

HIGH PERFORMANCE RESEARCH COMPUTING

ACES: Introduction to HPC and AI for Faculty and Researchers

Distributed AI Model Training on NVIDIA GPUs

10/07/2025

Zhenhua He and Dinesh S. Devarajan



High Performance
Research Computing
DIVISION OF RESEARCH

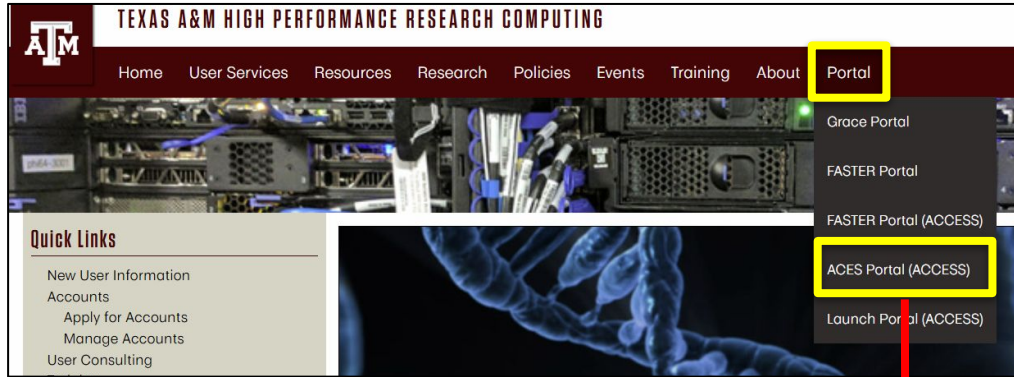


Outline

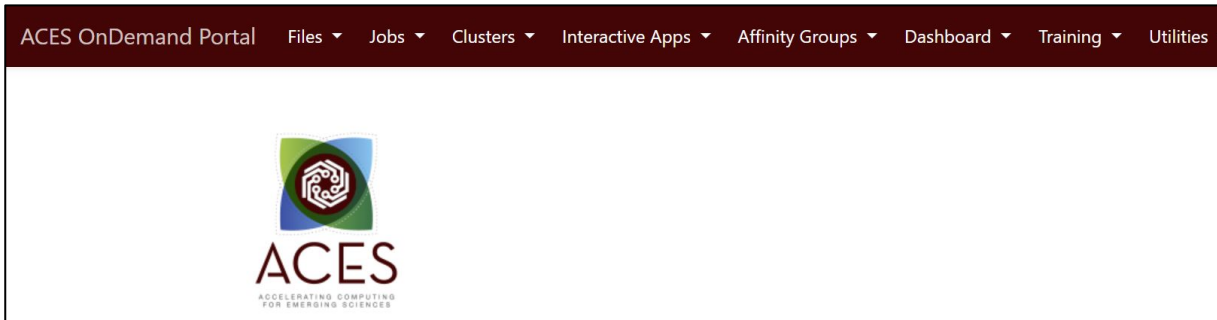
- Lab 0: Virtual Environment Set up on HPC cluster
- Lab 0 (cont.): Introduction to DL in JupyterLab
- Lab 1: Transition from CPU to a single GPU
- Lab 2: Scaling from one GPU to multiple GPUs on a single node
- Lab 3: Extending to multi-node distributed training

Lab 0. Virtual Environment Setup

ACES Portal

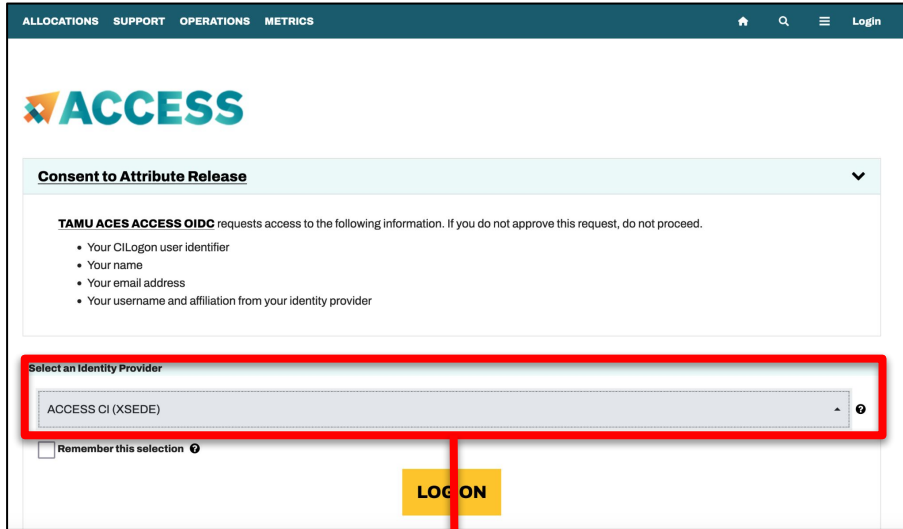


ACES Portal <https://portal-aces.hprc.tamu.edu/> is the web-based user interface for the Launch cluster



Open OnDemand (OOD) is an advanced web-based graphical interface framework for HPC users

Accessing ACES via the Portal (ACCESS)



ALLOCATIONS SUPPORT OPERATIONS METRICS

ACCESS

Consent to Attribute Release

TAMU ACES ACCESS OIDC requests access to the following information. If you do not approve this request, do not proceed.

