Machine Learning

- Scientific Support
- Courses
- DCSR-LLM - Toolkit for Research at UNIL
- Deep Learning with GPUs

Scientific Support

Need help with Machine Learning in your research?

Contact us at helpdesk@unil.ch with subject: DCSR ML support

Scientific support for Machine Learning projects, as outlined below, is provided free of charge to all UNIL members.

Introduction

Machine Learning provides a powerful framework for predictive modeling in scientific research:

- Infer outcomes from complex datasets using classification and regression models
- Evaluate and improve models based on predictive performance
- Use exploratory techniques to better understand and prepare your data

At DCSR, we support researchers in several key areas of Machine Learning:

Training

We help you understand how specific Machine Learning methods work and how to apply them in your research. We also offers short introductory courses on Machine Learning; see ML courses.

Methodology

We assist you in selecting and applying appropriate Machine Learning methods for your research.

This may include:

- A pilot phase, where we collaboratively develop and test code on your laptop or UNIL clusters
- A production phase, where we help scale and refine your workflow

More specifically, we can:

- Identify existing tools suited to your analysis
- Help install and run them on your laptop or UNIL clusters
- Explain key parameters and settings
- Help develop custom algorithms and code if no suitable tools exist

Infrastructure

We help you efficiently run your Machine Learning workflows on UNIL clusters.

This includes:

- Installing and configuring your code
- Profiling performance to optimize resource usage (RAM, CPUs/GPUs, number of nodes)

Collaboration at UNIL

We can connect you with relevant experts at UNIL to discuss specific Machine Learning challenges.

Example Use Cases:

1. Experimental scientist
   Wants to analyze data using Machine Learning on a laptop or UNIL clusters.
   → We help identify suitable tools, explain how they work, and support their use.

2. Data scientist (setup phase)
   Wants to implement a Machine Learning pipeline but is unsure how to proceed.
   → We help select and apply appropriate methods.

3. Data scientist (review phase)
   Has implemented a pipeline and wants feedback.
   → We review the methodology and suggest improvements or alternatives.

4. Scaling from laptop to cluster
   Wants to move a pipeline from a local computer to UNIL clusters.
   → We assist with deployment, software setup, and performance optimization.

Contact

You can reach us at helpdesk@unil.ch with subject: DCSR ML support

Courses

Here are the Machine Learning courses provided by the DCSR:

- A Gentle Introduction to Decision Trees and Random Forests with Python and R
- A Gentle Introduction to Deep Learning with Python and R
- An Introduction to Image Analysis with CNNs in Python
- An Introduction to Text Analysis with Transformers and LLMs in Python

These courses are free of charge for all UNIL members.

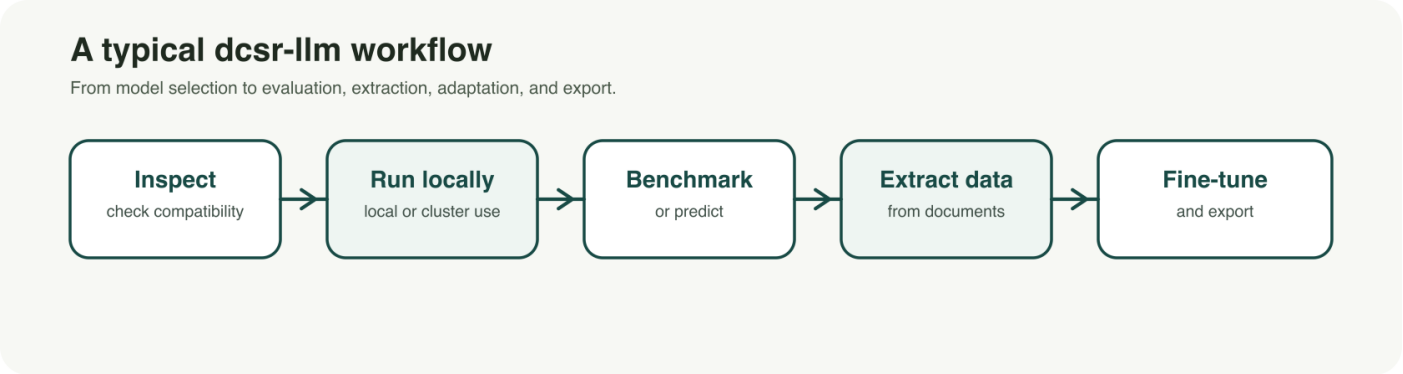
You can find the schedule and registration details here: https://courses.unil.ch/ci

