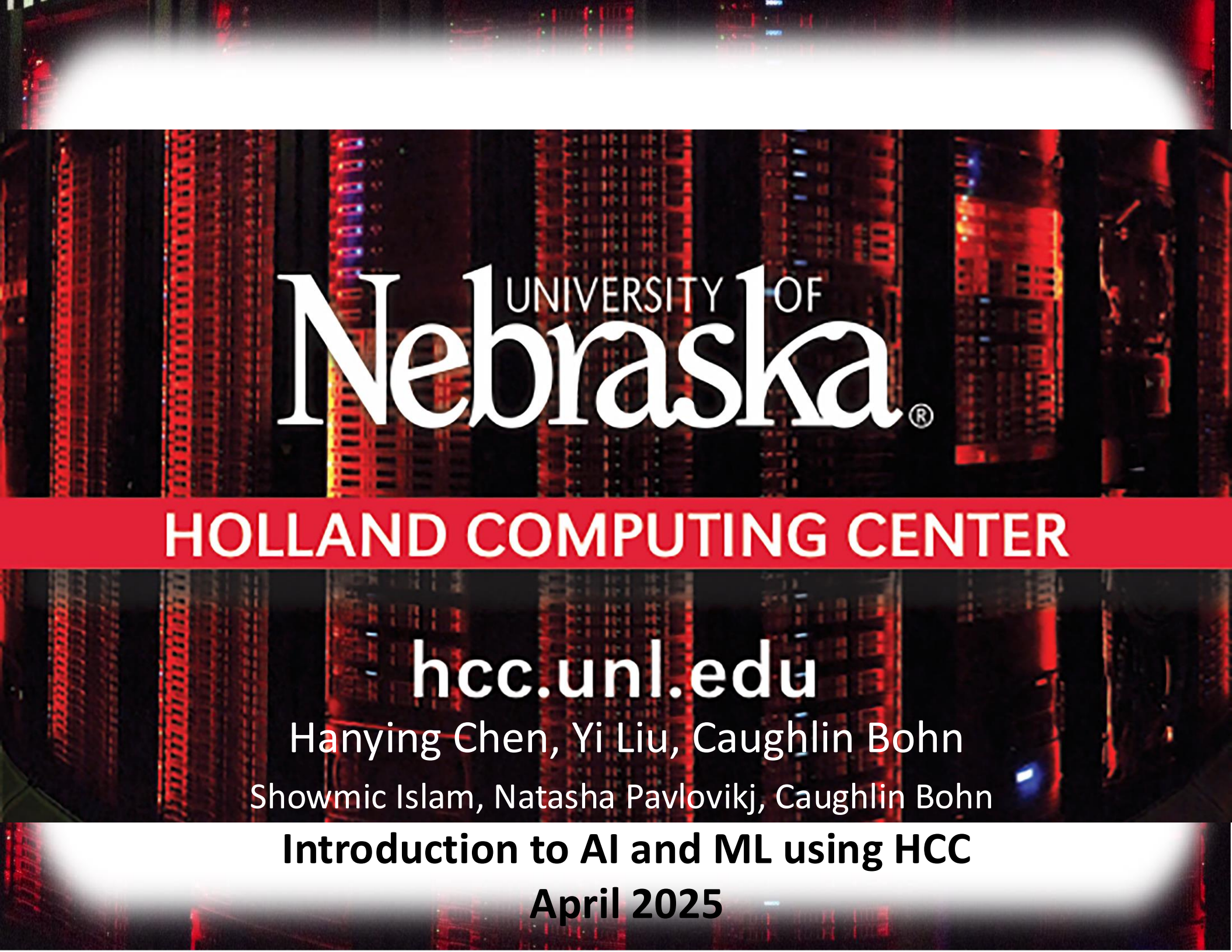


Hanying Chen, Yi Liu, Caughlin Bohn

Showmic Islam, Natasha Pavlovikj, Caughlin Bohn

Introduction to AI and ML using HCC

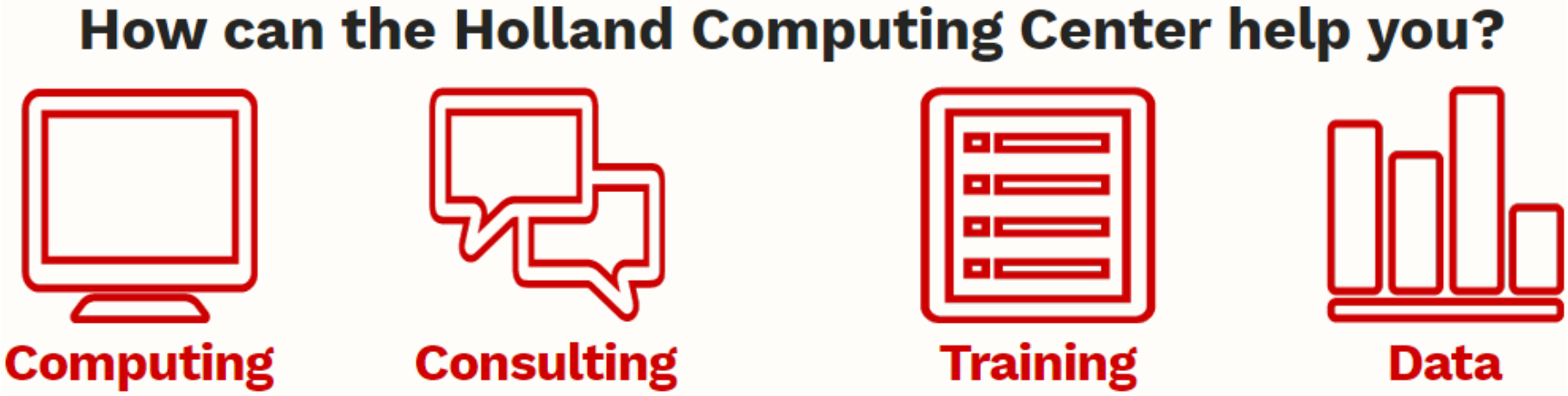
April 2025

Schedule

timestamp	Session
12:00 ~ 12:15pm	Setup and Support
12:15 ~ 1:00pm	**Introduction to HCC resources for ML and AI (45 Min)**
1:00 ~ 1:15pm	Short Break (15 Min)
1:15 ~ 2:00pm	**ML and AI workflows (45 Min)**
2:00 ~ 2:10pm	Short Break (10 Min)
2:10 ~ 2:50pm	**Introduction to PyTorch (40 Min)**
2:50 ~ 3:00pm	Break (10 Min)
3:00 ~ 3:50pm	**Introduction to PyTorch cont. (50 Min)**
3:50 ~ 4:00pm	Break (10 Min)
4:00 ~ 4:15pm	**Introduction to National Research Platform (NRP)**
4:15 ~ 4:30pm	Open Questions

Welcome!