- Your CILogon user identifier
- Your name
- Your email address
- Your username and affiliation from your identity provider

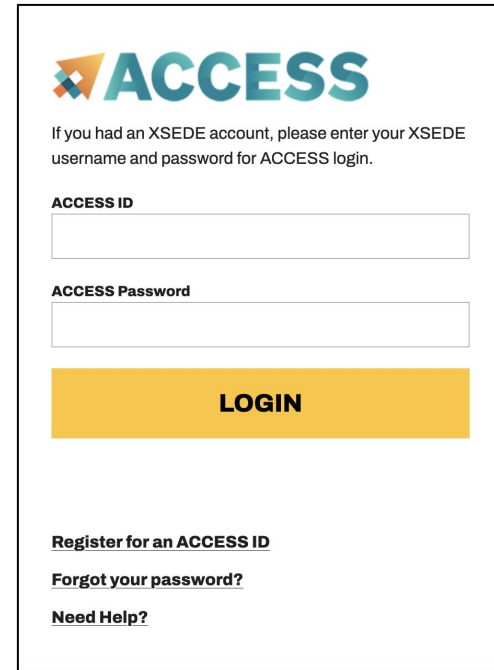
Select an Identity Provider

ACCESS CI (XSEDE)

☐ Remember this selection

LOG ON

Select the Identity Provider appropriate for your account.



ACCESS

If you had an XSEDE account, please enter your XSEDE username and password for ACCESS login.

ACCESS ID

ACCESS Password

LOGIN

[Register for an ACCESS ID](#)

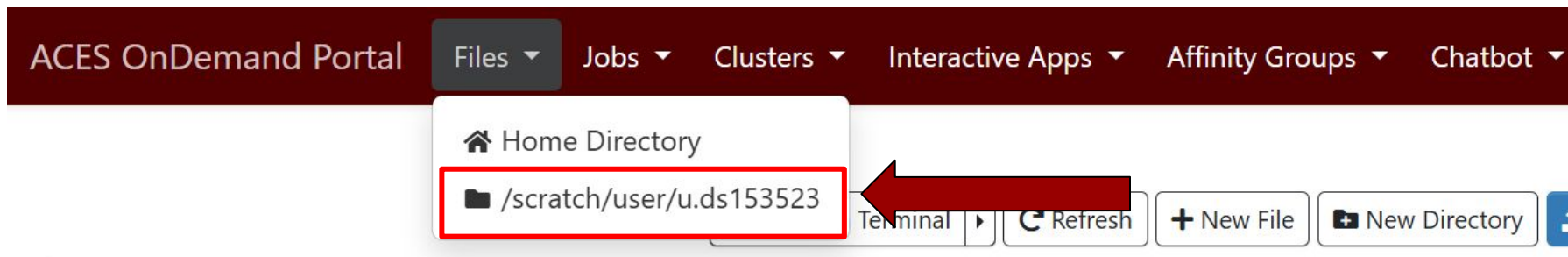
[Forgot your password?](#)

[Need Help?](#)

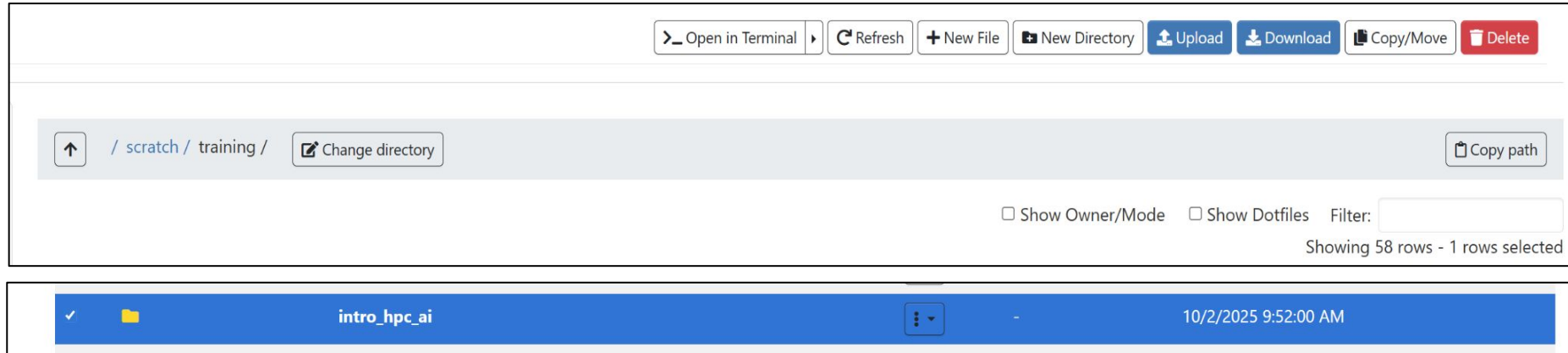
Log in using your ACCESS or institutional credentials.

Accessing File Manager

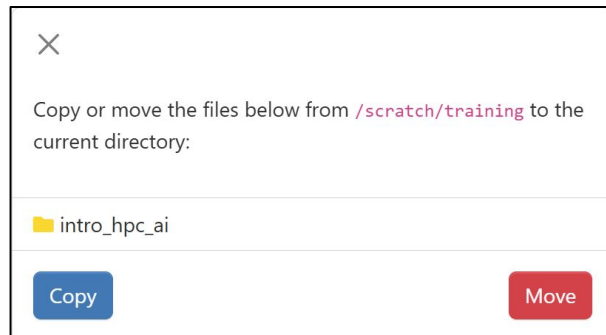
Click on “Files” menu → `/scratch/user/USERNAME`



Copying the training materials



The screenshot shows a file management interface. At the top, there is a toolbar with buttons: "Open in Terminal", "Refresh", "New File", "New Directory", "Upload", "Download", "Copy/Move", and "Delete". Below the toolbar, the current directory is shown as "/ scratch / training /" with a "Change directory" button. To the right, there is a "Copy path" button. Below the directory path, there are checkboxes for "Show Owner/Mode" and "Show Dotfiles", and a "Filter:" input field. At the bottom right, it says "Showing 58 rows - 1 rows selected". Below this, a blue bar represents the selected directory "intro_hpc_ai" with a checkmark icon on the left and a date "10/2/2025 9:52:00 AM" on the right.

