation

GPUs are great for accelerated training and inference, but processing is bounded by:

1. Available memory
2. Speed of computation

How to make AI better?

1. Improved algorithms
2. Larger models
3. More training data

The last two require *more compute*

Scaling up

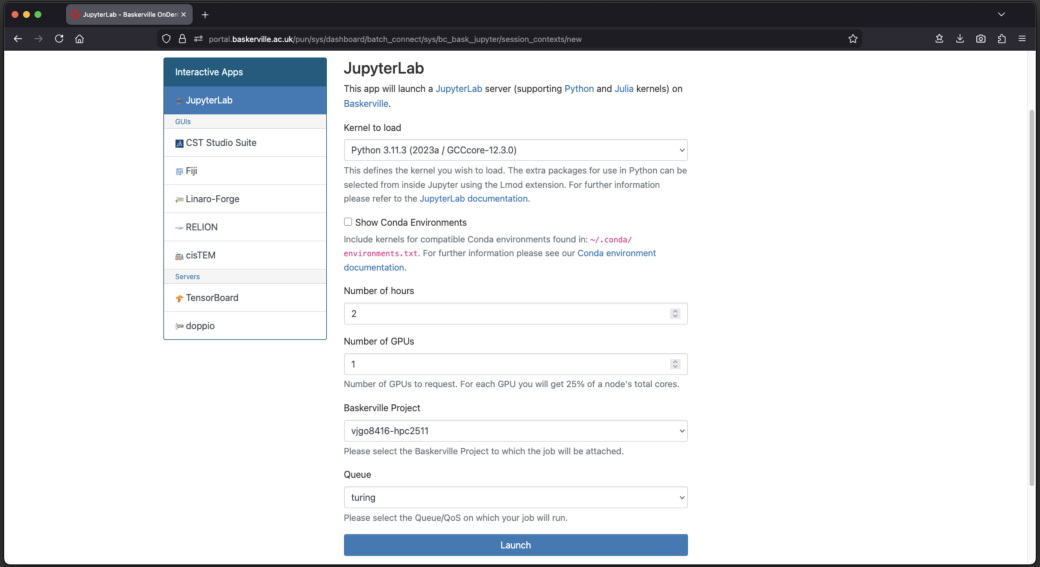
1. Strong motivation for scaling across GPUs
2. Single device: 4 or 8 GPUs maximum
3. Eventually want to scale across nodes
4. Get this right... the sky’s the limit

Preparation

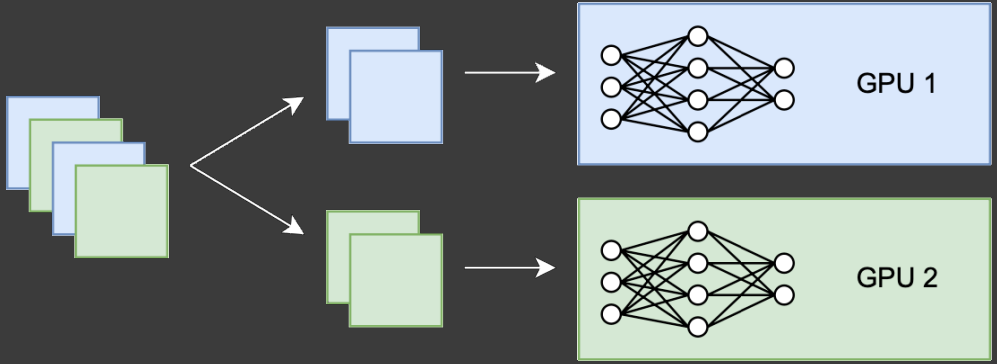
1. Open Baskerville docs
   `https://docs.baskerville.ac.uk`
2. Select Baserville Portal
3. Login if necessary
4. Select JupyterLab Apps list
5. Configure the session
6. Launch

Session configuration

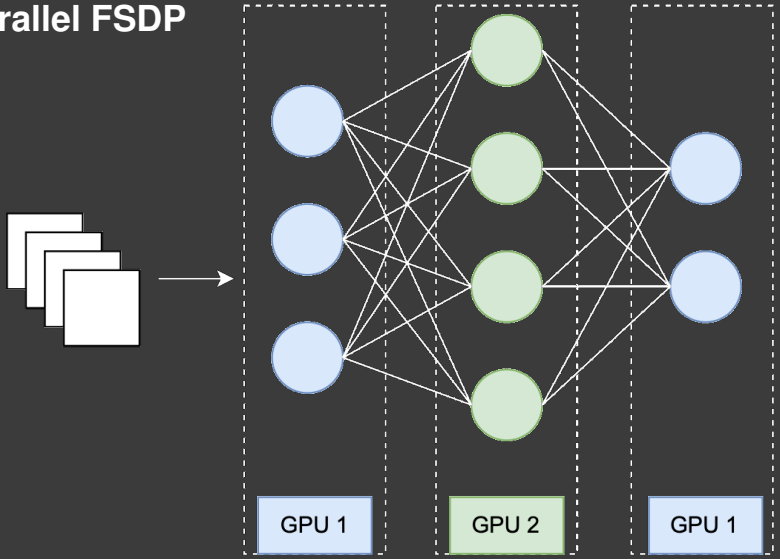
1. Kernel: `Python 3.11.3`
2. Show Conda Envs: `No`
3. Number of hours: `2`
4. Number of GPUs: `1`
5. Project: `vjgo8416-hpc2511`
6. Queue: `turing`