For more information about these courses, please contact us at helpdesk@unil.ch with subject: DCSR ML courses

DCSR-LLM - Toolkit for Research at UNIL

Large language models are attracting growing interest across research fields, but many academic uses require more than a simple chatbot interface. Researchers often need to compare models, test them on specific tasks, extract structured information from documents, or adapt them to a domain-specific workflow. For these needs, reproducibility, local control, and transparent experimentation matter as much as convenience.

dcsr-llm was developed with that reality in mind. It is a command-line toolkit designed to support research workflows with large language models in a more controlled and reproducible way. Rather than focusing only on conversational use, it brings together several core functions in a single framework: inspecting models before use, downloading and running them locally, generating predictions, benchmarking results, extracting structured data from text corpora, fine-tuning models, and exporting them for other environments.



For UNIL researchers, the value is practical. The tool is designed to work on local machines as well as on UNIL-supported GPU environments such as the Curnagl and Urblauna clusters. This makes it possible to move beyond isolated prompting and toward more systematic workflows. A team can, for example, inspect whether a model is compatible with its infrastructure, benchmark several models on the same question set, extract targeted variables from a document collection, or fine-tune an instruction model for a specialized task or terminology.

Several use cases are especially relevant in a research context. One is model selection: before downloading large files, researchers can inspect a model and estimate whether it is suitable for their hardware and intended workflow. Another is evaluation: instead of relying on impressions, researchers can benchmark baseline, quantized, or fine-tuned models on the same dataset and compare results consistently. A third is structured extraction: dcsr-llm can transform unstructured text into validated JSON outputs, with evidence tracking and review mechanisms that are useful for corpus-based work. For more advanced projects, the toolkit also supports fine-tuning existing instruct models to better match a domain, style, or task protocol.

A key strength of dcsr-llm is that it treats LLM use as a research workflow rather than a one-off interaction. Configurations, saved artifacts, and explicit processing steps help support reproducibility and make experiments easier to document, rerun, and compare. This is particularly important in academic settings, where results need to be traceable and methods need to remain understandable.

dcsr-llm is best understood as a technical research tool rather than a one-click application. It does not replace critical judgment, and model outputs still need to be checked and validated. But for researchers who want a more rigorous and flexible way to work with LLMs, it offers a strong foundation.

UNIL members who would like to learn more, try the tool, or provide feedback can visit the dcsr-llm repository or contact us at helpdesk@unil.ch with subject: DCSR-LLM.

Repository: https://git.dcsr.unil.ch/Scientific-Computing/dcsr-llm

Deep Learning with GPUs

The training phase of your deep learning model may be very time consuming. To accelerate this process you may want to use GPUs and you will need to install the deep learning packages, such as Keras or PyTorch, properly. Here is a short documentation on how to install some well known deep learning packages in Python. If you encounter any problem during the installation or if you need to install other deep learning packages (in Python, R or other programming languages), please send an email to helpdesk@unil.ch with subject DCSR: Deep Learning package installation, and we will try to help you.

TensorFlow and Keras

We will install the TensorFlow 2's implementation of the Keras API (tf.keras); see https://keras.io/about/

To install the packages in your work directory:

```
cd /work/PATH_TO_YOUR_PROJECT
```

Log into a GPU node:

```
Sinteractive -m 4G -G 1
```

Check that the GPU is visible:

```
nvidia-smi
```

If it works properly you should see a message including an NVIDIA table. If you instead receive an error message such as "nvidia-smi: command not found" it means there is a problem.

