

Configuration d'une station de travail Deep Learning avec Ubuntu 22.10

Il est essentiel, et c'est souvent un défi pour les passionnés d'apprentissage automatique, de mettre en place leurs propres stations de travail d'apprentissage profond. Dans ce tutoriel, on va expliquer comment mettre en place une station de travail d'apprentissage profond pour mon utilisation quotidienne avec Ubuntu 22.10. On montre comment on sépare les dépendances du framework.

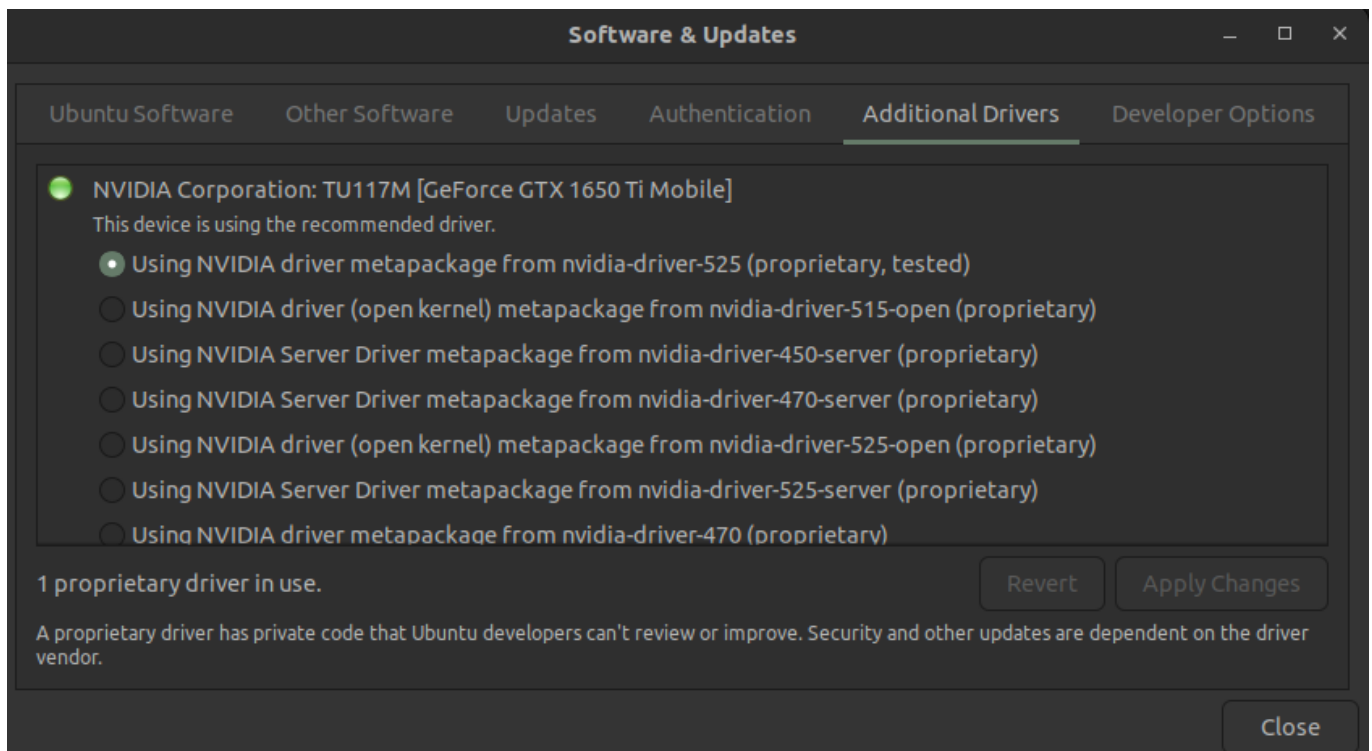
Préparation du système

On va commencer par mettre à niveau le système.

```
$ sudo apt update && sudo apt upgrade -y
```

```
$ sudo apt install git curl vim build-essential gcc-9 g++-9
```

Il n'est pas nécessaire de redémarrer Ubuntu après une mise à niveau, mais il est recommandé de le faire. Ubuntu 22.10 est livré avec un support de pilote prêt à l'emploi pour les cartes graphiques NVIDIA. Donc, on peut simplement utiliser les métapackages. Allez dans "Software & Updates" puis sélectionnez l'onglet "Additional Drivers". Cela devrait ressembler à ce qui suit,



Installation de CUDA 11.7

On va suivre la [documentation](#).