JupyterLab - Baskerville OnDem
portal.baskerville.ac.uk/pun/sys/dashboard/batch_connect/sys/bc_bask_jupyter/session_contexts/new
Interactive Apps
JupyterLab
GUIs
CST Studio Suite
Fiji
Linaro-Forge
RELION
cisTEM
Servers
TensorBoard
doppio
JupyterLab
This app will launch a JupyterLab server (supporting Python and Julia kernels) on Baskerville.
Kernel to load
Python 3.11.3 (2023a / GCCcore-12.3.0)
This defines the kernel you wish to load. The extra packages for use in Python can be selected from inside Jupyter using the Lmod extension. For further information please refer to the JupyterLab documentation.
Show Conda Environments
Include kernels for compatible Conda environments found in: ~/.conda/environments.txt. For further information please see our Conda environment documentation.
Number of hours
2
Number of GPUs
1
Number of GPUs to request. For each GPU you will get 25% of a node's total cores.
Baskerville Project
vjgo8416-hpc2511
Please select the Baskerville Project to which the job will be attached.
Queue
turing
Please select the Queue/QoS on which your job will run.
Launch

Data parallel

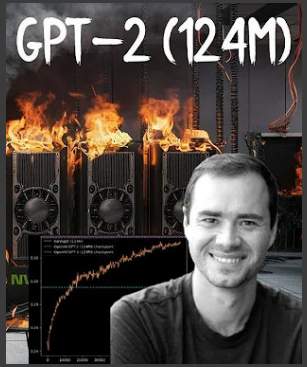


Model parallel FSDP

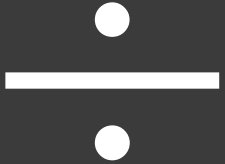


The plan

1. No more theory
2. Simplified GPT2 nano code for one GPU
3. Andrej Karpathy:
   `https://youtu.be/l8pRSuU81PU`
4. Extend to support Distributed Data Parallel
5. Extend using PyTorch Lightning