To use TensorFlow on NVIDIA GPUs we recommend the use of NVIDIA containers including TensorFlow and its dependences such as CUDA and CuDNN that are necessary for GPU acceleration. The NVIDIA containers will also include various Python libraries and Python itself in such a way that everything is compatible with the version of TensorFlow you choose. Nevertheless, if you prefer to use the virtual environment method, please look at the instructions in the comments below.

```
module load singularityce/4.1.0


export SINGULARITY_BINDPATH="/scratch,/dcsrsoft,/users,/work,/reference"
```

We have already downloaded several versions of TensorFlow:

```
/dcsrsoft/singularity/containers/tensorflow/tensorflow-ngc-24.05-2.15.sif
/dcsrsoft/singularity/containers/tensorflow/tensorflow-ngc-24.01-2.14.sif
/dcsrsoft/singularity/containers/tensorflow/tensorflow-ngc-23.10-2.13.sif
/dcsrsoft/singularity/containers/tensorflow/tensorflow-ngc-23.07-2.12.sif
/dcsrsoft/singularity/containers/tensorflow/tensorflow-ngc-23.03-2.11.sif
/dcsrsoft/singularity/containers/tensorflow/tensorflow-ngc-22.12-2.10.sif
```

Here the last two numbers indicate the TensorFlow version, for example "tensorflow-ngc-24.05-2.15.sif" corresponds to TensorFlow version "2.15". In case you want to use another version, see the instructions in the comments below.

To run it:

```
singularity run --nv /dcsrsoft/singularity/containers/tensorflow/tensorflow-ngc-24.05-2.15.sif
```

You may receive a few error messages such as “not a valid test operator”, but this is ok and should not cause any problem. You should see a message by NVIDIA including the TensorFlow version. The prompt should now start with "Singularity>" emphasising that you are working within a singularity container.

To check that TensorFlow was properly installed:

```
Singularity> python -c 'import tensorflow; print(tensorflow.__version__)'
```

There might be a few warning messages such as "Unable to register", but this is ok, and the output should be something like "2.15.0".

To confirm that TensorFlow is using the GPU:

```
Singularity> python -c 'import tensorflow as tf; gpus =
tf.config.list_physical_devices("GPU"); print("Num GPUs Available: ", len(gpus)); print("GPUs:
", gpus)'
```

You can check the list of python libraries available:

```
Singularity> pip list
```

Notice that on top of TensorFlow several well known libraries, such as "notebook", "numpy", "pandas", "scikit-learn" and "scipy", were installed in the container. The great news here is that NVIDIA made sure that all these libraries were compatible with TensorFlow so there should not be any version incompatibilities.

If necessary you may install extra packages that your deep learning code will use. For that you should create a virtual environment. Here we will call it "venv_tensorflow_gpu", but you may choose another name:

```
Singularity> python -m venv --system-site-packages venv_tensorflow_gpu
```

Activate the virtual environment:

```
Singularity> source venv_tensorflow_gpu/bin/activate
```

To install for example "tf_keras_vis":

```
(venv_tensorflow_gpu) Singularity> pip install tf_keras_vis
```

Deactivate your virtual environment and logout from singularity and the GPU node:

```
(venv_tensorflow_gpu) Singularity> deactivate
Singularity> exit
exit
```

Comments

Reproducibility

The container version specifies all Python libraries versions, ensuring consistency across different environments. If you also use a virtual environment and want to make your installation more reproducible, you may proceed as follows:

1. Create a file called "requirements.txt" and write the package names inside. You may also specify the package versions. For example:

```
tf_keras_vis==0.8.7
```

2. Proceed as above, but instead of installing the packages individually, type

```
pip install -r requirements.txt
```

Build your own container

Go to the webpage: https://docs.nvidia.com/deeplearning/frameworks/tensorflow-release-notes/index.html

Click on the latest release, which is "TensorFlow Release 24.05" at the time we're writing this documentation, and scroll down to see the table "NVIDIA TensorFlow Container Versions". It will

show you the container versions and associated TensorFlow versions. For exemple, if you want to use TensorFlow 2.14 you could select the container 24.01.

Go to the webpage: https://catalog.ngc.nvidia.com/orgs/nvidia/containers/tensorflow/tags

Select the appropriate container, for 24.01 it is "nvcr.io/nvidia/tensorflow:24.01-tf2-py3". Do not choose any "-igpu" containers because they do not work on the UNIL clusters.

