



La science pour la santé _____
_____ From science to health

Guide d'utilisation

**Mise en œuvre de vLLM avec 2 GPUs
H100 sur le cluster HPC GPU**

INSERM - Cloud self-service - Offre HPC

Objectifs du document

Ce tutoriel vise à vous guider pas-à-pas pour :

1

Configurer un environnement Python isolé sur le cluster HPC

2

Installer des librairies au travers d'un serveur Nexus faisant office de proxy

3

Exécuter vLLM en tirant parti de 2 GPUs H100

4

Mettre en oeuvre les calculs depuis une console Linux, une interface graphique (GUI Linux) ou JupyterHub

Table des matières

1

Éléments contextuels

2

Création d'un environnement virtuel Python

3

Mise en œuvre des calculs via une console Linux

4

Mise en œuvre des calculs via une GUI Linux

5

Mise en œuvre des calculs via jupyterhub

1 Éléments contextuels

Mise en œuvre de vLLM avec 2 GPUs H100 sur le cluster HPC GPU

CONTEXTE

Les LLM (Large Language Models) sont des **modèles d'intelligence artificielle très volumineux qui nécessitent beaucoup de puissance de calcul et de mémoire**, en particulier lors de l'inférence ou de l'adaptation de modèles. Le HPC permet d'**exploiter des GPU performants et de répartir les calculs sur plusieurs ressources**, ce qui réduit fortement les temps d'exécution, notamment pour des usages d'intelligence artificielle et de traitement de données scientifiques.

Parmi les possibilités de mise en œuvre de LLM, 2 solutions peuvent être mises en œuvre : Ollama et vLLM.

Dans ce tutoriel, nous aborderons la solution vLLM.

EXEMPLES DE CAS D'USAGE VISÉS

- Tests et validation de modèles LLM
- Expérimentation GPU multi-cartes
- Prototypage et développement de workflows IA
- Évaluation de performances avant déploiement applicatif
- Utilisation interactive (notebooks) ou batch (scripts)

Ressources de calcul

La plateforme HPC propose divers types de ressources GPU. Pour les cas d'usage VLLM, nous utiliserons la ressource ci-dessous :

Partition 2xH100-96

Composée de 2 nœuds de calculs fournissant chacun :

- 32 Cores CPU
- 2 GPUs H100 – 96Go

Dédiée en intégralité à l'utilisateur qui a réservé le nœud.

L'accès et la réservation de nœuds de calcul d'une partition se fait au travers de l'orchestrateur Slurm et de ses commandes.

2 Création d'un environnement virtuel Python

Contraintes de développement Python

Python 3.9

Version par défaut sur les nœuds de calcul HPC

Python 3.10 & 3.11

Disponibles au travers d'environnement virtuel python

L'environnement HPC étant restreint, les utilisateurs peuvent télécharger et installer des librairies au travers d'un serveur Nexus faisant office de proxy. L'utilisation de ce serveur est conditionnée par un mécanisme d'authentification.

Bonnes pratiques de développement Python

Depuis home directory, nous allons créer un environnement virtuel. Nous pouvons créer autant d'environnement que nécessaire (en principe un par projet).

Un environnement virtuel Python sert à isoler les dépendances d'un projet Python. Il crée un espace indépendant avec :

- Une version dédiée de Python
- Un dossier distinct pour installer des bibliothèques spécifiques au projet

Cela permet de :

Éviter les conflits de dépendances

Ne pas polluer le système global

Gérer proprement les dépendances

Tester facilement sur différentes versions de bibliothèque

1) Cliquer sur le menu Clusters > HPC Shell Access pour accéder au home directory depuis la plateforme OOD HPC

Inserm Portail HPC GitLab Apps Files Jobs Clusters Interactive Apps My Interactive Sessions Help Logged in as hpc_admin Log Out

>_ HPC Shell Access

Inserm

La science pour la santé
From science to health

OnDemand provides an integrated, single access point for all of your HPC resources.