Your most important tools



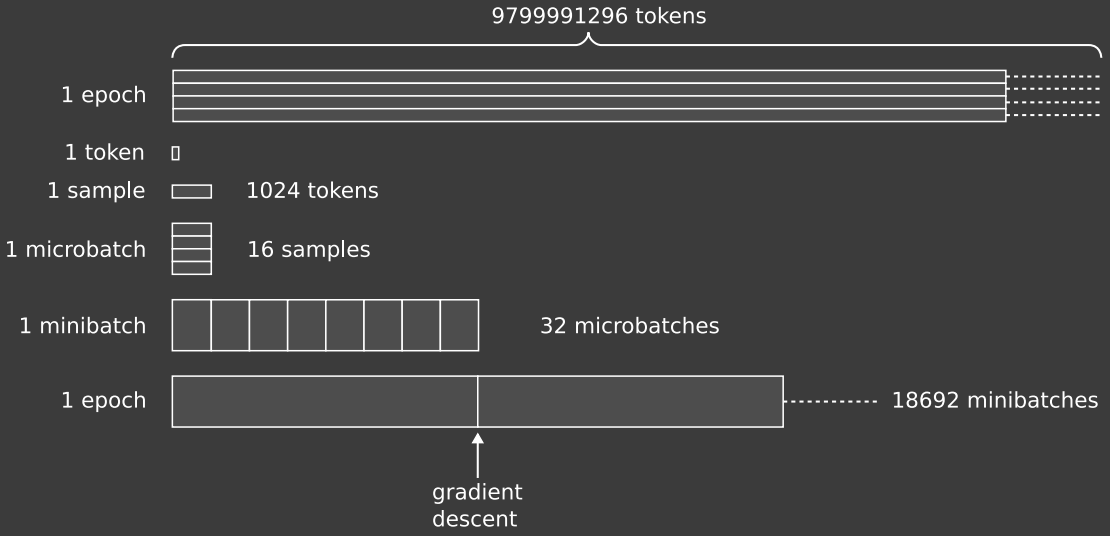
$(*, /)$

Training terminology

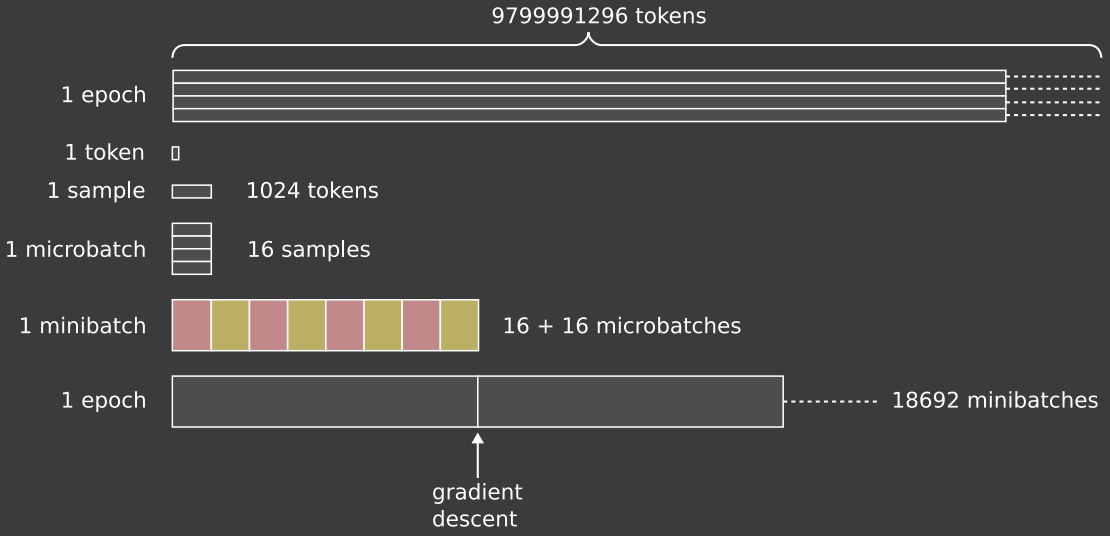
1. Vocabulary: 50257 embeddings
2. Dataset: 9799991296 FineWeb tokens, our training data
3. Sample: a sequence of 1024 tokens from the dataset
4. Microbatch: 16 samples
5. Minibatch (also called a batch): 32 microbatches, gradient accumulation performed afterwards
6. Step: the process for training on one minibatch
7. Epoch: one training sweep of the entire dataset, 18692 steps

$$18692 \times 16 \times 32 \times 1024 = 9799991296$$

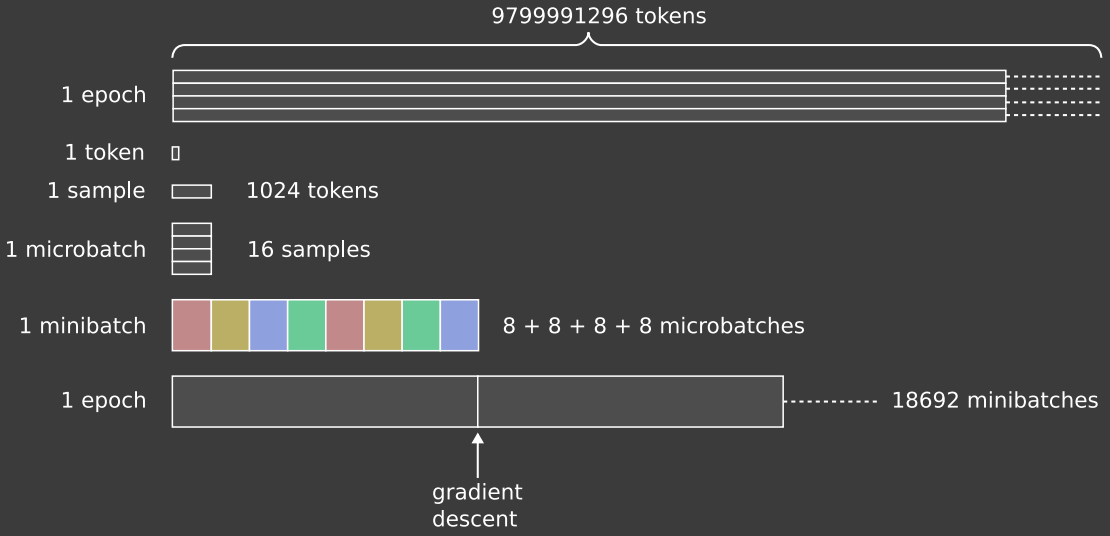
Training with 1 GPU



Training with 2 GPUs



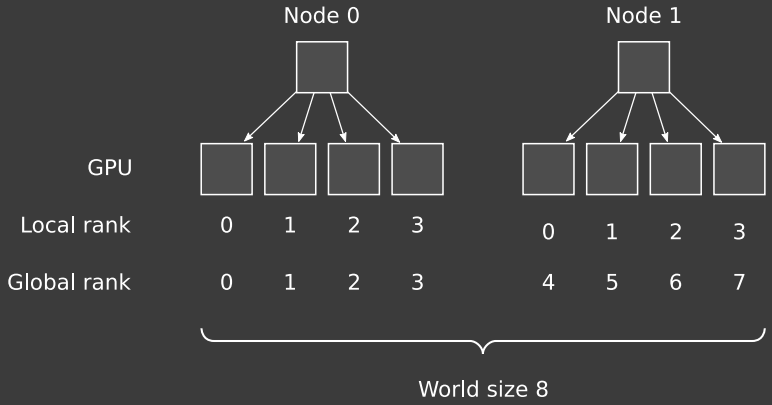
Training with 4 GPUs



Distributed training terminology

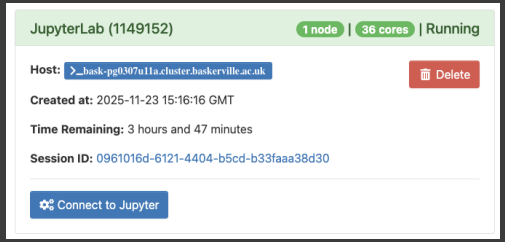
1. Training on n nodes, each with m GPUs
2. Total $n \times m$ GPUs
3. World size: $w = m \times n$
4. Global rank r_G is a unique index $r_G \in \{0, \ldots, w-1\}$
5. Local rank r_L is unique per device $r_L \in \{0, \ldots, m-1\}$
6. No node index, but we do have *hostnames*, *e.g.* `bask-pg0309u05a`, `bask-pg0309u06a`
7. We may also have multiple CPU *workers* for each node