CUDA Toolkit 11.7 Update 1 Downloads

[Home](#)

Select Target Platform

Click on the green buttons that describe your target platform. Only supported platforms will be shown. By downloading and using the software, you agree to fully comply with the terms and conditions of the [CUDA EULA](#).

Operating System

Linux

Windows

Architecture

x86_64

ppc64le

arm64-sbsa

Distribution

CentOS

Debian

Fedora

OpenSUSE

RHEL

Rocky

SLES

Ubuntu

WSL-Ubuntu

Version

18.04

20.04

22.04

Installer Type

deb (local)

deb (network)

runfile (local)

Download Installer for Linux Ubuntu 22.04 x86_64

```
$ wget https://developer.download.nvidia.com/compute/cuda/repos/ubuntu2204/x86_64/cuda-ubuntu2204.pin
```

```
$ sudo mv cuda-ubuntu2204.pin /etc/apt/preferences.d/cuda-repository-pin-600
```

```
$ wget https://developer.download.nvidia.com/compute/cuda/11.7.1/local_installers/cuda-repo-ubuntu2204-11-7-local_11.7.1-515.65.01-1_amd64.deb
```

```
$ sudo dpkg -i cuda-repo-ubuntu2204-11-7-local_11.7.1-515.65.01-1_amd64.deb
```

```
$ sudo cp /var/cuda-repo-ubuntu2204-11-7-local/cuda-*-keyring.gpg /usr/share/keyrings/
```

```
$ sudo apt-get update
```

```
$ sudo apt-get -y install cuda
```

Maintenant, on met à jour les variables d'environnement, et ajoutez les lignes suivantes à `~/.bashrc`

```
export PATH=/usr/local/cuda-11.7/bin${PATH:+:${PATH}}
```

```
export LD_LIBRARY_PATH=/usr/local/cuda-11.7/lib64${LD_LIBRARY_PATH:+:${LD_LIBRARY_PATH}}
```

```
export CUDA_HOME=/usr/local/cuda
```

```
export LD_LIBRARY_PATH=/usr/local/cuda/lib64:$LD_LIBRARY_PATH
```

Puis, on les active

```
$ source ~/.bashrc
```

Installation de CUDNN 8.7

Pour cela, on devra télécharger le fichier archivé en se connectant sur le site, [cuDNN](#).

[Home](#)

cuDNN Download

NVIDIA cuDNN is a GPU-accelerated library of primitives for deep neural networks.

☒ I Agree To the Terms of the [cuDNN Software License Agreement](#)

Note: Please refer to the [Installation Guide](#) for release prerequisites, including supported GPU architectures and compute capabilities, before downloading.

For more information, refer to the cuDNN Developer Guide, Installation Guide and Release Notes on the [Deep Learning SDK Documentation](#) web page.