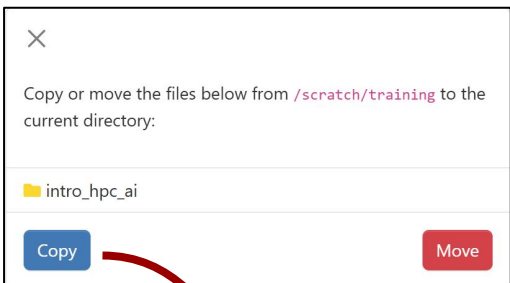


A modal dialog box with a close button (X) in the top left corner. The text inside says "Copy or move the files below from /scratch/training to the current directory:". Below this text, there is a list of files: "intro_hpc_ai". At the bottom of the dialog, there are two buttons: "Copy" (blue) and "Move" (red).

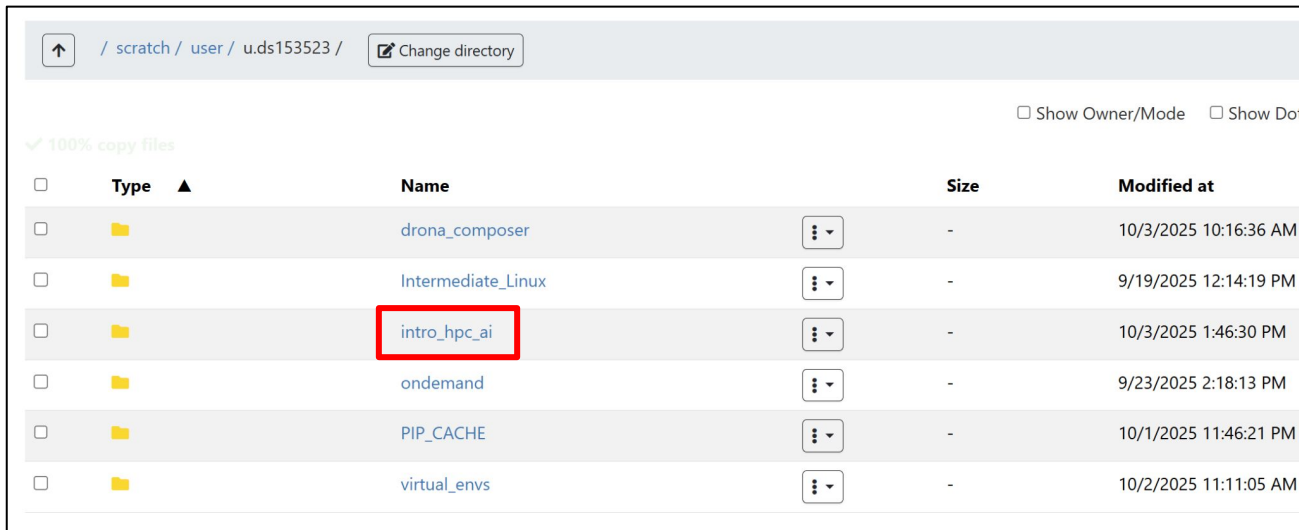


Selection confirmation

Copying the training materials (cont.)

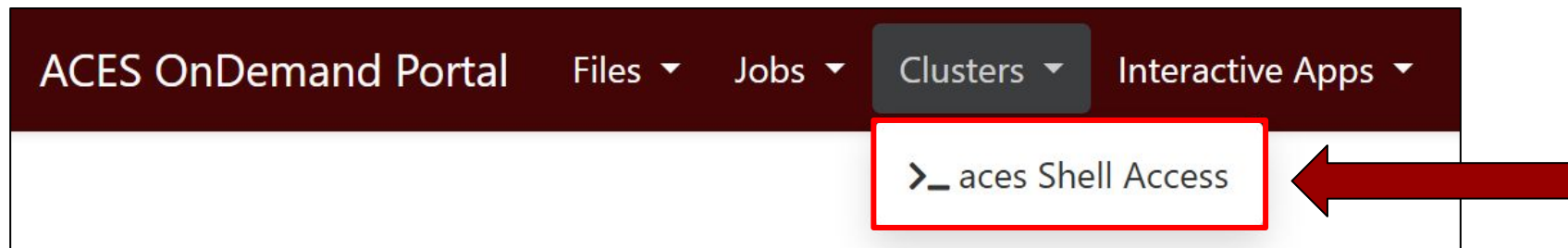


Copying from scratch/training to scratch/user/USERNAME



Get a Shell on ACES

Click on “Clusters” menu → _aces Shell Access



Success!

Welcome to the ACES login node.

Check which login node you are on and type it in the chat.

Your current disk quotas are:

Disk	Disk Usage	Limit	File Usage	Limit
/home/u.ds153523	84K	10.0G	17	10000
/scratch/user/u.ds153523	793M	1.0T	47	250000
/scratch/group/p.tra220029.000	12.5G	1.0T	10286	500000

Type 'showquota' to view these quotas again.

[u.ds153523@aces-login2 ~]\$

ModuLair Framework

- The ModuLair Framework is designed to streamline the management of Python environments on HPC clusters.