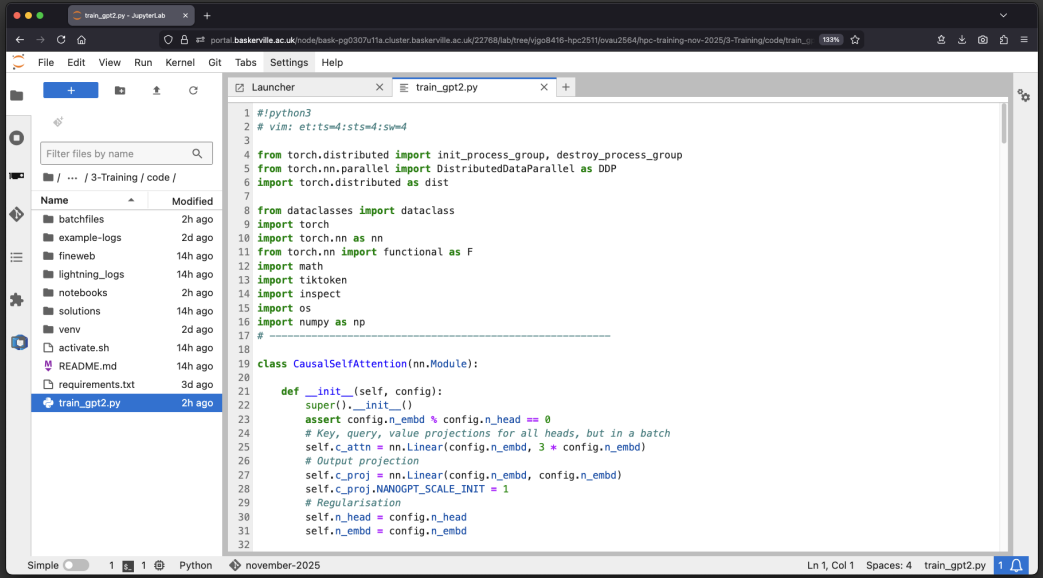
Training with multiple nodes



Time to look at the code

1. Open your JupyterLab page
2. Connect to Jupyter
3. Select the File Browser in the vertical bar on the left
4. Move to the directory:
   `/vjgo8416-hpc2511/$USER/hpc-training-nov-2025/3-Training/code`
5. Open the `train_gpt2.py` file by double-clicking on it




train_gpt2.py - JupyterLab
portal.baskerville.ac.uk/node/bask-pg0307u11a.cluster.baskerville.ac.uk/22768/lab/tree/vjgo8416-hpc2511/ovau2564/hpc-training-nov-2025/3-Training/code/train_g
133%
File Edit View Run Kernel Git Tabs Settings Help
Launcher
train_gpt2.py
Filter files by name
/ ··· / 3-Training / code /
Name
Modified
batchfiles 2h ago
example-logs 2d ago
fineweb 14h ago
lightning_logs 14h ago
notebooks 2h ago
solutions 14h ago
venv 2d ago
activate.sh 14h ago
README.md 14h ago
requirements.txt 3d ago
train_gpt2.py 2h ago
#!python3
# vim: et:ts=4:sts=4:sw=4
from torch.distributed import init_process_group, destroy_process_group
from torch.nn.parallel import DistributedDataParallel as DDP
import torch.distributed as dist
from dataclasses import dataclass
import torch
import torch.nn as nn
from torch.nn import functional as F
import math
import tiktoken
import inspect
import os
import numpy as np
# ---
class CausalSelfAttention(nn.Module):
def __init__(self, config):
super().__init__()
assert config.n_embd % config.n_head == 0
# Key, query, value projections for all heads, but in a batch
self.c_attn = nn.Linear(config.n_embd, 3 * config.n_embd)
# Output projection
self.c_proj = nn.Linear(config.n_embd, config.n_embd)
self.c_proj.NANOGPT_SCALE_INIT = 1
# Regularisation
self.n_head = config.n_head
self.n_embd = config.n_embd
Simple
1 1
Python
november-2025
Ln 1, Col 1
Spaces: 4
train_gpt2.py
1