[Download cuDNN v8.7.0 \(November 28th, 2022\), for CUDA 11.x](#)

Local Installers for Windows and Linux, Ubuntu(x86_64, armsbsa)

[Local Installer for Windows \(Zip\)](#)

[Local Installer for Linux x86_64 \(Tar\)](#)

[Local Installer for Linux PPC \(Tar\)](#)

[Local Installer for Linux SBSA \(Tar\)](#)

[Local Installer for Debian 11 \(Deb\)](#)

[Local Installer for Ubuntu20.04 x86_64 \(Deb\)](#)

[Local Installer for Ubuntu22.04 x86_64 \(Deb\)](#)

[Local Installer for Ubuntu20.04 aarch64sbsa \(Deb\)](#)

[Local Installer for Ubuntu22.04 aarch64sbsa \(Deb\)](#)

[Local Installer for Ubuntu20.04 cross-sbsa \(Deb\)](#)

[Local Installer for Ubuntu22.04 cross-sbsa \(Deb\)](#)

Télécharger le fichier **Local installer for Linux x86_64 (Tar)**

```
$ tar -zxvf cudnn-linux-x86_64-8.7.0.84_cuda11-archive.tar.xz
```

```
$ sudo cp -P cuda/include/cudnn.h /usr/local/cuda-11.7/include
```

```
$ sudo cp -P cuda/lib/libcudnn* /usr/local/cuda-11.7/lib64/
```

```
$ sudo chmod a+r /usr/local/cuda-11.2/lib64/libcudnn*
```

Vérifier l'installation avec

```
$ nvcc --version
```

```
(base) bassem@bassem:~$ nvcc --version
nvcc: NVIDIA (R) Cuda compiler driver
Copyright (c) 2005-2022 NVIDIA Corporation
Built on Wed_Jun__8_16:49:14_PDT_2022
Cuda compilation tools, release 11.7, V11.7.99
Build cuda_11.7.r11.7/compiler.31442593_0
(base) bassem@bassem:~$
```

Installation de TensorFlow 2.11

On commence par créer un environnement conda

```
$ conda create -n tfenv
```

```
$ conda activate tfenv
```

Puis, on installe tensorflow

```
$ pip install tensorflow-gpu
```

```
$ pip install -n tfenv ipykernel
```

```
$ pip install jupyter notebook
```

Pour tester, on peut utiliser

```
$ import tensorflow as tf
```

```
$ print(tf.__version__)
```

```
$ print(tf.test.is_gpu_available())
```

```
$ print(tf.test.is_built_with_cuda())
```

```
$ print(tf.test.gpu_device_name())
```

```
$ from tensorflow.python.client import device_lib
```

```
$ print(device_lib.list_local_devices())
```

```

+ Code + Markdown | ▶ Run All | ⌵ Clear Outputs of All Cells | ⌲ Restart | 📄 Variables | 📄 Outline | ...
tfenv (Python 3.10.8)

[4] ✓ 0.8s Python
... True

[5] ✓ 0.1s Python
... /device:GPU:0

2023-01-14 07:54:50.714919: I tensorflow/compiler/xla/stream_executor/cuda/cuda_gpu_executor.cc:981] successful NUMA node read from SysFS had negative value (-1), but there must be at least one NUMA node, so returning NUMA node zero
2023-01-14 07:54:50.715418: I tensorflow/compiler/xla/stream_executor/cuda/cuda_gpu_executor.cc:981] successful NUMA node read from SysFS had negative value (-1), but there must be at least one NUMA node, so returning NUMA node zero
2023-01-14 07:54:50.715737: I tensorflow/compiler/xla/stream_executor/cuda/cuda_gpu_executor.cc:981] successful NUMA node read from SysFS had negative value (-1), but there must be at least one NUMA node, so returning NUMA node zero
2023-01-14 07:54:50.716145: I tensorflow/compiler/xla/stream_executor/cuda/cuda_gpu_executor.cc:981] successful NUMA node read from SysFS had negative value (-1), but there must be at least one NUMA node, so returning NUMA node zero
2023-01-14 07:54:50.716479: I tensorflow/compiler/xla/stream_executor/cuda/cuda_gpu_executor.cc:981] successful NUMA node read from SysFS had negative value (-1), but there must be at least one NUMA node, so returning NUMA node zero
2023-01-14 07:54:50.716785: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1613] Created device /device:GPU:0 with 2624 MB memory: -> device: 0, name: NVIDIA GeForce GTX 1650 Ti, pci bus id: 0000:01:00.0, compute capability: 7.5