- Available tools and software on HCC
- Requesting resources
- Monitoring model performance



Takeaways

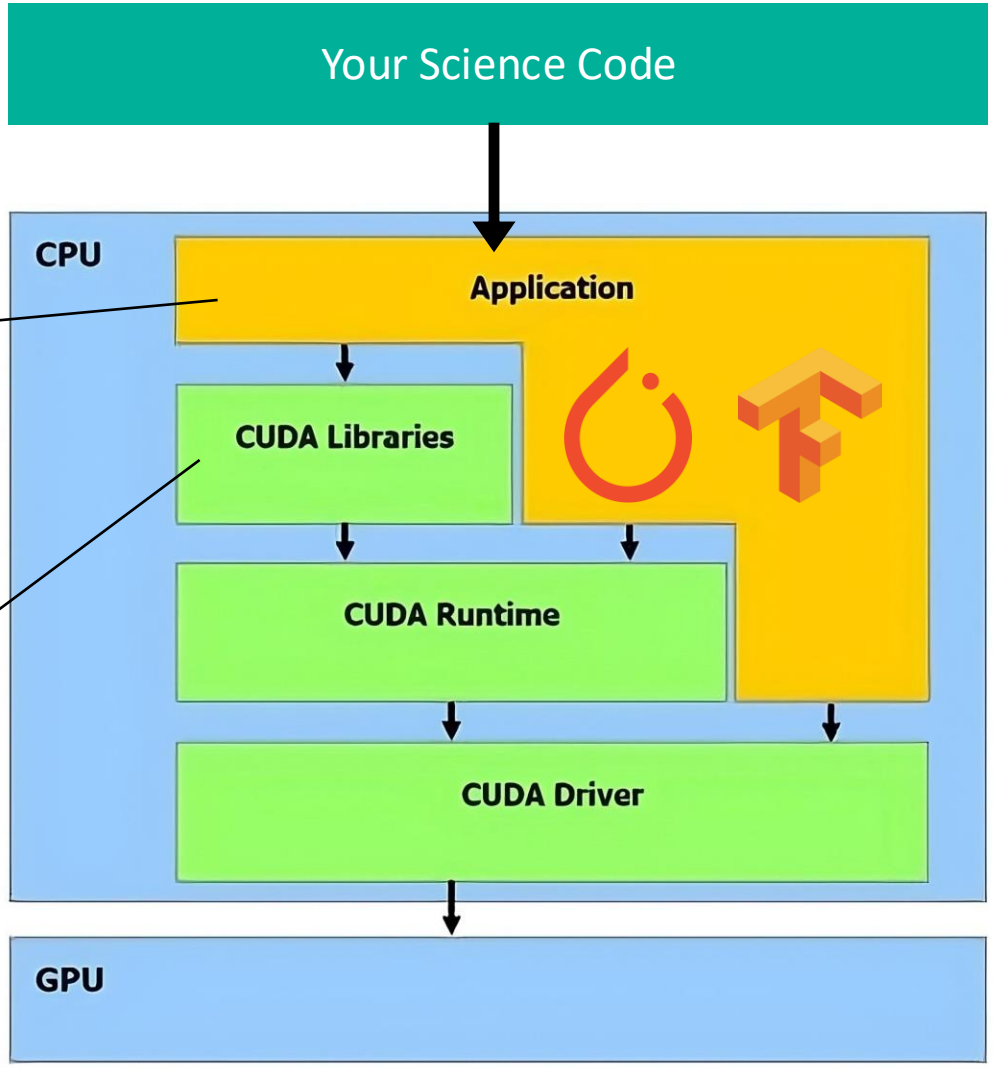
In this session, we will introduce the resources available on HCC's Swan. You will learn:

- The **available software** on Swan for machine learning and AI
- **Options to develop and run** experiments using Swan
- **Options to monitor** and manage your experiments on Swan

Available Software

GPU acceleration ready framework:
PyTorch, TensorFlow

Custom implementation:
dedicated but complicated

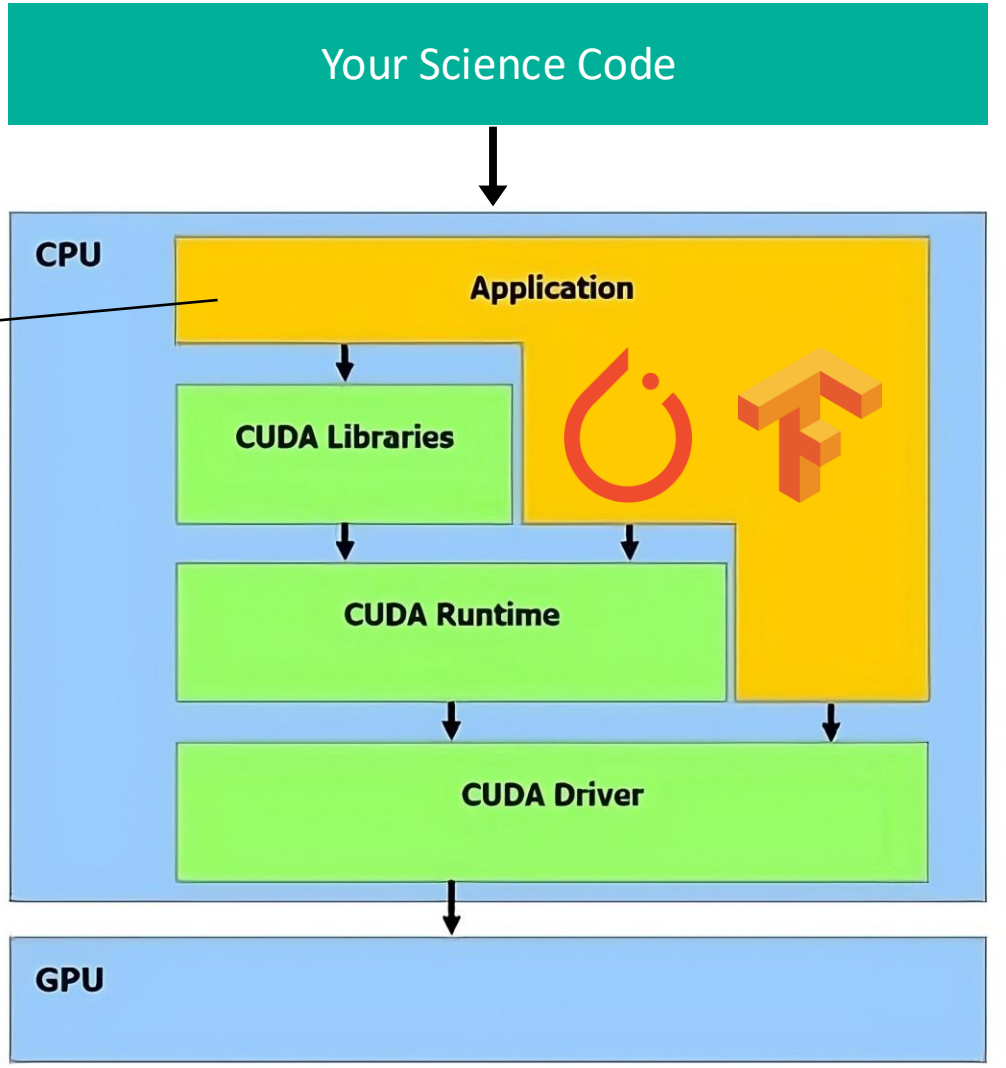


Available Software

GPU acceleration ready framework:
PyTorch, TensorFlow

PyTorch:
Fast prototyping, Ease of developing

TensorFlow:
Production-ready deployment features



Overview

- HCC also offers
  - Pure PyTorch/TensorFlow framework via **Module**
  - **Jupyter Lab** kernels for development
  - Options to create a custom **Anaconda** environment or **Docker** image to support your research
  - **LM Studio** and **Ollama** for running local Large Language Models (LLMs)