GPT2 nano classes and functions

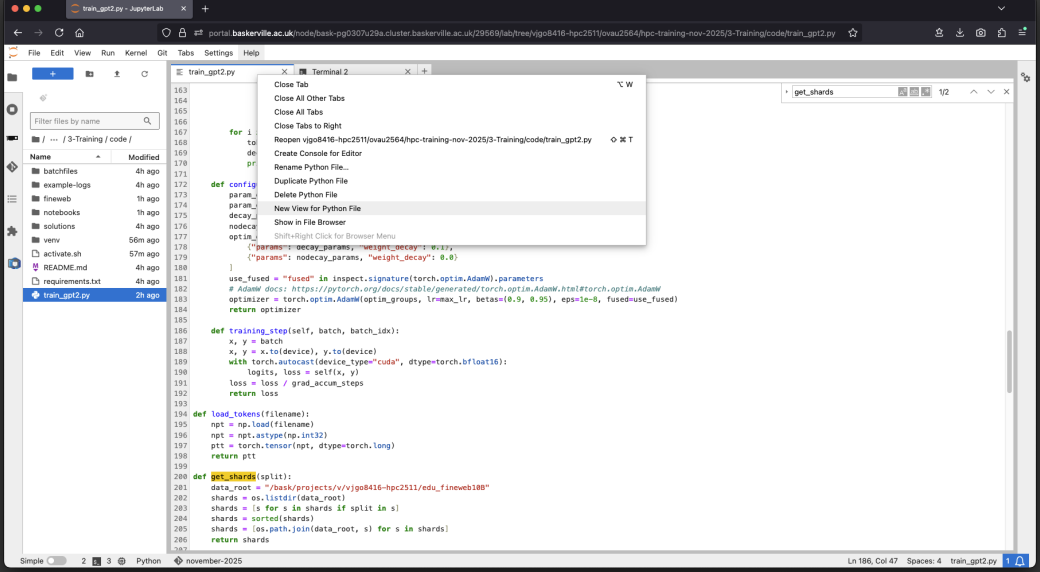
1. `CausalSelfAttention`, `MLP`, `Block`: model components
2. `GPTConfig`: model configuration
3. `GPT`: the model
4. `generate()`: inference
5. `configure_optimizers()`: training configuration
6. `training_step()`: one training step
7. `load_tokens()`: load a single FineWeb shard
8. `get_shards()`: find the FineWeb shards on disk
9. `DataIterator`: our dataset and data loader
10. `get_lr()`: calculate the learning rate

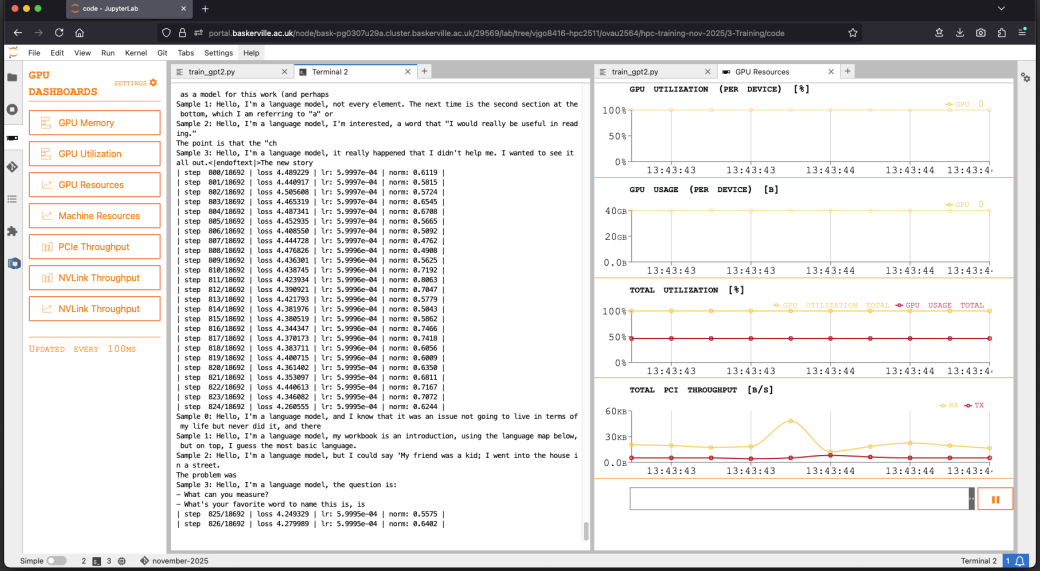
GPT2 nano execution

1. Set hyperparameters
2. Create model
3. Perform training loop

Training time!

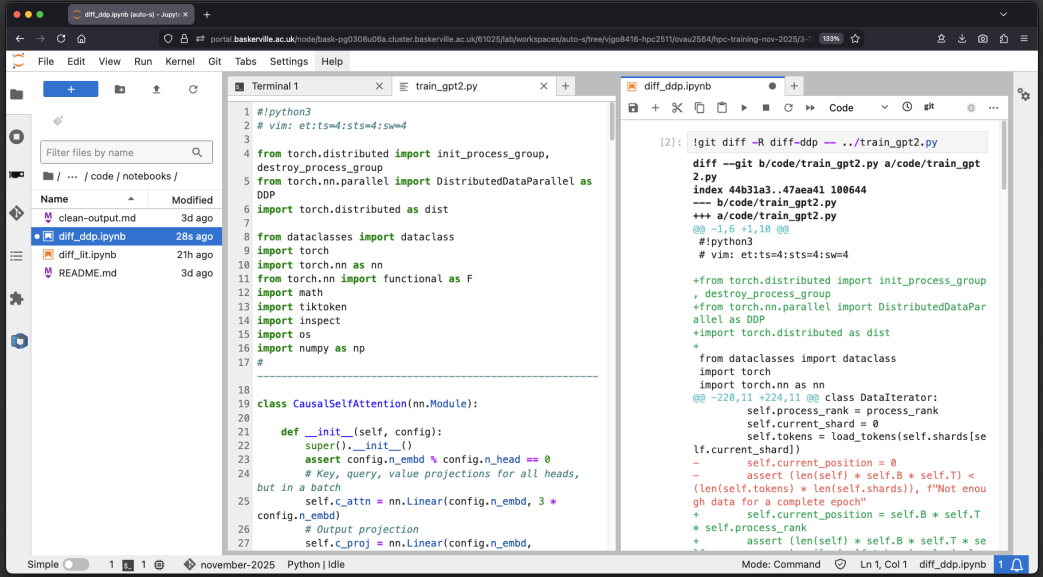
1. Right click on the `train_gpt2.py` tab
2. Select **New View for Python File**
3. Open a **GPU Resources** pane from the GPU Dashboard
4. Drag the pane to the tab space on the right
5. Open a terminal in the left tab space
6. `source activate.sh`
7. `python train_gpt2.py`