Choose a name for the container, for example "tensorflow-ngc-24.01-tf2.14.sif", and create the following file by using your favorite editor:

```
cd /scratch/username/


vi tensorflow-ngc.def
```

```
Bootstrap: docker
From: nvcr.io/nvidia/tensorflow:24.01-tf2-py3

%post
    apt-get update && apt -y upgrade
    PYTHONVERSION=$(python3 --version|cut -f2 -d\ | cut -f-2 -d.)
    apt-get install -y bash wget gzip locales virutalenv git
    sed -i '/^#.* en_.*.UTF-8 /s/^#//' /etc/locale.gen
    sed -i '/^#.* fr_.*.UTF-8 /s/^#//' /etc/locale.gen
    locale-gen
```

Note that if you choose a difference container version, you will need to replace "24.01" by the appropriate container version in the script.

You can now download the container:

```
module load singularityce/4.1.0


export SINGULARITY_DISABLE_CACHE=1


singularity build --fakeroot tensorflow-ngc-24.01-tf2.14.sif tensorflow-ngc.def


mv tensorflow-ngc-24.01-tf2.14.sif /work/PATH_TO_YOUR_PROJECT
```

That's it. You can then use it as it was explained above.

Warning: Do not log into a GPU node for building a singularity container, it will not work. But of course you will need to log into a GPU node to use it as shown below.

Use a virtual environment

Using containers is convenient because it is often difficult to install TensorFlow directly within a virtual environment. The reason is that TensorFlow has several dependencies and we must load or install the correct versions of them. Here are some instructions:

```
cd /work/PATH_TO_YOUR_PROJECT

Sinteractive -m 4G -G 1

module load python/3.10.13 tk/8.6.11 tcl/8.6.12

python -m venv venv_tensorflow_gpu

source venv_tensorflow_gpu/bin/activate

pip install tensorflow[and-cuda]==2.14.0 "numpy<2"
```

Run your deep learning code

To test your deep learning code (maximum 1h), say "my_deep_learning_code.py", you may use the interactive mode:

```
cd /PATH_TO_YOUR_CODE/

Sinteractive -m 4G -G 1

module load singularityce/4.1.0

export SINGULARITY_BINDPATH="/scratch,/dcsrsoft,/users,/work,/reference"

singularity run --nv /dcsrsoft/singularity/containers/tensorflow/tensorflow-ngc-24.05-2.15.sif

source /work/PATH_TO_YOUR_PROJECT/venv_tensorflow_gpu/bin/activate
```

Run your code:

```
python my_deep_learning_code.py
```

or copy/paste your code inside a python environment:

```
python

copy/paste your code. For example:

import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.utils import to_categorical

etc
```

Once you have finished testing your code, you must close your interactive session (by typing exit), and then run it on the cluster by using an sbatch script, say "my_sbatch_script.sh":

```
#!/bin/bash -l
#SBATCH --account your_account_id
#SBATCH --mail-type ALL
#SBATCH --mail-user firstname.surname@unil.ch

#SBATCH --chdir /scratch/username/
#SBATCH --job-name my_deep_learning_job
#SBATCH --output my_deep_learning_job.out

#SBATCH --partition gpu
#SBATCH --gres gpu:1
#SBATCH --gres-flags enforce-binding
#SBATCH --nodes 1
#SBATCH --ntasks 1
#SBATCH --cpus-per-task 1
#SBATCH --mem 10G
#SBATCH --time 01:00:00

module load singularityce/4.1.0

export SINGULARITY_BINDPATH="/scratch,/dcsrsoft,/users,/work,/reference"

# To use only singularity
export singularity_python="singularity run --nv
/dcsrsoft/singularity/containers/tensorflow/tensorflow-ngc-24.05-2.15.sif python"
```

```
# To use singularity and virtual environment
export singularity_python="singularity run --nv
/dcsrsoft/singularity/containers/tensorflow/tensorflow-ngc-24.05-2.15.sif
/work/PATH_TO_YOUR_PROJECT/venv_tensorflow_gpu/bin/python"

$singularity_python /PATH_TO_YOUR_CODE/my_deep_learning_code.py
```

To launch your job:

```
cd PATH_TO_YOUR_SBATCH_SCRIPT/

sbatch my_sbatch_script.sh
```

Remember that you should write the output files in your /scratch directory.