Core Functionalities:

- Creating Environments (`create_venv`)
- Activating Environments (`activate_venv`)
- Listing Environments (`list_envs`)
- Deleting Environments (`delete_venv`)

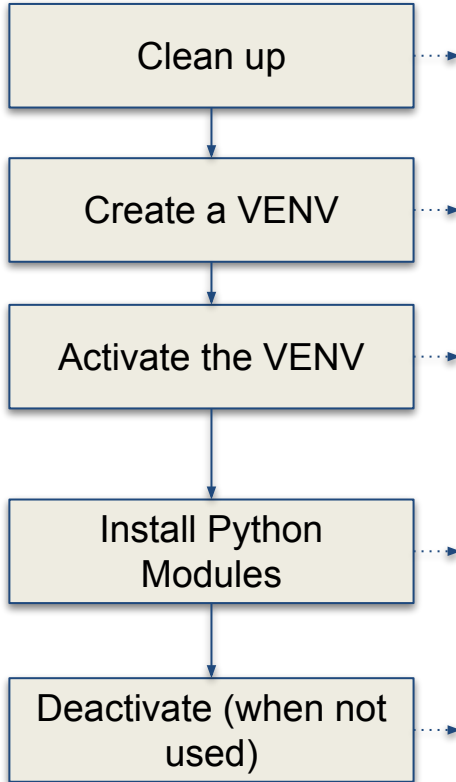
Short course: Setting up coding environments for HPC
(Nov. 21, 2025)

hprc.tamu.edu/kb/Software/ModuLair

Advantages of ModuLair

- ModuLair builds virtual environments in \$SCRATCH (at \$SCRATCH/virtual_envs) instead of using \$HOME to avoid filling your home directory space quickly.
- ModuLair automatically records the toolchain and Python modules used to create each environment, so you don't need to keep track of them yourself.

ModuLair



```
# clean up
cd $SCRATCH/intro_hpc_ai/notebooks
module purge

# create a Python virtual environment
create_venv hpc_ai_env -d "Environment for Intro to HPC
and AI" -t "GCCcore/12.2.0 Python/3.10.8"

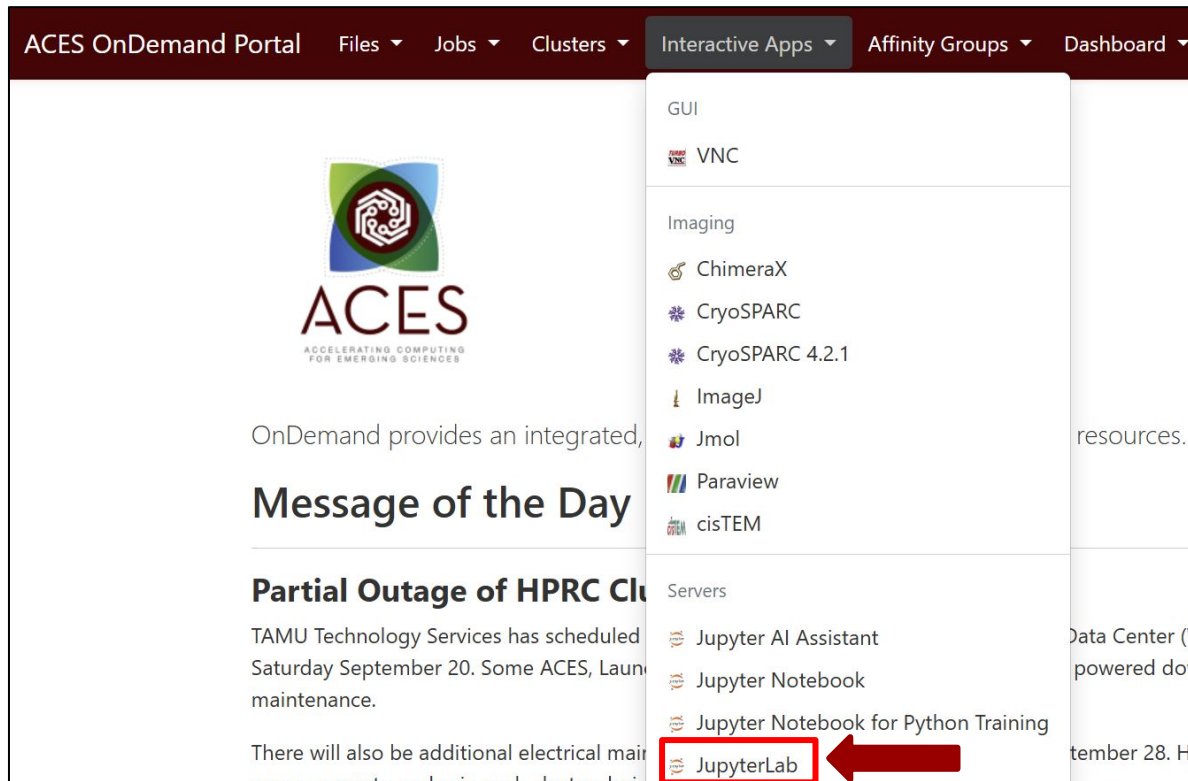
# activate the virtual environment
source activate_venv hpc_ai_env

# install required packages
pip3 install jupyter
pip3 install pandas matplotlib
pip3 install scikit-learn
pip3 install torch torchvision \
    --index-url https://download.pytorch.org/whl/cu126

# deactivate the virtual environment
deactivate
```

The training materials are also available at
https://github.com/happidence/Intro_HPC_AI.git

Go to JupyterLab Page



The screenshot shows the ACES OnDemand Portal interface. The top navigation bar includes links for 'Files', 'Jobs', 'Clusters', 'Interactive Apps', 'Affinity Groups', and 'Dashboard'. The 'Interactive Apps' dropdown menu is open, displaying a list of applications: GUI, VNC, Imaging (ChimeraX, CryoSPARC, CryoSPARC 4.2.1, ImageJ, Jmol, Paraview, cisTEM), Servers (Jupyter AI Assistant, Jupyter Notebook, Jupyter Notebook for Python Training, and JupyterLab), and Data Center (V-powered dov). The 'JupyterLab' option is highlighted with a red box and a red arrow pointing to it. The main content area features the ACES logo, a 'Message of the Day' section, and a 'Partial Outage of HPRC Clusters' announcement.