train_gpt2.py - JupyterLab
portal.baskerville.ac.uk/node/bask-pg0307u29a.cluster.baskerville.ac.uk/29569/lab/tree/vjgo8416-hpc2511/ovau2564/hpc-training-nov-2025/3-Training/code/train_gpt2.py
File Edit View Run Kernel Git Tabs Settings Help
Filter files by name
/ ... / 3-Training / code /
Name Modified
batchfiles 4h ago
example-logs 4h ago
fineweb 1h ago
notebooks 1h ago
solutions 4h ago
venv 56m ago
activate.sh 57m ago
README.md 4h ago
requirements.txt 4h ago
train_gpt2.py 2h ago
train_gpt2.py
Terminal 2
Close Tab ⌥ W
Close All Other Tabs
Close All Tabs
Close Tabs to Right
Reopen vjgo8416-hpc2511/ovau2564/hpc-training-nov-2025/3-Training/code/train_gpt2.py ⇧ ⌘ T
Create Console for Editor
Rename Python File...
Duplicate Python File
Delete Python File
New View for Python File
Show in File Browser
Shift+Right Click for Browser Menu
get_shards
1/2
for i
to
de
pr
def config
param_
param_
decay_
nodeca
optim_
{"params": decay_params, "weight_decay": 0.1},
{"params": nodecay_params, "weight_decay": 0.0}
use_fused = "fused" in inspect.signature(torch.optim.AdamW).parameters
# AdamW docs: https://pytorch.org/docs/stable/generated/torch.optim.AdamW.html#torch.optim.AdamW
optimizer = torch.optim.AdamW(optim_groups, lr=max_lr, betas=(0.9, 0.95), eps=1e-8, fused=use_fused)
return optimizer
def training_step(self, batch, batch_idx):
x, y = batch
x, y = x.to(device), y.to(device)
with torch.autocast(device_type="cuda", dtype=torch.bfloat16):
logits, loss = self(x, y)
loss = loss / grad_accum_steps
return loss
def load_tokens(filename):
npt = np.load(filename)
npt = npt.astype(np.int32)
ptt = torch.tensor(npt, dtype=torch.long)
return ptt
def get_shards(split):
data_root = "/bask/projects/v/vjgo8416-hpc2511/edu_fineweb10B"
shards = os.listdir(data_root)
shards = [s for s in shards if split in s]
shards = sorted(shards)
shards = [os.path.join(data_root, s) for s in shards]
return shards
Simple 2 3 Python november-2025
Ln 186, Col 47 Spaces: 4 train_gpt2.py 1