```

Installation de PyTorch 1.13

On commence par créer un environnement conda

```
$ conda create -n torchenv
```

```
$ conda activate torchenv
```

Puis, on installe pytorch (voir [PyTorch](#))

```
$ conda install pytorch torchvision torchaudio pytorch-cuda=11.7 -c pytorch -c nvidia
```

```
$ pip install -n torchenv ipykernel
```

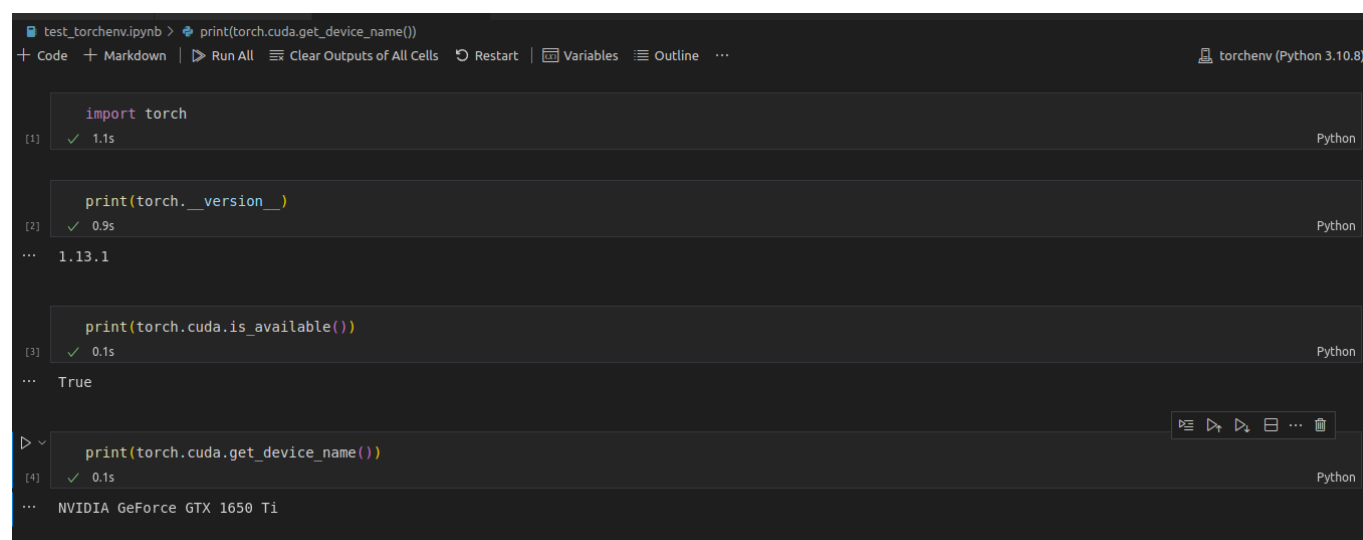
Pour tester, on peut utiliser

```
$ import torch
```

```
$ print(torch.__version__)
```

```
$ print(torch.cuda.is_available())
```

```
$ print(torch.cuda.get_device_name())
```



The screenshot shows a Jupyter Notebook titled 'test_torchenv.ipynb' with the following code cells and outputs:

- Cell 1: `import torch` (1.1s)
- Cell 2: `print(torch.__version__)` (0.9s) → Output: `1.13.1`
- Cell 3: `print(torch.cuda.is_available())` (0.1s) → Output: `True`
- Cell 4: `print(torch.cuda.get_device_name())` (0.1s) → Output: `NVIDIA GeForce GTX 1650 Ti`

The interface includes a top bar with 'Code', 'Markdown', 'Run All', 'Clear Outputs of All Cells', 'Restart', 'Variables', and 'Outline' tabs. The bottom right corner shows 'torchenv (Python 3.10.8)'.