Multi-GPU parallelism

If you want to use a single GPU, you do not need to tell Keras to use the GPU. Indeed, if a GPU is available, Keras will use it automatically.

On the other hand, if you want to use 2 (or more) GPUs (on the same node), you need to use a special TensorFlow function, called "tf.distribute.MirroredStrategy", in your python code "my_deep_learning_code.py": see the Keras documentation https://keras.io/guides/distributed_training/ If no devices are specified in the constructor argument of the strategy then it will use all the available GPUs. If no GPUs are found, it will use the available CPUs.

This function implements single-machine multi-GPU data parallelism. It works in the following way: divide the batch data into multiple sub-batches, apply a model copy on each sub-batch, where every model copy is executed on a dedicated GPU, and finally concatenate the results (on CPU) into one big batch. For example, if your batch_size is 64 and you use 2 GPUs, then we will divide the input data into 2 sub-batches of 32 samples, process each sub-batch on one GPU, then return the full batch of 64 processed samples. This induces quasi-linear speedup.

And the sbatch script must contain the line:

```
#SBATCH --gres gpu:2
```

TensorBoard

To use TensorBoard on Curnagl, you need to modify your code as explained in https://keras.io/api/callbacks/tensorboard/ .

After your TensorBoard "logs" directory has been created, you need to proceed as follows:

```
[/scratch/pjacquet] Sinteractive -m 4G -G 1
```

```
Sinteractive is running with the following options:

--gres=gpu:1 -c 1 --mem 4G -J interactive -p interactive -t 1:00:00 --x11

salloc: Granted job allocation 2466209
salloc: Waiting for resource configuration
salloc: Nodes dnagpu001 are ready for job
```

You need to remember the GPU node's name dnagpuXXX. Here it is dnagpu001.

Then

```
[/scratch/pjacquet] module load singularityce/4.1.0

[/scratch/pjacquet] export SINGULARITY_BINDPATH="/scratch,/dcsrsoft,/users,/work,/reference"

[/scratch/pjacquet] singularity run --nv /dcsrsoft/singularity/containers/tensorflow-ngc-
24.05-2.15.sif

Singularity> source /work/PATH_TO_YOUR_PROJECT/venv_tensorflow_gpu/bin/activate

(venv_tensorflow_gpu) Singularity> ls
logs

(venv_tensorflow_gpu) Singularity> tensorboard --logdir=./logs --port=6006
```

You will see the following message:

```
Serving TensorBoard on localhost; to expose to the network, use a proxy or pass --bind_all
TensorBoard 2.6.0 at http://localhost:6006/ (Press CTRL+C to quit)
```

On your laptop, you need to type:

```
ssh -J curnagl.dcsr.unil.ch -L 6006:localhost:6006 dnagpuXXX
```

where dnagpuXXX is the GPU node's name you used to launch TensorBoard (above it was dnagpu001).

Finally, on your laptop, you may use any web browser (e.g. Chrome) to open the page http://localhost:6006 (copy/paste this link into your web browser). You should then see TensorBoard with the information located in the "logs" folder.

PyTorch

To install the packages in your work directory:

```
cd /work/PATH_TO_YOUR_PROJECT
```

Log into a GPU node:

```
Sinteractive -m 4G -G 1
```

Check that the GPU is visible:

```
nvidia-smi
```

If it works properly you should see a message including an NVIDIA table. If you instead receive an error message such as "nvidia-smi: command not found" it means there is a problem.