code - JupyterLab
portal.baskerville.ac.uk/node/bask-pg0307u29a.cluster.baskerville.ac.uk/29569/lab/tree/vjgo8416-hpc2511/ovau2564/hpc-training-nov-2025/3-Training/code
File Edit View Run Kernel Git Tabs Settings Help
GPU DASHBOARDS
SETTINGS
GPU Memory
GPU Utilization
GPU Resources
Machine Resources
PCIe Throughput
NVLink Throughput
NVLink Throughput
UPDATED EVERY 100MS
train_gpt2.py
Terminal 2
as a model for this work (and perhaps
Sample 1: Hello, I'm a language model, not every element. The next time is the second section at the bottom, which I am referring to "a" or
Sample 2: Hello, I'm a language model, I'm interested, a word that "I would really be useful in reading."
The point is that the "ch
Sample 3: Hello, I'm a language model, it really happened that I didn't help me. I wanted to see it all out.<|endoftext|>The new story
step 000/18692	loss 4.489229	lr: 5.9997e-04	norm: 0.6119
step 001/18692	loss 4.440917	lr: 5.9997e-04	norm: 0.5815
step 002/18692	loss 4.505608	lr: 5.9997e-04	norm: 0.5724
step 003/18692	loss 4.465319	lr: 5.9997e-04	norm: 0.6545
step 004/18692	loss 4.487341	lr: 5.9997e-04	norm: 0.6708
step 005/18692	loss 4.452935	lr: 5.9997e-04	norm: 0.5665
step 006/18692	loss 4.408550	lr: 5.9997e-04	norm: 0.5092
step 007/18692	loss 4.444728	lr: 5.9997e-04	norm: 0.4762
step 008/18692	loss 4.476826	lr: 5.9996e-04	norm: 0.4988
step 009/18692	loss 4.436301	lr: 5.9996e-04	norm: 0.5625
step 010/18692	loss 4.438745	lr: 5.9996e-04	norm: 0.7192
step 011/18692	loss 4.423934	lr: 5.9996e-04	norm: 0.8063
step 012/18692	loss 4.390921	lr: 5.9996e-04	norm: 0.7047
step 013/18692	loss 4.421793	lr: 5.9996e-04	norm: 0.5779
step 014/18692	loss 4.381976	lr: 5.9996e-04	norm: 0.5043
step 015/18692	loss 4.380519	lr: 5.9996e-04	norm: 0.5862
step 016/18692	loss 4.344347	lr: 5.9996e-04	norm: 0.7466
step 017/18692	loss 4.370173	lr: 5.9996e-04	norm: 0.7418
step 018/18692	loss 4.383711	lr: 5.9996e-04	norm: 0.6056
step 019/18692	loss 4.400715	lr: 5.9996e-04	norm: 0.6009
step 020/18692	loss 4.361402	lr: 5.9995e-04	norm: 0.6350
step 021/18692	loss 4.353097	lr: 5.9995e-04	norm: 0.6811
step 022/18692	loss 4.440613	lr: 5.9995e-04	norm: 0.7167
step 023/18692	loss 4.346082	lr: 5.9995e-04	norm: 0.7072
step 024/18692	loss 4.260555	lr: 5.9995e-04	norm: 0.6244
Sample 0: Hello, I'm a language model, and I know that it was an issue not going to live in terms of my life but never did it, and there
Sample 1: Hello, I'm a language model, my workbook is an introduction, using the language map below, but on top, I guess the most basic language.
Sample 2: Hello, I'm a language model, but I could say 'My friend was a kid; I went into the house in a street.
The problem was
Sample 3: Hello, I'm a language model, the question is:
— What can you measure?
— What's your favorite word to name this is, is
| step 025/18692 | loss 4.249329 | lr: 5.9995e-04 | norm: 0.5575 |
| step 026/18692 | loss 4.279989 | lr: 5.9995e-04 | norm: 0.6402 |
train_gpt2.py
GPU Resources
GPU UTILIZATION (PER DEVICE) [%]
GPU 0
100%
50%
0%
13:43:43 13:43:43 13:43:44 13:43:44 13:43:4
GPU USAGE (PER DEVICE) [B]
GPU 0
40GB
20GB
0.0B
13:43:43 13:43:43 13:43:44 13:43:44 13:43:4
TOTAL UTILIZATION [%]
GPU UTILIZATION TOTAL GPU USAGE TOTAL
100%
50%
0%
13:43:43 13:43:43 13:43:44 13:43:44 13:43:4
TOTAL PCI THROUGHPUT [B/S]
RX TX
60KB
30KB
0.0B
13:43:43 13:43:43 13:43:44 13:43:44 13:43:4
Simple
2 3
november-2025
Terminal 2
1

Upgrade to DDP

1. Open `train_gpt2.py` in the left hand tab space
2. Open `notebooks/diff_ddp.ipynb` in the right hand tab space
3. Execute the first cell of `diff_ddp.ipynb`


diff_ddp.ipynb (auto-s) - Jupyte
portal.baskerville.ac.uk/node/bask-pg0308u06a.cluster.baskerville.ac.uk/61025/lab/workspaces/auto-s/tree/vjgo8416-hpc2511/ovau2564/hpc-training-nov-2025/3-
133%
File Edit View Run Kernel Git Tabs Settings Help
Filter files by name
/ ... / code / notebooks /
Name
Modified
clean-output.md 3d ago
diff_ddp.ipynb 28s ago
diff_lit.ipynb 21h ago
README.md 3d ago
Terminal 1
train_gpt2.py
#!python3
# vim: et:ts=4:sts=4:sw=4
from torch.distributed import init_process_group, destroy_process_group
from torch.nn.parallel import DistributedDataParallel as DDP
import torch.distributed as dist
from dataclasses import dataclass
import torch
import torch.nn as nn
from torch.nn import functional as F
import math
import tiktoken
import inspect
import os
import numpy as np
#
class CausalSelfAttention(nn.Module):
def __init__(self, config):
super().__init__()
assert config.n_embd % config.n_head == 0
# Key, query, value projections for all heads, but in a batch
self.c_attn = nn.Linear(config.n_embd, 3 * config.n_embd)
# Output projection
self.c_proj = nn.Linear(config.n_embd,
diff_ddp.ipynb
Code
git
[2]:
!git diff -R diff-ddp -- ../train_gpt2.py
diff --git b/code/train_gpt2.py a/code/train_gpt2.py
index 44b31a3..47aea41 100644
--- b/code/train_gpt2.py
+++ a/code/train_gpt2.py
@@ -1,6 +1,10 @@
#!python3
vim: et:ts=4:sts=4:sw=4
+from torch.distributed import init_process_group, destroy_process_group
+from torch.nn.parallel import DistributedDataParallel as DDP
+import torch.distributed as dist
+
from dataclasses import dataclass
import torch
import torch.nn as nn
@@ -220,11 +224,11 @@ class DataIterator:
self.process_rank = process_rank
self.current_shard = 0
self.tokens = load_tokens(self.shards[self.current_shard])
- self.current_position = 0
- assert (len(self) * self.B * self.T) < (len(self.tokens) * len(self.shards)), f"Not enough data for a complete epoch"
+ self.current_position = self.B * self.T * self.process_rank
+ assert (len(self) * self.B * self.T * se
Simple
1 1
november-2025
Python | Idle
Mode: Command
Ln 1, Col 1
diff_ddp.ipynb
1