PyTorch

TensorFlow

LM Studio

Ollama



ANACONDA.

docker

Pre-Installed Software

PyTorch

Multiple PyTorch versions are installed on Swan, including both CPU and GPU versions

```
-----------------------------------------------------------
 To find other possible module matches execute:

     $ module -r spider '.*pytorch.*'

-----------------------------------------------------------
```

User-friendly web-based module search:
https://hcc.unl.edu/docs/applications/modules/available_software_for_swan/

Name	Version	Module Name	Prerequisites	Type	Domain	Description
pytorch						
pytorch-gpu	2.5.1	pytorch-gpu/py310/2.5	None	application	deep learning	Tensors and Dynamic neural networks in Python with strong GPU acceleration
pytorch-gpu	2.5.1	pytorch-gpu/py311/2.5	None	application	deep learning	Tensors and Dynamic neural networks in

Pre-Installed Software

TensorFlow

Similarly, multiple CPU/GPU TensorFlow versions are installed on Swan

```
-------------------------------------------------------
 To find other possible module matches execute:

     $ module -r spider '.*tensorflow.*'

-------------------------------------------------------
```

- Modules provide pure PyTorch/TensorFlow framework
- HCC offers
  - Create a custom Anaconda environment
  - https://hcc.unl.edu/docs/applications/user_software/using_anaconda_package_manager/#creating-custom-gpu-anaconda-environment

Pre-Installed Software

Command line access:
Load **PyTorch/TensorFlow** module

For example, to load the latest PyTorch GPU version:

```
$ module load pytorch-gpu/py312/2.5
```

To load the latest TensorFlow GPU version:

```
$ module load tensorflow-gpu/py311/2.17
```

```
[yliu95@login1.swan ~]$ module -r spider '.*pytorch.*'

----------------------------------------------------------------------------
  pytorch:
----------------------------------------------------------------------------
    Description:
      PyTorch is an optimized tensor library for deep learning using GPUs
      and CPUs.

     Versions:
        pytorch/py27/0.4
        pytorch/py27/1.2
        pytorch/py35/0.4
        pytorch/py36/0.4
        pytorch/py36/1.2
```

Running in Command Line

Submitting an interactive job is useful to:

- Test your command before submitting SLURM job
- Debug your experiment

For example:

```
$ srun --nodes=1 --ntasks-per-node=4 --mem-per-cpu=1024 --time=3:00:00 --pty
$SHELL
```

To request GPU resources

```
$ srun --nodes=1 --ntasks-per-node=4 --mem-per-cpu=1024 --time=3:00:00
--partition=gpu --gres=gpu --pty $SHELL
```

More details see:
https://hcc.unl.edu/docs/submitting_jobs/creating_an_interactive_job/

Running with SLURM

For example:

```
#!/bin/bash
#SBATCH --time=03:15:00          # Run time in hh:mm:ss
#SBATCH --mem-per-cpu=1024       # Maximum memory required per CPU (in megabytes)
#SBATCH --job-name=hello-world
#SBATCH --error=/work/[groupname]/[username]/job.%J.err
#SBATCH --output=/work/[groupname]/[username]/job.%J.out

module load example/test

hostname
sleep 60
```

Parameters to request resources

Load necessary modules

Command to run your experiments

More resource requesting command see:
https://hcc.unl.edu/docs/submitting_jobs/

Running with SLURM

To request GPU resources, two additional parameters are required:

```
--partition=gpu --gres=gpu
```

For example:

```
#!/bin/bash
#SBATCH --time=03:15:00
#SBATCH --mem-per-cpu=1024
#SBATCH --job-name=cuda
#SBATCH --partition=gpu
#SBATCH --gres=gpu
#SBATCH --error=/work/[groupname]/[username]/job.%J.err
#SBATCH --output=/work/[groupname]/[username]/job.%J.out

module load cuda
./cuda-app.exe
```

More resource requesting command see:
https://hcc.unl.edu/docs/submitting_jobs/submitting_gpu_jobs/

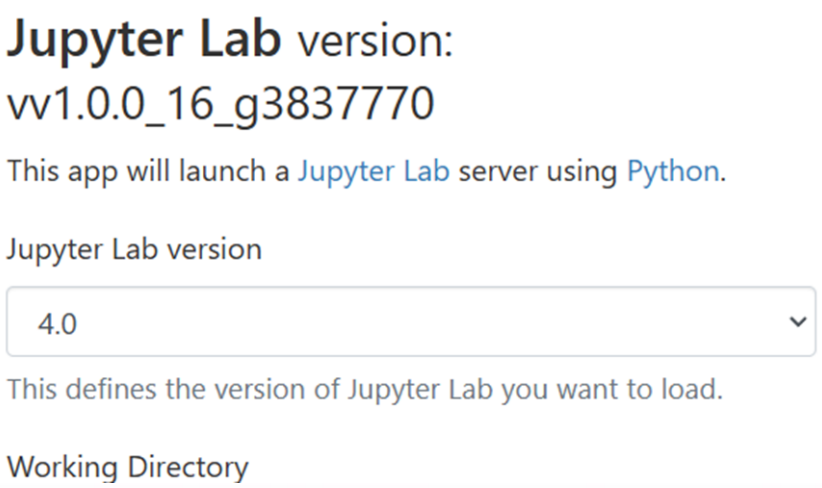
Running with Jupyter Lab

Interactive coding environments on Open OnDemand:

- Run code part-by-part and visualize results instantly
- Develop your experiments

Open OnDemand access (https://swan-ood.unl.edu/):
Interactive Apps → Jupyter Lab

- Select preferred Jupyter Lab (Ver. 3.4 or 4.0)



Command Line vs. Open OnDemand

Command line:

- Allow extended running time using SLURM (7 days maximum)
- Better for running experiments
- Submit your job to Swan using SLURM
  https://hcc.unl.edu/docs/submitting_jobs/

Jupyter Lab in Open OnDemand:

- GUI IDE
- Limited running time (8 hours maximum)
- Better for developing code
- Create interactive app using Open OnDemand
  https://hcc.unl.edu/docs/open_ondemand/virtual_desktop_and_interactive_apps/

Custom Environment

ML/AI is data-driven

- Additional package often required data processing and post-analysis
- For example:
  - OpenCV – image processing
  - NLTK – natural language processing
  - SciPy – computational analysis

HCC provides support for custom environments using:

- Anaconda – easier environment build
- Docker – more consistency across machines

Custom Environment

Anaconda:

- Start by cloning one of the pre-installed kernel
- Tutorial:

https://hcc.unl.edu/docs/applications/user_software/using_anaconda_package_manager/#creating-custom-gpu-anaconda-environment

Docker:

- Similarly, starting from a GPU-enabled Docker image on HCC's Docker hub (https://hub.docker.com/u/unlhcc/)
- Tutorial:

https://hcc.unl.edu/docs/applications/user_software/using_apptainer/

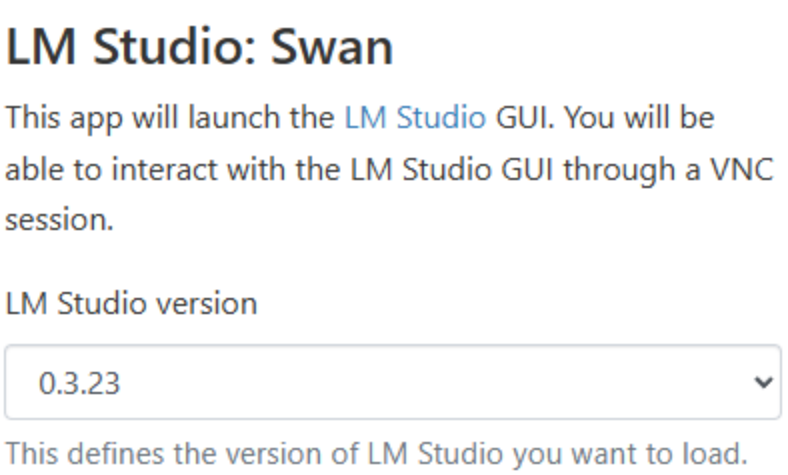
Large Language Models

Large language models (LLMs) are large models that are pre-trained on vast amounts of data. LLMs can be used on different ways on HCC resources.

CLI Tool:
Ollama and Ollama-GPU

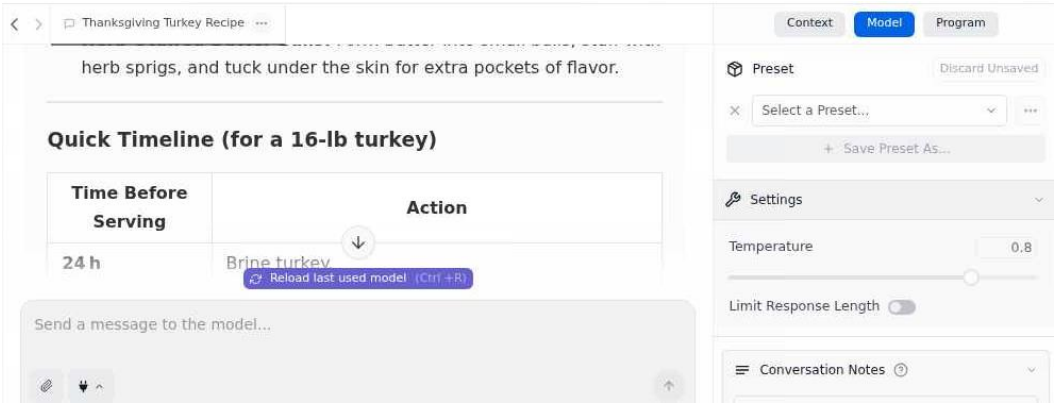
Name	Version	Module Name
ollama		
ollama-gpu	0.11.4	ollama-gpu/0.11
ollama	0.11.4	ollama/0.11

Open OnDemand Tool:
LM Studio



LM Studio

- Beginner-friendly, no-code setup with Graphical user interface (GUI)
- Ideal for:
  - Local testing and prototyping
  - GUI-based prompt engineering
  - Lightweight experiments
  - Non-developer use



Ollama

- Command-line interface (CLI) and flexible API for developers
- Ideal for:
  - Scripting and chaining models in developer workflows
  - Seamless integration into backend systems
  - Fine-tuning and customizing LLM pipelines

```python
class LocalLLM:
    def instruct_to_actions(self, instruction: str) -> Tuple[Dict[str, Any], str, ChatResponse]:
        system = (
            "[Context configurations]"
        )
        schema_hint = (
            "[Prompt instructions]"
        )
        # Prepare the prompt
        prompt = (
            f"{system}\n\n"
            f"Allowed operations: {schema_hint}\n\n"
            f"Instruction: {instruction}\n\n"
            f"JSON:"
        )
        # call the LLM via ollama pypi
        # ...
```

More info: https://hcc.unl.edu/docs/applications/app_specific/using_llm/

Monitor Resource Usage

Monitoring Resources Usage is important in optimizing machine learning and deep learning models

- Hyper-parameter decision based on memory usage
  - batch size – larger batch size needs more memory
  - training precision – higher precision (32/16/8-bit) needs more memory
  - model architecture – deeper model (AlexNet vs. ResNet) needs more memory

- Computing resource usage (%CPU/GPU)
  - long GPU idle time – large data loading overhead

Monitor Resource Usage

For more tips for monitoring the running jobs see:

Monitoring CPU Usage:
https://hcc.unl.edu/docs/submitting_jobs/monitoring_jobs/

Monitoring GPU Usage:
https://hcc.unl.edu/docs/submitting_jobs/monitoring_gpu_usage/