To use PyTorch on NVIDIA GPUs we recommend the use of NVIDIA containers including PyTorch and its dependences such as CUDA and CuDNN that are necessary for GPU acceleration. The NVIDIA containers will also include various Python libraries and Python itself in such a way that everything is compatible with the version of PyTorch you choose. Nevertheless, if you prefer to use the virtual environment method, please look at the instructions in the comments below.

```
module load singularityce/4.1.0


export SINGULARITY_BINDPATH="/scratch,/dcsrsoft,/users,/work,/reference"
```

We have already downloaded several versions of PyTorch:

```
/dcsrsoft/singularity/containers/pytorch/pytorch-ngc-24.05-2.4.sif
/dcsrsoft/singularity/containers/pytorch/pytorch-ngc-24.04-2.3.sif
/dcsrsoft/singularity/containers/pytorch/pytorch-ngc-24.01-2.2.sif
/dcsrsoft/singularity/containers/pytorch/pytorch-ngc-23.10-2.1.sif
/dcsrsoft/singularity/containers/pytorch/pytorch-ngc-23.05-2.0.sif
```

Here the last two numbers indicate the PyTorch version, for example "pytorch-ngc-24.05-2.4.sif" corresponds to PyTorch version "2.4". In case you want to use another version, see the instructions in the comments below.

To run it:

```
singularity run --nv /dcsrsoft/singularity/containers/pytorch/pytorch-ngc-24.05-2.4.sif
```

You may receive a few error messages such as “not a valid test operator”, but this is ok and should not cause any problem. You should see a message by NVIDIA including the PyTorch version. The prompt should now start with "Singularity>" emphasising that you are working within a singularity container.

To check that PyTorch was properly installed:

```
Singularity> python -c 'import torch; print(torch.__version__)'
```

There might be a few warning messages such as "Unable to register", but this is ok, and the output should be something like "2.4.0".

To confirm that PyTorch is using the GPU:

```
Singularity> python -c 'import torch; cuda_available = torch.cuda.is_available(); num_gpus =
torch.cuda.device_count(); gpus = [torch.cuda.get_device_name(i) for i in range(num_gpus)];
print("Num GPUs Available: ", num_gpus); print("GPUs: ", gpus)'
```

You can check the list of python libraries available:

```
Singularity> pip list
```

Notice that on top of PyTorch several well known libraries, such as "notebook", "numpy", "pandas", "scikit-learn" and "scipy", were installed in the container. The great news here is that NVIDIA made sure that all these libraries were compatible with PyTorch so there should not be any version incompatibilities.

If necessary you may install extra packages that your deep learning code will use. For that you should create a virtual environment. Here we will call it "venv_pytorch_gpu", but you may choose another name:

```
Singularity> python -m venv --system-site-packages venv_pytorch_gpu
```

Activate the virtual environment:

```
Singularity> source venv_pytorch_gpu/bin/activate
```

To install for example "captum":

```
(venv_pytorch_gpu) Singularity> pip install captum
```

Deactivate your virtual environment and logout from singularity and the GPU node:

```
(venv_pytorch_gpu) Singularity> deactivate
Singularity> exit
exit
```

Comments

Reproducibility

The container version specifies all Python libraries versions, ensuring consistency across different environments. If you also use a virtual environment and want to make your installation more reproducible, you may proceed as follows:

1. Create a file called "requirements.txt" and write the package names inside. You may also specify the package versions. For example:

```
captum==0.7.0
```

2. Proceed as above, but instead of installing the packages individually, type

```
pip install -r requirements.txt
```

Build your own container

Go to the webpage: https://docs.nvidia.com/deeplearning/frameworks/pytorch-release-notes/index.html

Click on the latest release, which is "PyTorch Release 24.05" at the time we're writing this documentation, and scroll down to see the table "NVIDIA PyTorch Container Versions". It will show you the container versions and associated PyTorch versions. For exemple, if you want to use PyTorch 2.4 you could select the container 24.05.

Go to the webpage: https://catalog.ngc.nvidia.com/orgs/nvidia/containers/pytorch/tags

Select the appropriate container, for 24.05 it is "nvcr.io/nvidia/pytorch:24.05-py3". Do not choose any "-igpu" containers because they do not work on the UNIL clusters.