Pinned Apps A featured subset of [all available apps](#)

Interactive Apps

```
Host: localhost Themes: Default  
saucouturier@localhost's password:  
Last login: Wed Jan 21 08:29:33 2026 from ::1  
[saucouturier@ood ~]$
```

2) Exécuter la commande ci-dessous depuis la console pour configurer l'accès à un serveur Nexus afin de faciliter l'installation de bibliothèques Python et R :

```
$>/shared/nexus-config.py
```


- a. Saisir le nom utilisateur de votre compte ADS et son mot de passe associé
- b. Taper y à la question "voulez-vous utiliser .bashrc pour le setup python ?"


```
[saucouturier@ood ~]$ /shared/nexus-config.py
```

```
=====
```

```
Configuration Nexus Repository  
(Python pip + R CRAN)
```

```
=====
```

```
 Utilisateur AD ADS: saucouturier
```

```
 Mot de passe AD ADS:
```

```
Voulez vous Utiliser .bashrc pour le setup python ? [y/n]: y
```

La configuration se déroule :

```
✓ /home/saucouturier/.Renviron supprimé
✓ /home/saucouturier/.Rprofile supprimé

✓ Configuration Nexus supprimée

Python/pip configuré:
• PIP_INDEX_URL définie (session courante)
• Repository: https://nexus.hds.inserm.fr/repository/pypi-group/

R/CRAN configuré:
• /home/saucouturier/.Renviron (variables d'environnement)
• /home/saucouturier/.Rprofile (configuration repos)
• Repository: https://nexus.hds.inserm.fr/repository/cran-group/

====
✓ Configuration Nexus terminée
====

Terminal Shell: rechargez la configuration: source ~/.bashrc
Python: pip utilisera automatiquement Nexus
R: Redémarrez le kernel R pour appliquer la config

💡 Commandes utiles:
• pip install <package> - Installer un package Python
• install.packages('pkg') - Installer un package R (kernel R)
• test_pip() - Tester la connexion pip
• test_r() - Tester la connexion R
• show_status() - Afficher l'état de la config
• clear_config() - Supprimer la configuration

⚠ Kernel R: sous jupyter, Redémarrez- votre kernel R pour appliquer la configuration
⚠ Python : Rechargez la configuration Shell : source ~/.bashrc
```

c. Recharger la configuration Shell pour utiliser le nouveau paramétrage :

```
[saucouturier@ood ~]$ source ~/.bashrc
[saucouturier@ood ~]$
```

Dans le cadre de ce document, nous allons créer un environnement virtuel que nous allons appeler « vllm » et utilisant python 3.11. Cet environnement permettra d'installer les bibliothèques python d'un projet « VLLM ».

3) Exécuter la commande ci-dessous depuis la console :

```
$>/shared/Packages/python/3.11/bin/python3.11 -m venv /home/$USER/vllm
```

- /shared/Packages/python/3.11/bin/python3.11 : indique que nous allons utiliser la version 3.11 de python mise à disposition par le HPC via le répertoire /shared/packages/python/3.11
- vllm : nom de l'environnement virtuel python
- /home/\$USER : préciser le chemin de votre home directory afin que notre venv puisse être utilisé dans le futur avec jupyterhub

4) Exécuter la commande ci-dessous pour activer le venv et vérifier la version de python3 active :

```
$> source vllm/bin/activate  
$> python3 --version
```



```
Host: localhost Themes: Default  
[saucoturier@ood ~]$ python3 --version  
Python 3.9.18  
[saucoturier@ood ~]$ /shared/Packages/python/3.11/bin/python3.11 -m venv /home/$USER/vllm  
[saucoturier@ood ~]$ source vllm/bin/activate  
(vllm) [saucoturier@ood ~]$ python3 --version  
Python 3.11.12  
(vllm) [saucoturier@ood ~]$
```

A noter : la version python 3.11 versus 3.9 sans venv activé.

5) Exécuter la commande ci-dessous pour installer les bibliothèques nécessaires à jupyterhub :