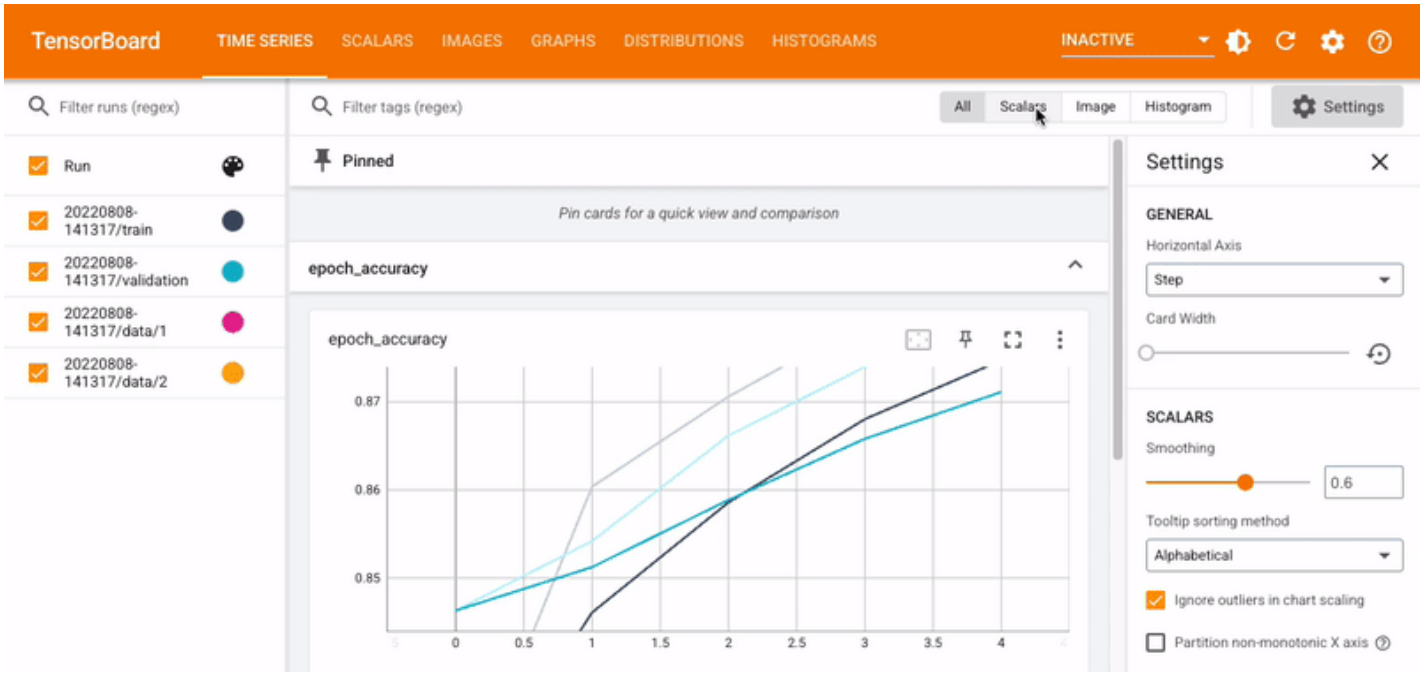
```
top - 14:22:31 up 7 days, 21:16,  0 users,  load average: 1.85, 2.04, 1.64
Tasks: 812 total,   3 running, 809 sleeping,   0 stopped,   0 zombie
%Cpu(s):  8.1 us,  1.0 sy,  0.0 ni, 89.9 id,  0.8 wa,  0.1 hi,  0.1 si,  0.0 st
MiB Mem : 256738.5 total, 247767.9 free,   4959.9 used,   4010.8 buff/cache
MiB Swap:      0.0 total,      0.0 free,      0.0 used. 250108.0 avail Mem

    PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
1736758 hccdemo   20   0 5940976   1.1g 191832 S 135.9   0.4   0:04.57 MATLAB
1736757 hccdemo   20   0 5938428   1.1g 192432 S 135.2   0.4   0:04.54 MATLAB
1736756 hccdemo   20   0 6004472   1.1g 192460 S 134.9   0.4   0:04.53 MATLAB
```

```
Every 1.0s: nvidia-smi                  c2420.swan.hcc.unl.edu: Fri Sep 27 14:28:23 2024

Fri Sep 27 14:28:23 2024
+-----------------------------------------------------------------------------------------+
| NVIDIA-SMI 550.107.02             Driver Version: 550.107.02     CUDA Version: 12.4     |
|-----------------------------------------+------------------------+----------------------+
| GPU  Name                 Persistence-M | Bus-Id          Disp.A | Volatile Uncorr. ECC |
| Fan  Temp   Perf          Pwr:Usage/Cap |           Memory-Usage | GPU-Util  Compute M. |
|                                         |                        |               MIG M. |
|=========================================+========================+======================|
|   0  Tesla V100-PCIE-32GB           On  |   00000000:5E:00.0 Off |                    0 |
| N/A   34C    P0             37W /  250W |     380MiB /  32768MiB |      4%      Default |
|                                         |                        |                  N/A |
+-----------------------------------------+------------------------+----------------------+
|   1  Tesla V100-PCIE-32GB           On  |   00000000:D8:00.0 Off |                    0 |
| N/A   51C    P0             46W /  250W |    1094MiB /  32768MiB |     27%      Default |
|                                         |                        |                  N/A |
```

Monitor Model Performance

TensorBoard is useful to analyze the model performance during or post training

- Two steps in general
  - add logging code and write to file
  - open a TensorBoard session to check



Monitor Model Performance

Add logging code and write to file

For example, write the model graph in PyTorch:

Python

```
from torch.utils.tensorboard import SummaryWriter
writer = SummaryWriter()
writer.add_graph(model, dummy_input)
```

You can add metrics recording such as loss, accuracy, recall, etc. in training loop, see more in:
https://pytorch.org/docs/stable/tensorboard.html
https://pytorch.org/tutorials/intermediate/tensorboard_tutorial.html

Monitor Model Performance

Similarly, write the model graph in TensorFlow, for example:

Python

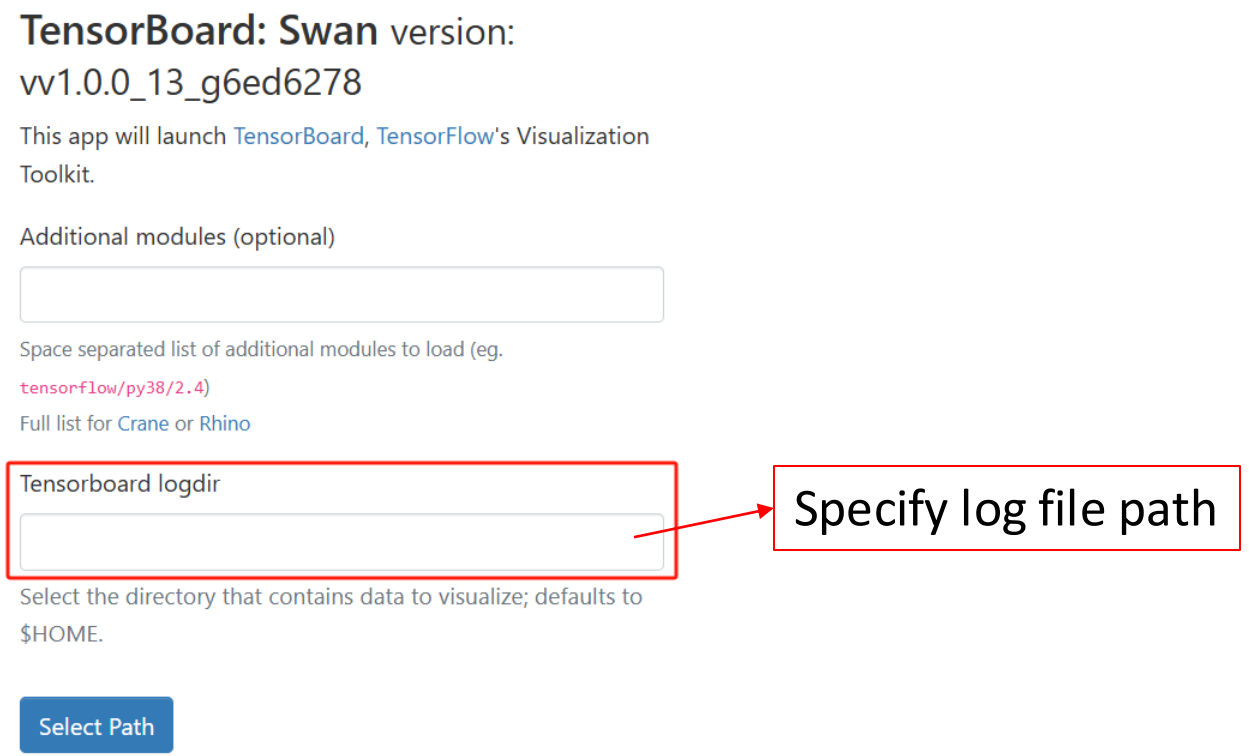
```python
writer = tf.summary.create_file_writer(log_dir)

# Trace and export the graph
tf.summary.trace_on(graph=True, profiler=False)
forward_pass(sample_input)
with writer.as_default():
    tf.summary.trace_export(
        name="model_graph",
        step=0,
        profiler_outdir=log_dir)
```

Find more tutorials on logging models at:
https://www.tensorflow.org/tensorboard/get_started

Monitor Model Performance

Open an interactive TensorBoard session to check the logged information

- Open OnDemand:
  Interactive Apps → TensorBoard



Setup for Hands on

In this session, we will use Jupyter Lab via Swan's Open OnDemand portal for hands-on practice.