Choose a name for the container, for example "pytorch-ngc-24.05-2.4.sif", and create the following file by using your favorite editor:

```
cd /scratch/username/


vi pytorch-ngc.def
```

```
Bootstrap: docker
From: nvcr.io/nvidia/pytorch:24.05-py3

%post
    apt-get update && apt -y upgrade
    apt-get install -y bash wget gzip locales virtualenv git
    sed -i '/^#.* en_.*.UTF-8 /s/^#//' /etc/locale.gen
    sed -i '/^#.* fr_.*.UTF-8 /s/^#//' /etc/locale.gen
    locale-gen
```

Note that if you choose a difference container version, you will need to replace "24.05" by the appropriate container version in the script.

You can now download the container:

```
module load singularityce/4.1.0

export SINGULARITY_DISABLE_CACHE=1

singularity build --fakeroot pytorch-ngc-24.05-2.4.sif pytorch-ngc.def

mv pytorch-ngc-24.05-2.4.sif /work/PATH_TO_YOUR_PROJECT
```

That's it. You can then use it as it was explained above.

Warning: Do not log into a GPU node for building a singularity container, it will not work. But of course you will need to log into a GPU node to use it as shown below.

Use a virtual environment

Using containers is convenient because it is often difficult to install PyTorch directly within a virtual environment. The reason is that PyTorch has several dependencies and we must load or install the correct versions of them. Here are some instructions:

```
cd /work/PATH_TO_YOUR_PROJECT

Sinteractive -m 4G -G 1

module load python/3.10.13 cuda/11.8.0 cudnn/8.7.0.84-11.8

python -m venv venv_pytorch_gpu
```

```
source venv_pytorch_gpu/bin/activate

pip install torch torchvision torchaudio --index-url https://download.pytorch.org/whl/cu118
```

Run your deep learning code

To test your deep learning code (maximum 1h), say "my_deep_learning_code.py", you may use the interactive mode:

```
cd /PATH_TO_YOUR_CODE/

Sinteractive -m 4G -G 1

module load singularityce/4.1.0

export SINGULARITY_BINDPATH="/scratch,/dcsrsoft,/users,/work,/reference"

singularity run --nv /dcsrsoft/singularity/containers/pytorch/pytorch-ngc-24.05-2.4.sif

source /work/PATH_TO_YOUR_PROJECT/venv_pytorch_gpu/bin/activate
```

Run your code:

```
python my_deep_learning_code.py
```

or copy/paste your code inside a python environment:

```
python

copy/paste your code
```

Once you have finished testing your code, you must close your interactive session (by typing exit), and then run it on the cluster by using an sbatch script, say "my_sbatch_script.sh":

```
#!/bin/bash -l
#SBATCH --account your_account_id
#SBATCH --mail-type ALL
#SBATCH --mail-user firstname.surname@unil.ch

#SBATCH --chdir /scratch/username/
#SBATCH --job-name my_deep_learning_job
```

```
#SBATCH --output my_deep_learning_job.out

#SBATCH --partition gpu
#SBATCH --gres gpu:1
#SBATCH --gres-flags enforce-binding
#SBATCH --nodes 1
#SBATCH --ntasks 1
#SBATCH --cpus-per-task 1
#SBATCH --mem 10G
#SBATCH --time 01:00:00

module load singularityce/4.1.0

export SINGULARITY_BINDPATH="/scratch,/dcsrsoft,/users,/work,/reference"

# To use only singularity
export singularity_python="singularity run --nv
/dcsrsoft/singularity/containers/pytorch/pytorch-ngc-24.05-2.4.sif python"

# To use singularity and virtual environment
export singularity_python="singularity run --nv
/dcsrsoft/singularity/containers/pytorch/pytorch-ngc-24.05-2.4.sif
/work/PATH_TO_YOUR_PROJECT/venv_pytorch_gpu/bin/python"

$singularity_python /PATH_TO_YOUR_CODE/my_deep_learning_code.py
```

To launch your job:

```
cd $HOME/PATH_TO_YOUR_SBATCH_SCRIPT/

sbatch my_sbatch_script.sh
```

TensorBoard

You may use TensorBoard with PyTorch by looking at the documentation

https://pytorch.org/tutorials/recipes/recipes/tensorboard_with_pytorch.html

and by adapting slightly the instructions above (see TensorBoard in TensorFlow and Keras).