ACES OnDemand Portal Files Jobs Clusters Interactive Apps Affinity Groups Dashboard

GUI

VNC

Imaging

ChimeraX

CryoSPARC

CryoSPARC 4.2.1

ImageJ

Jmol

Paraview

cisTEM

Servers

Jupyter AI Assistant

Jupyter Notebook

Jupyter Notebook for Python Training

JupyterLab

Data Center (V-powered dov)

September 28. H

OnDemand provides an integrated, resources.

Message of the Day

Partial Outage of HPRC Clusters

TAMU Technology Services has scheduled Saturday September 20. Some ACES, Launch maintenance.

There will also be additional electrical main

JupyterLab Page

ACES OnDemand Portal Files Jobs Clusters Interactive Apps Affinity Groups Dashboard Training Utilities My Interactive Sessions

Home / My Interactive Sessions / JupyterLab

Interactive Apps

- GUI
- VNC
- Imaging
- ChimeraX
- CryoSPARC
- CryoSPARC 4.2.1
- ImageJ
- Jmol
- Paraview
- cisTEM
- Servers

JupyterLab

This app will launch a [JupyterLab](#) server on the [ACES](#) cluster.

Type of environment

TAMU ModuLair (Python virtualenv manager) ←

Select the type of environment in which Jupyter is installed. [Help me choose](#)

TAMU ModuLair environment (required)

hpc_ai_env ←

Select the name of the TAMU ModuLair environment to be activated.

Your TAMU ModuLair environment is expected to have a Jupyter package installed. Please see [instructions](#)

Node type

CPU only

Other fields:

Node Type: CPU

Number of hours: 3

Number of cores: 3

Total memory (GB): 10

If your venv fails, please use a shared one

ACES OnDemand Portal Files Jobs Clusters Interactive Apps Affinity Groups Chatbot Dashboard Training

Home / My Interactive Sessions / JupyterLab

Interactive Apps

GUI

NextSilicon VNC

VNC

Imaging

ChimeraX

CryoSPARC

CryoSPARC 4.2.1

ImageJ

Jmol

Paraview

cisTEM

JupyterLab

This app will launch a JupyterLab server on the ACES cluster.

Type of environment

Module load + Python virtualenv

Select the type of environment in which Jupyter is installed. [Help me choose](#)

Python module to be loaded

Python/3.12.3

Select a Python module to load.

Optional Python virtual environment to be activated

/sw/hprc/sw/Python/virtualenvs/3.12.3/hpc_ai_env/bin/activate

Enter the full path to a python virtual environment "activate" script to be activated. This field is optional.

Path to the shared environment:

E.g. /scratch/user/netid/...path/sw/hprc/sw/Python/virtualenvs/3.12.3/hpc_ai_env/bin/activate

Other fields:

Node Type: CPU

Number of hours: 3

Number of cores: 3

Total memory (GB): 10

Connect to JupyterLab

ACES OnDemand Portal Files ▾ Jobs ▾ Clusters ▾ Interactive Apps ▾ Affinity Groups ▾ Dashboard ▾ Training ▾ Utilities ▾

Session was successfully created. ✕

[Home](#) / [My Interactive Sessions](#)

Interactive Apps

- GUI
-  VNC
- Imaging
-  ChimeraX
-  CryoSPARC
-  CryoSPARC 4.2.1
-  ImageJ
-  Jmol
-  Paraview