Understanding unified diffs

1. `@@` prefix indicates line numbers
   `@@ -before,len +after,len @@`
2. `+` prefix in green indicates lines added
3. `-` prefix in red indicates lines removed
4. Replay the cell to update the diff after making changes

```
@@ -145,7 +149,7 @@ class GPT(nn.Module):
                loss = F.cross_entropy(logits.view
s.view(-1))
        return logits, loss

-    def generate(self):
+    def generate(self, rank):
        num_return_sequences = 4
        max_length = 32
        tokens = enc.encode("Hello, I'm a lang
@@ -153,7 +157,7 @@ class GPT(nn.Module):
        tokens = tokens.unsqueeze(0).repeat(nu
        xgen = tokens.to(device)
        sample_rng = torch.Generator(device=de
-       sample_rng.manual_seed(42)
+       sample_rng.manual_seed(42 + rank)
        while xgen.size(1) < max_length:
            with torch.no_grad():
                with torch.autocast(device_typ
6):
@@ -207,22 +211,24 @@ def get_shards(split):

 class DataIterator:

-    def __init__(self, B, T):
+    def __init__(self, B, T, num_processes, pr
```

Manually apply the diff

1. Imports
2. Wrap the model in the `DDP` class
3. Initialise the world parameters
   3.1 Harvest **world size**, **rank** and **local rank** from the environment
   3.2 Where do these come from?
4. Initialise the process group
5. Add a stride to our data iterator

Observations

1. Most of this is boilerplate
2. The hardest part is sharding the data correctly
3. Because this is *data parallel*
4. Communication between processes is also hard

. . . but done for us by `torch.distributed` and `DDP`

Training time!

See `batch-ddp-1n1g.sh`. `batch-ddp-1n2g.sh` and `batch-ddp-2n2g.sh`

```
# Single node
python -m torch.distributed.launch \
    --standalone \
    --nproc-per-node=${SLURM_GPUS_PER_NODE} \
    train_gpt2.py

# Multi-node
srun bash -c 'python -m torch.distributed.launch \
    --nproc_per_node=${SLURM_GPUS_PER_NODE} \
    --nnodes=${SLURM_NNODES} \
    --master-port=${MASTER_PORT} \
    --master-addr=${MASTER_ADDR} \
    --node_rank=${SLURM_PROCID} \
    train_gpt2.py'
```

Upgrade to Lightning

1. Lightning is conceptually different
2. Code is organised in `LightningModule`: init, train step, validation step, test step, optimisers
3. Code outside `LightningModule` is automated by `Trainer`
4. Remove code moving data to the GPU

Upgrade to Lightning

1. Open `train_gpt2.py` in the left hand tab space
2. Open `diff_lit.ipynb` in the right hand tab space
3. Execute the first cell of `diff_lit.ipynb`

Manually apply the diff

1. Imports
2. Switch from `Module` to `LightningModule`
3. The device is established for us
4. The Learning Rate scheduler is baked in
5. Lightning handles where the data goes
6. Drop periodic example generation
7. DataIterator must be managed by a DataLoader
8. World initialisation now handled by Lightning
9. The training loop is all Lightning

Observations

1. The training code is greatly simplified
2. Lightning is SLURM-aware
3. So multi-node training is even easier
4. We can now switch to other strategies

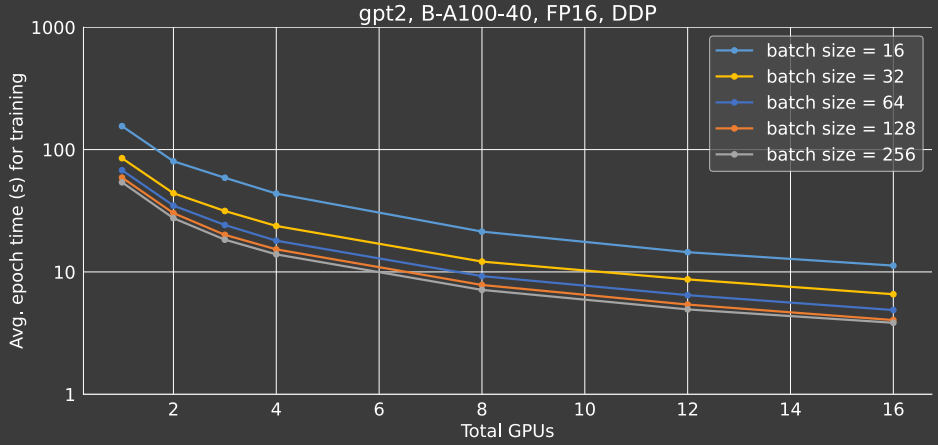
Training time!

See `batch-lit-1n1g.sh` – `batch-lit-2n4g.sh`

```
# From inside a SLURM batch script
srun python train_gpt2.py

# Queue execution to run
sbatch batch-lit-1n1g.sh
```

Scaling across nodes on A100 (40 GiB) GPUs



Wrapping up

1. Converting code for distributed training is doable
2. Review and try out the batch scripts
3. More examples in the `hpc-landscapes` repository

`https://github.com/alan-turing-institute/hpc-landscape`