1. Login to Swan's Open OnDemand portal: https://swan-ood.unl.edu/
2. Copy the practice notebook and data to your $WORK directory:
   - using terminal:
     - Select "Clusters" → "Swan Shell Access"

```
$ cd $WORK
$ git clone https://github.com/unlhcc/hcc-ai-ml-workshop-2025.git
$ ls hcc-ai-ml-workshop-2025
```

If you **are done**, please put up your yellow sticky note.

If you **need help**, please put up your red sticky note.

Setup for Hands on

3. Select “Interactive Apps” → “Jupyter Lab”
4. Navigate to the "**hcc-ai-ml-workshop-2025**" folder in left-hand sidebar
5. Open the “**00_prepare_datasets.ipynb**" notebook and run all cells

Open OnDemand Settings

Parameter	Value
Jupyter Lab version	4.0
Working Directory	/work/groupname/username
Number of cores	8
Running time in hours	3
Requested RAM in GBs	32
Partition selection	guest_gpu
Reservation	Located on the back of your name tag
GRES	gpu
Job Constraints	---Leave BLANK---

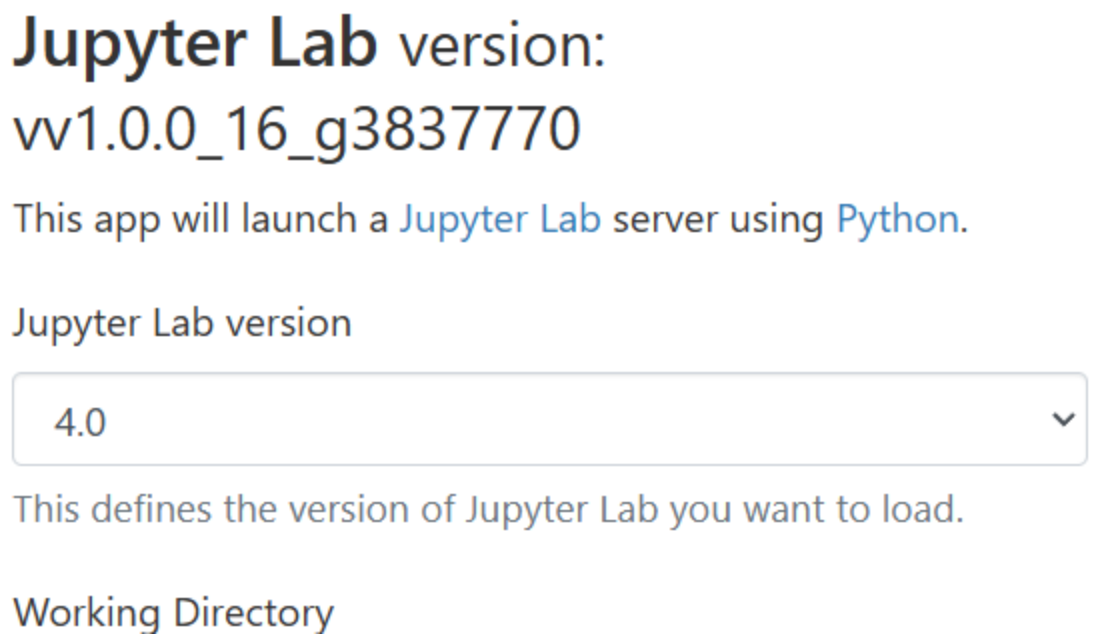
Note that, since this is an intermediate workshop, we assume that participants:

- Have basic knowledge of Python, so we will not explain Python code.
- Are familiar with using Swan, including terminal access and interactive apps.

Jupyter Lab Kernels

Open OnDemand access (https://swan-ood.unl.edu/):
Interactive Apps → Jupyter Lab

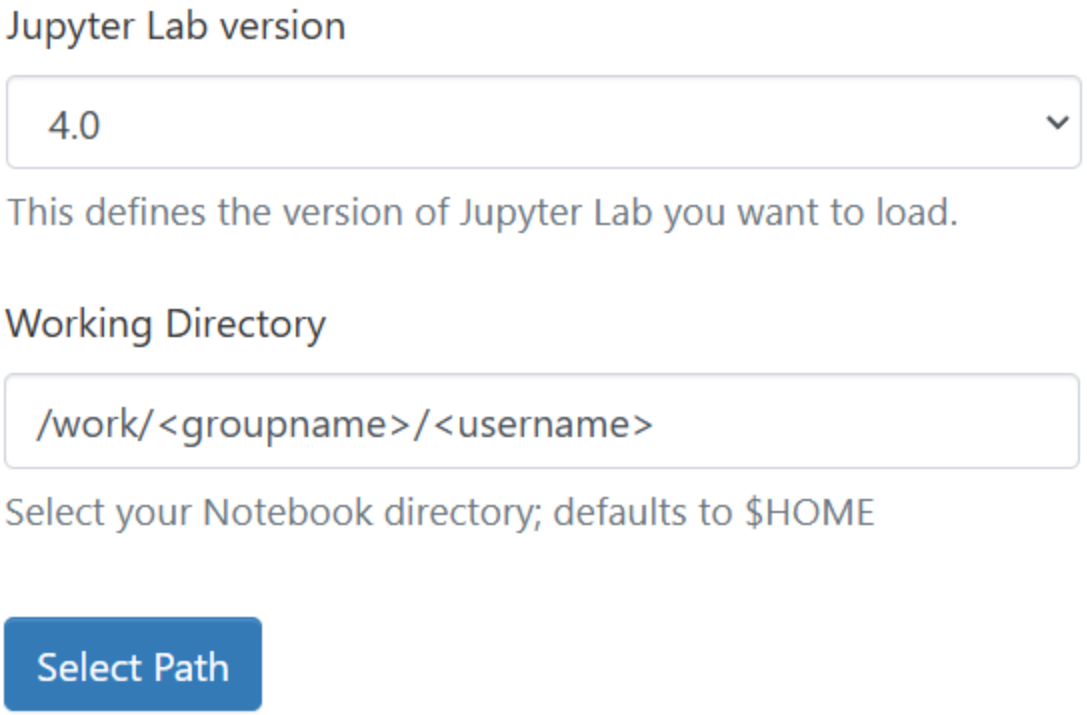
- Select preferred Jupyter Lab (Ver. 3.4 or 4.0)