JupyterLab (1267915) 1 node | 3 cores | Running

Host: >_ ac014 ✕ Delete

Created at: 2025-10-01 23:56:51 CDT

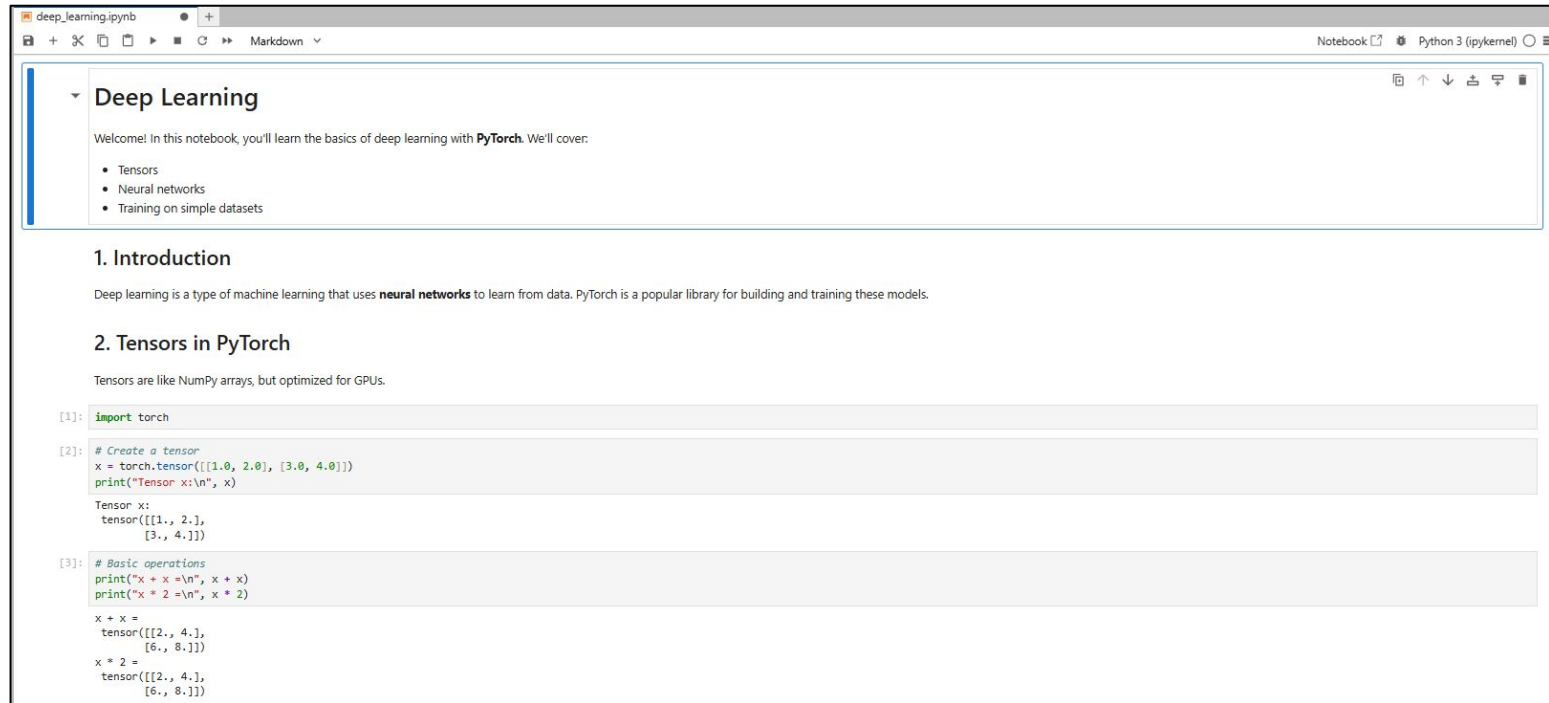
Time Remaining: 2 hours and 59 minutes

Session ID: 137f6636-bb28-4a58-8d7e-975a3d71c8c8

Type of environment: modular

 [Connect to JupyterLab](#) 

Lab 0 (cont.). Intro to DL in JupyterLab



The screenshot shows a JupyterLab interface with a notebook titled 'deep_learning.ipynb'. The notebook is in 'Markdown' view. The left sidebar shows a tree view with 'Deep Learning' selected. The main content area displays the following text:

Deep Learning

Welcome! In this notebook, you'll learn the basics of deep learning with **PyTorch**. We'll cover:

- Tensors
- Neural networks
- Training on simple datasets

1. Introduction

Deep learning is a type of machine learning that uses **neural networks** to learn from data. PyTorch is a popular library for building and training these models.

2. Tensors in PyTorch

Tensors are like NumPy arrays, but optimized for GPUs.

```
[1]: import torch

[2]: # Create a tensor
x = torch.tensor([[1.0, 2.0], [3.0, 4.0]])
print("Tensor x:\n", x)

Tensor x:
tensor([[1., 2.],
        [3., 4.]])

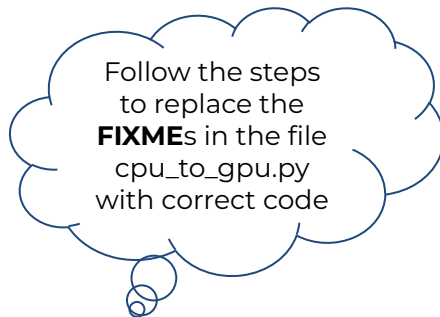
[3]: # Basic operations
print("x + x =\n", x + x)
print("x * 2 =\n", x * 2)

x + x =
tensor([[2., 4.],
        [6., 8.]])
x * 2 =
tensor([[2., 4.],
        [6., 8.]])
```

Lab 1. Transition from CPU to a single GPU

1. Define the device

In `cpu_to_gpu.py`, add this near the top:



```
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
```

2. Move the model to GPU

Change:

```
model = Net()
```

to:

```
model = Net().to(device)
```

3. Move data and labels to GPU during training

Inside the training loop, change:

```
outputs = model(data)
loss = criterion(outputs, target)
```

to:

```
data, target = data.to(device), target.to(device)

outputs = model(data)
loss = criterion(outputs, target)
```

4. Move data and labels to GPU during evaluation

Inside the evaluation loop, change:

```
outputs = model(data)
loss = criterion(outputs, target)
```

to:

```
data, target = data.to(device), target.to(device)
outputs = model(data)
loss = criterion(outputs, target)
```

5. Handle Lazy Initialization Layer

In `Net.forward`, the fully connected layer is created lazily. Modify:

```
self._fc = nn.Linear(self.flatten_dim, 10)
```

to:

```
self._fc = nn.Linear(self.flatten_dim, 10).to(x.device)
```

6. Modify SLURM Script for GPU

In `cpu_to_gpu.slurm`,

```
#SBATCH --job-name=gpu_job # Change the job name to be gpu related
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=1
#SBATCH --partition=gpu # Change the partition to be gpu
#SBATCH --gres=gpu:h100:1 # Request 1 h100 GPU
#SBATCH --cpus-per-task=8 # Set number of CPU cores per task (e.g., 8)
#SBATCH --mem=32G
#SBATCH --time=01:00:00
##SBATCH --reservation=training

module load WebProxy
source activate_venv hpc_ai_env

# ---- Run Training ----
# use modified GPU training script
python cpu_to_gpu.py
```