Afin de pouvoir facilement utiliser notre venv depuis jupyterhub, nous devons ajouter les bibliothèques nécessaires : ipykernel ipywidgets

```
$> pip3 install ipykernel ipywidgets
```

```
Host: localhost Themes: Default
(vllm) [saucouturier@ood ~]$ pip3 install ipykernel ipywidgets
Looking in indexes: https://saucouturier:****@nexus.hds.inserm.fr/repository/pypi-group/simple
Collecting ipykernel
  Using cached https://nexus.hds.inserm.fr/repository/pypi-group/packages/ipykernel/7.1.0/ipykernel-7.1.0-py3-none-any.whl (117 kB)
Collecting ipywidgets
  Using cached https://nexus.hds.inserm.fr/repository/pypi-group/packages/ipywidgets/8.1.8/ipywidgets-8.1.8-py3-none-any.whl (139 kB)
Collecting comm>=0.1.1 (from ipykernel)
  Using cached https://nexus.hds.inserm.fr/repository/pypi-group/packages/comm/0.2.3/comm-0.2.3-py3-none-any.whl (7.3 kB)
Collecting debugpy>=1.6.5 (from ipykernel)
  Using cached https://nexus.hds.inserm.fr/repository/pypi-group/packages/debugpy/1.8.19/debugpy-1.8.19-cp311-cp311-manylinux_2_34_x86_64.whl (3.2 MB)
Collecting ipython>=7.23.1 (from ipykernel)
  Using cached https://nexus.hds.inserm.fr/repository/pypi-group/packages/ipython/7.23.1/ipython-7.23.1-py3-none-any.whl (433 kB)
Collecting executing>=1.2.0 (from stack_data>=0.6.0->ipython>=7.23.1->ipykernel)
  Using cached https://nexus.hds.inserm.fr/repository/pypi-group/packages/executing/2.2.1/executing-2.2.1-py2.py3-none-any.whl (28 kB)
Collecting asttokens>=2.1.0 (from stack_data>=0.6.0->ipython>=7.23.1->ipykernel)
  Using cached https://nexus.hds.inserm.fr/repository/pypi-group/packages/asttokens/3.0.1/asttokens-3.0.1-py3-none-any.whl (27 kB)
Collecting pure-eval (from stack_data>=0.6.0->ipython>=7.23.1->ipykernel)
  Using cached https://nexus.hds.inserm.fr/repository/pypi-group/packages/pure-eval/0.2.3/pure_eval-0.2.3-py3-none-any.whl (11 kB)
Installing collected packages: pure-eval, ptyprocess, widgetsnbextension, wcwidth, typing_extensions, traitlets, tornado, six, pyzmq, pygments, psutil, platformdirs, pexpect, parso, packaging, nest-asyncio, jupyterlab_widgets, executing, decorator, debugpy, comm, asttokens, stack_data, python-dateutil, prompt_toolkit, matplotlib-inline, jupyter-core, jedi, ipython-pygments-lexers, jupyter-client, ipython, ipywidgets, ipykernel
Successfully installed asttokens-3.0.1 comm-0.2.3 debugpy-1.8.19 decorator-5.2.1 executing-2.2.1 ipykernel-7.1.0 ipython-9.9.0 ipython-pygments-lexers-1.1.1 ipywidgets-8.1.8 jedi-0.19.2 jupyter-client-8.8.0 jupyter-core-5.9.1 jupyterlab_widgets-3.0.16 matplotlib-inline-0.2.1 nest-asyncio-1.6.0 packaging-25.0 parso-0.8.5 pexpect-4.9.0 platformdirs-4.5.1 prompt_toolkit-3.0.52 psutil-7.2.1 ptyprocess-0.7.0 pure-eval-0.2.3 pygments-2.19.2 python-dateutil-2.9.0.post0 pyzmq-27.1.0 six-1.17.0 stack_data-0.6.3 tornado-6.5.4 traitlets-5.14.3 typing_extensions-4.15.0 widgetsnbextension-4.0.15

[notice] A new release of pip is available: 24.0 -> 25.3
[notice] To update, run: pip install --upgrade pip
(vllm) [saucouturier@ood ~]$
```

6) Exécuter la commande ci-dessous pour rendre le venv visible depuis la GUI jupyterhub :

```
$> python3 -m ipykernel install --user --name=test-vllm --display-name "vllm"
```

```
(vllm) [saucouturier@ood ~]$ python3 -m ipykernel install --user --name=test-vllm --display-name "vllm"  
Installed kernelspec test-vllm in /nfs/inserm/home_users/saucouturier/.local/share/jupyter/kernels/test-vllm  
(vllm) [saucouturier@ood ~]$
```

7) Exécuter la commande ci-dessous pour installer les librairies nécessaires au projet « VLLM » :