Jupyter Lab Kernels

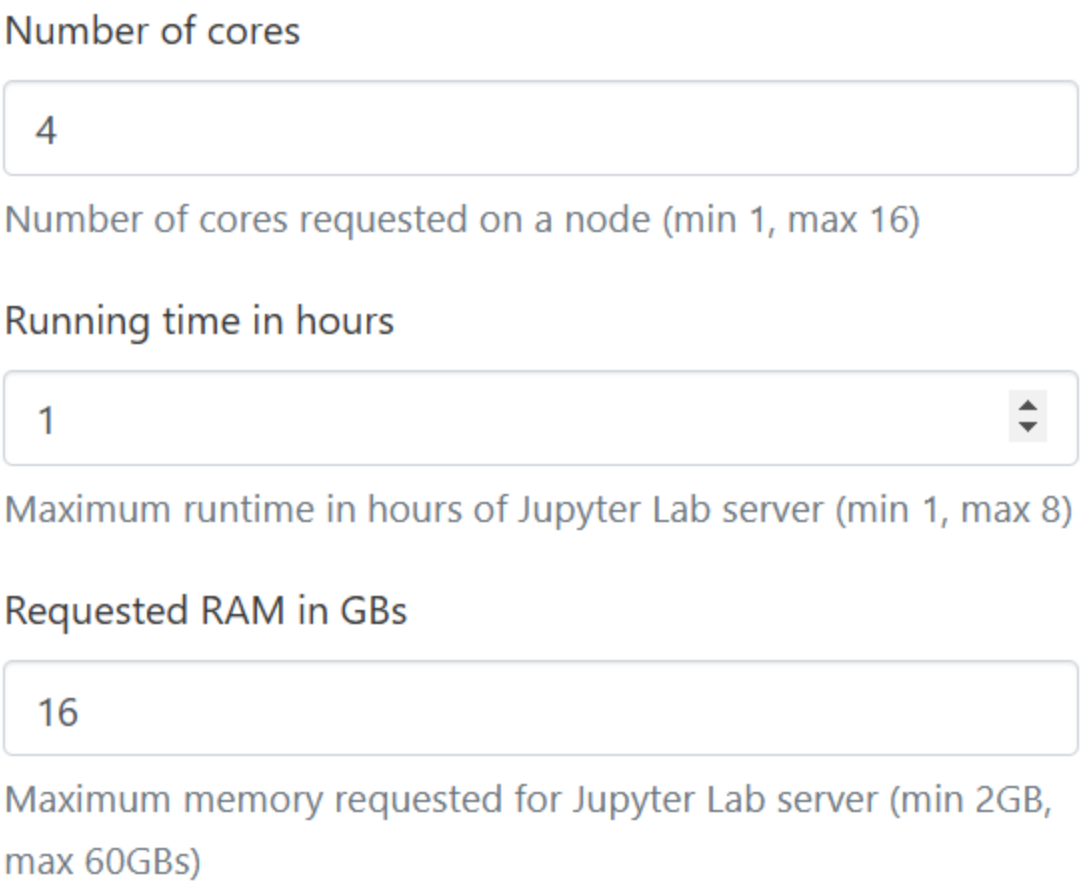
Parameters to request resources:

- Jupyter Lab version
- Path to working directory



Jupyter Lab Kernels

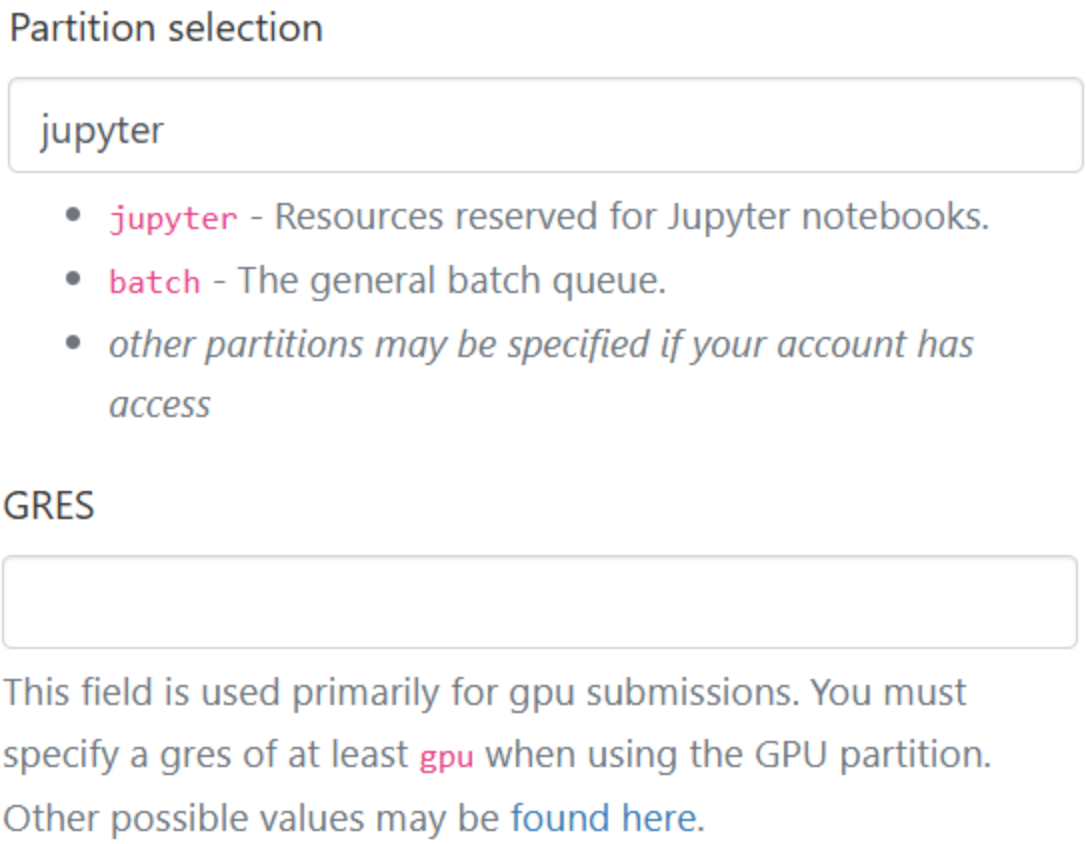
Parameters to request resources: CPU related resources