After completing all the steps above, go back to your terminal and run:

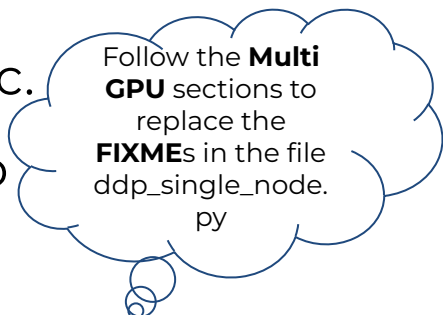
```
cd $SCRATCH/intro_hpc_ai/exercises/1.CPU_to_GPU
sbatch cpu_to_gpu.slurm
```

Lab 2. Scaling from one GPU to multiple GPUs on a single node

1. Imports

- **Single GPU** only torch, torch.nn, torch.optim, etc.
- **Multi GPU** adds distributed training modules to

`ddp_single_node.py`



Follow the **Multi GPU** sections to replace the **FIXMEs** in the file `ddp_single_node.py`

```
import torch.distributed as dist
from torch.nn.parallel import DistributedDataParallel as DDP
from torch.utils.data import distributed
```

2&3. Process group initialization

- **Single GPU** No process group (runs as one process).
- **Multi GPU** Adds setup/cleanup for distributed training.

```
def setup(rank, world_size):  
    os.environ['MASTER_ADDR'] = '127.0.0.1'  
    os.environ['MASTER_PORT'] = '29500'  
    dist.init_process_group("nccl", rank=rank, world_size=world_size)  
    torch.cuda.set_device(rank)  
  
def cleanup():  
    dist.destroy_process_group()
```

4. Device setup

- **Single GPU**

```
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
```

- **Multi GPU** Each process explicitly binds to a GPU based on rank

```
device = torch.device(f'cuda:{rank}')
```

5. Dataset handling

- **Single GPU** Normal DataLoader with shuffle=True

```
train_loader = DataLoader(dataset, batch_size=64, shuffle=True)
```

- **Multi GPU** Use DistributedSampler so each process gets a unique shard of data.

```
sampler = distributed.DistributedSampler(dataset,  
    num_replicas=world_size, rank=rank)  
loader = DataLoader(dataset, batch_size=64, sampler=sampler)
```

6. Model wrapping

- **Single GPU** Just moves the model to device.

```
model = Net().to(device)
```

- **Multi GPU** Wraps the model in DDP.

```
model = Net().to(device)  
model = DDP(model, device_ids=[rank])
```

7. Training loop

- **Single GPU** Standard training loop — one process prints progress.
- **Multi GPU**
Calls `sampler.set_epoch(epoch)` at the start of each epoch to reshuffle across processes and avoid bias.

```
sampler.set_epoch(epoch)
```

Only rank 0 (the main process) prints logs

```
if batch_idx % 100 == 0 and rank == 0:  
    print(...)
```

End-of-epoch stats also restricted to `rank == 0`.

8. Launch training

- **Single GPU**

```
python single_gpu.py
```

- **Multi GPU** Use `torch.multiprocessing.spawn` to launch one process per GPU

```
if __name__ == "__main__":  
    world_size = torch.cuda.device_count()  
    torch.multiprocessing.spawn(train, args=(world_size,),  
                                nprocs=world_size, join=True)
```

9. Modify SLURM Script for Multi GPU

Follow the instructions in `ddp_single_node.slurm` and modify it. After completing all the steps above, go back to your terminal and run:

```
cd $SCRATCH/intro_hpc_ai/exercises/2.SingleGPU_to_MultiGPU  
sbatch ddp_single_node.slurm
```

Lab 3. Extending to multi-node distributed training

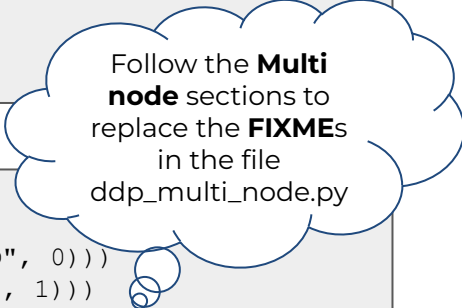
1&2. DDP Setup Changes

- **Single node**

```
def setup(rank, world_size):  
    os.environ['MASTER_ADDR'] = '127.0.0.1'  
    os.environ['MASTER_PORT'] = '29500'  
    dist.init_process_group("nccl", rank=rank, world_size=world_size)  
    torch.cuda.set_device(rank)
```

- **Multi node** In `ddp_multi_node.py`

```
def setup():  
    rank = int(os.environ.get("RANK", os.environ.get("SLURM_PROCID", 0)))  
    local_rank = int(os.environ.get("LOCAL_RANK", os.environ.get("SLURM_LOCALID", 0)))  
    world_size = int(os.environ.get("WORLD_SIZE", os.environ.get("SLURM_NTASKS", 1)))  
  
    if "MASTER_ADDR" not in os.environ:  
        os.environ["MASTER_ADDR"] = os.environ.get("SLURM_NODELIST", "127.0.0.1")  
    if "MASTER_PORT" not in os.environ:  
        os.environ["MASTER_PORT"] = str(12000 + os.getpid() % 10000)  
  
    dist.init_process_group("nccl", rank=rank, world_size=world_size)  
    torch.cuda.set_device(local_rank)
```



Follow the **Multi node** sections to replace the **FIXMEs** in the file `ddp_multi_node.py`

Key differences

- Single-node: rank and world size explicitly passed.
- Multi-node: rank, local_rank, and world_size inferred from SLURM environment variables.
- MASTER_ADDR and MASTER_PORT can be dynamically set from SLURM for multi-node runs.