Jupyter Lab Kernels

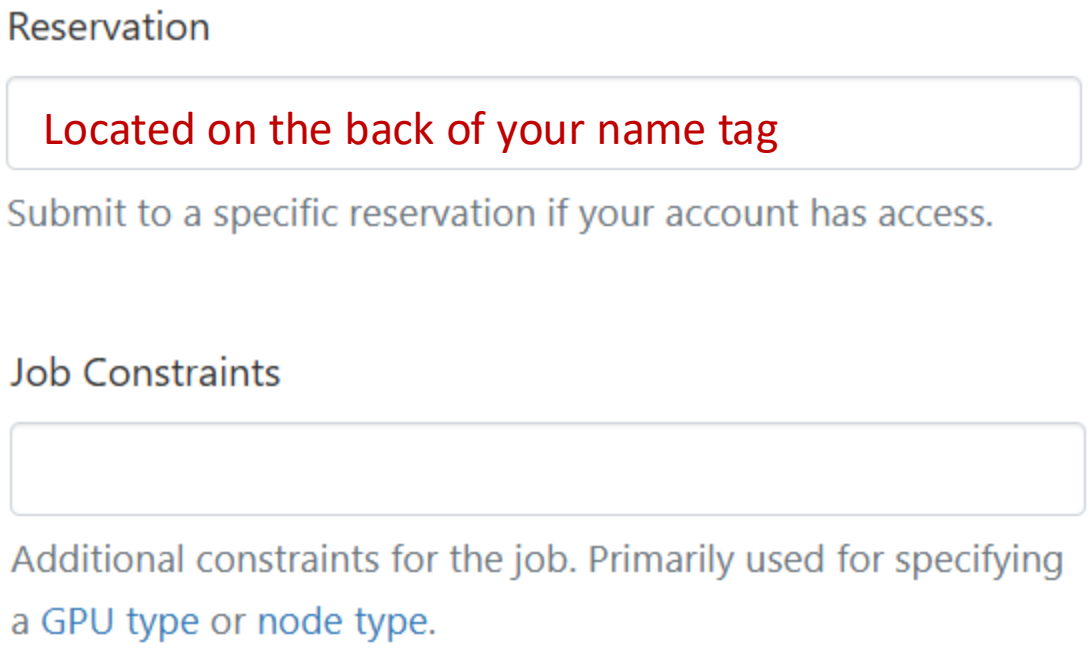
Parameters to request resources: GPU related resources

- Type “guest_gpu” in Partition selection and GRES to request GPU
- “jupyter” or “batch” for CPU resources



Jupyter Lab Kernels

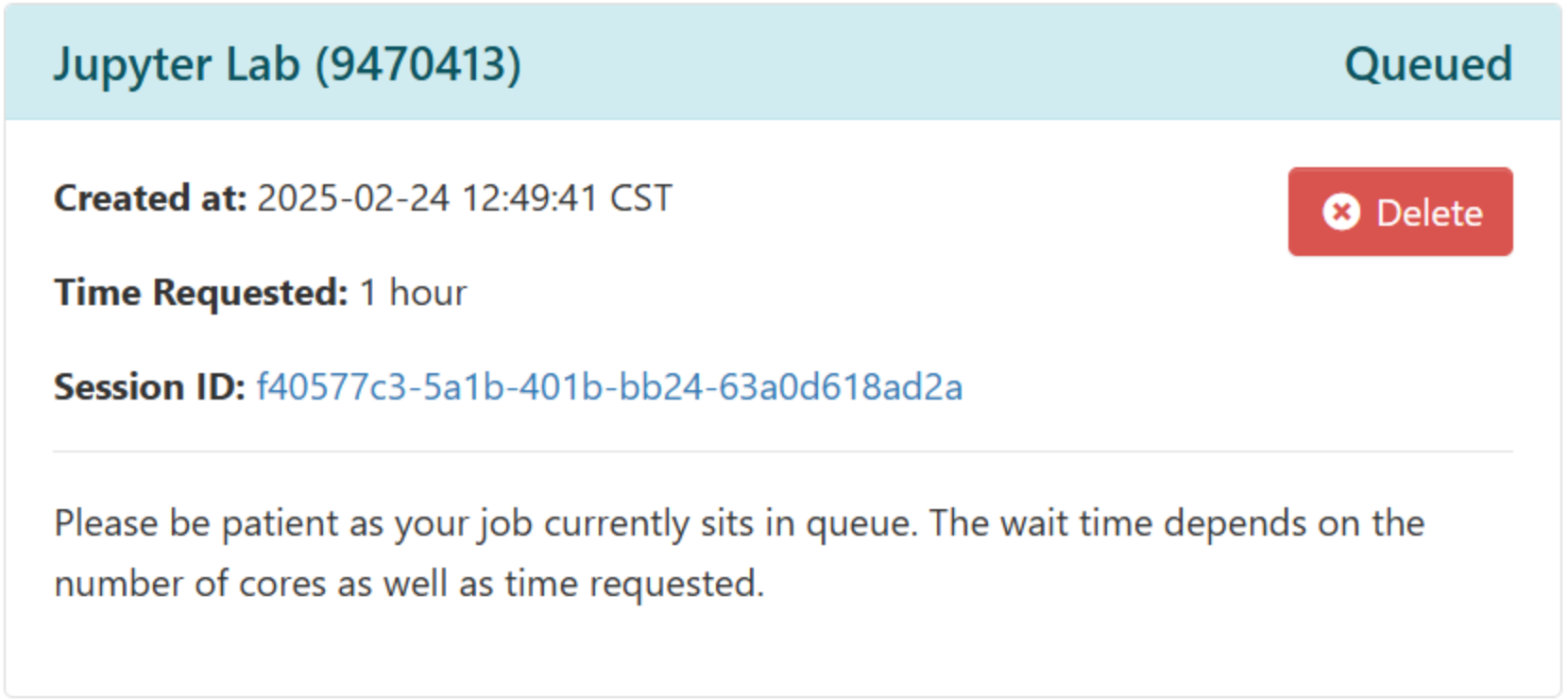
Parameters to request resources: other parameters



Jupyter Lab Kernels

Open OnDemand access:

- Wait for your Jupyter Lab session to start



Setup for Hands on

3. Select “Interactive Apps” → “Jupyter Lab”
4. Navigate to the "**hcc-ai-ml-workshop-2025**" folder in left-hand sidebar
5. Open the "**00_prepare_datasets.ipynb**" notebook and run all cells

Open OnDemand Settings

Parameter	Value
Jupyter Lab version	4.0
Working Directory	/work/groupname/username
Number of cores	8
Running time in hours	3
Requested RAM in GBs	32
Partition selection	guest_gpu
Reservation	Located on the back of your name tag
GRES	gpu
Job Constraints	---Leave BLANK---

Note that, since this is an intermediate workshop, we assume that participants:

- Have basic knowledge of Python, so we will not explain Python code.
- Are familiar with using Swan, including terminal access and interactive apps.

Schedule

timestamp	Session
12:00 ~ 12:15pm	Setup and Support
12:15 ~ 1:00pm	**Introduction to HCC resources for ML and AI (45 Min)**
1:00 ~ 1:15pm	Short Break (15 Min)
1:15 ~ 2:00pm	**ML and AI workflows (45 Min)**
2:00 ~ 2:10pm	Short Break (10 Min)
2:10 ~ 2:50pm	**Introduction to PyTorch (40 Min)**
2:50 ~ 3:00pm	Break (10 Min)
3:00 ~ 3:50pm	**Introduction to PyTorch cont. (50 Min)**
3:50 ~ 4:00pm	Break (10 Min)
4:00 ~ 4:15pm	**Introduction to National Research Platform (NRP)**
4:15 ~ 4:30pm	Open Questions