3. Multiprocessing / Launching

- **Single node**

```
world_size = torch.cuda.device_count()
torch.multiprocessing.spawn(train, args=(world_size,), nprocs=world_size, join=True)
```

- **Multi node**

```
# Direct call to train(), as SLURM + srun handles spawning across nodes
train()
```

Key differences

- Single-node uses `torch.multiprocessing.spawn`.
- Multi-node relies on SLURM `srun` to launch one process per GPU per node, so no explicit spawn is needed.

4. SLURM Job Script Changes

- **Single node**

```
#SBATCH --nodes=1
#SBATCH --gres=gpu:h100:2
#SBATCH --ntasks-per-node=2

...
python ddp_single_node.py
```

- **Multi node** In `ddp_multi_node.slurm`,

```
#SBATCH --nodes=2
#SBATCH --gres=gpu:h100:2
#SBATCH --ntasks-per-node=2
...
export MASTER_ADDR=$(scontrol show hostnames $SLURM_NODELIST | head -n 1)
export MASTER_PORT=$((12000 + RANDOM % 10000))
export NCCL_DEBUG=INFO
export NCCL_SOCKET_IFNAME=^lo,docker

srun --export=ALL python ddp_multi_node.py
```

Key differences

- Nodes increased from 1 $\rightarrow$ 2.
- MASTER_ADDR is set to the first hostname in the node list.
- MASTER_PORT is randomized to avoid conflicts.
- srun handles launching multiple processes across nodes.
- Optional: NCCL debug flags for multi-node troubleshooting.

Submit the job

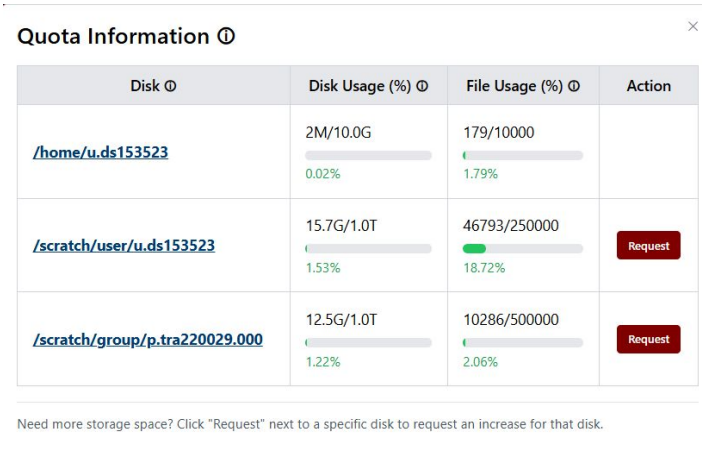
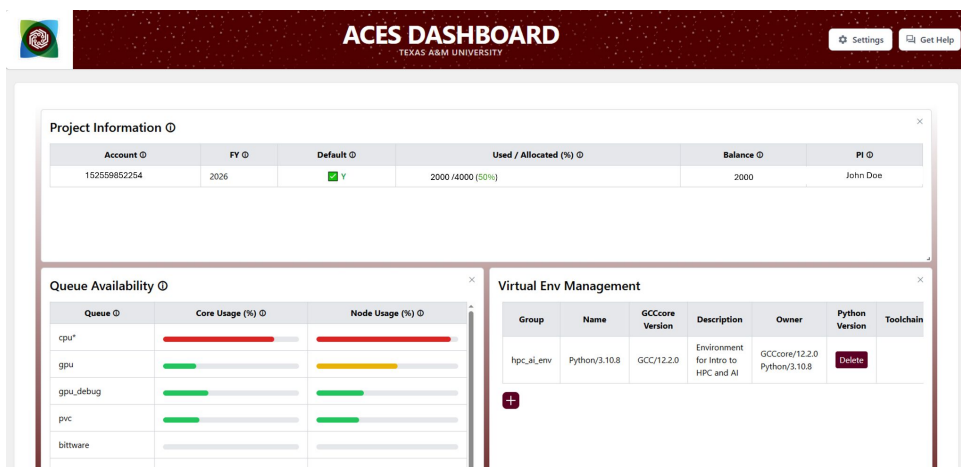
After completing all the steps above, go back to your terminal and run:

```
cd $SCRATCH/intro_hpc_ai/exercises/3.SingleNode_to_MultiNode  
sbatch ddp_multi_node.slurm
```

Need Help?

First check the FAQ: <https://hprc.tamu.edu/kb/FAQ/Accounts>

- ACES User guide: <https://hprc.tamu.edu/kb/User-Guides/ACES>
- Email your questions to help@hprc.tamu.edu
- See our training sessions: hprc.tamu.edu/training



Remember the Dashboard!



High Performance
Research Computing
DIVISION OF RESEARCH

<https://hprc.tamu.edu>

HPRC Helpdesk:

help@hprc.tamu.edu

Phone: 979-845-0219

Take our short course survey!



HPRC Survey

https://u.tamu.edu/hprc_shortcourse_survey

Thank you! Any Questions?