Nous allons installer la libraririe vllm dans sa version 0.8.5.post1 avec toutes ses dépendances. L'installation peut prendre plusieurs minutes.

```
$> pip3 install vllm==0.8.5.post1
```

```
(vllm) [saucouturier@ood ~]$ pip3 install vllm==0.8.5.post1  
Looking in indexes: https://saucouturier:***@nexus.hds.inserm.fr/repository/pypi-group/simple  
Collecting vllm==0.8.5.post1  
  Downloading https://nexus.hds.inserm.fr/repository/pypi-group/packages/vllm/0.8.5.post1/vllm-0.8.5.post1-cp38-abi3-manylinux1_x86_64.whl (326.4 MB)  
    326.4/326.4 MB 12.2 MB/s eta 0:00:00  
Collecting cachetools (from vllm==0.8.5.post1)  
  Using cached https://nexus.hds.inserm.fr/repository/pypi-group/packages/cachetools/6.2.4/cachetools-6.2.4-py3-none-any.whl (11 kB)  
Requirement already satisfied: psutil in /home/saucouturier/vllm/lib/python3.11/site-packages (from vllm==0.8.5.post1) (7.2.1)  
Collecting sentencepiece (from vllm==0.8.5.post1)  
  Using cached https://nexus.hds.inserm.fr/repository/pypi-group/packages/sentencepiece/0.2.1/sentencepiece-0.2.1-cp311-cp311-manylinux_2_27_x86_64.manylinux_2_28_x86_64.whl (1.4 MB)  
Collecting numpy (from vllm==0.8.5.post1)  
  Using cached https://nexus.hds.inserm.fr/repository/pypi-group/packages/numpy/2.4.1/numpy-2.4.1-cp311-cp311-manylinux_2_27_x86_64.manylinux_2_28_x86_64.whl (16.7 MB)  
Collecting requests>=2.26.0 (from vllm==0.8.5.post1)  
  Using cached https://nexus.hds.inserm.fr/repository/pypi-group/packages/requests/2.32.5/requests-2.32.5-py3-none-any.whl (64 kB)  
Collecting tqdm (from vllm==0.8.5.post1)
```

```

Installing collected packages: triton, py-cpuinfo, nvidia-cusparselt-cu12, mpmath, fastrlock, zipp, wrapt, websockets, uvloop, urllib3, typing-inspection, tqdm, sympy, sniffio, shellingham, sentencepiece, safetensors, rpdpy, rignore, regex, pyyaml, python-multipart, python-json-logger, python-dotenv, pydantic-core, pycountry, protobuf, propcache, prometheus_client, pillow, partial-json-parser, opentelemetry-semantic-conventions-ai, nvidia-nvtx-cu12, nvidia-nvjitlink-cu12, nvidia-nccl-cu12, nvidia-curand-cu12, nvidia-cufft-cu12, nvidia-cuda-runtime-cu12, nvidia-cuda-nvrtc-cu12, nvidia-cuda-cupti-cu12, nvidia-cublas-cu12, numpy, ninja, networkx, multidict, msgspec, msgpack, mdurl, MarkupSafe, llvmlite, llguidance, lark, jiter, interegular, idna, httpx, httpcore, httptools, hf-xet, h11, grpcio, fsspec, frozenlist, filelock, fastar, einops, dnspython, distro, diskcache, dill, cloudpickle, click, charset-normalizer, certifi, cachetools, blake3, attrs, astor, annotated-types, annotated-doc, airportsdata, aiohappyeyeballs, yarl, uvicorn, sentry-sdk, scipy, requests, referencing, pydantic, opentelemetry-proto, opencv-python-headless, nvidia-cusparselt-cu12, nvidia-cudnn-cu12, numba, markdown-it-py, Jinja2, importlib_metadata, httpcore, httpx, googleapis-common-protos, gguf, email-validator, depyf, deprecated, cupy-cuda12x, anyio, aiosignal, watchfiles, tiktoken, starlette, rich, pydantic-settings, pydantic-extra-types, opentelemetry-exporter-otlp-proto-common, opentelemetry-api, nvidia-cusolver-cu12, lm-format-enforcer, jsonschema-specifications, huggingface-hub, httpx, aiohttp, typer, torch, tokenizers, rich-toolkit, prometheus-fastapi-instrumentator, opentelemetry-semantic-conventions, openai, jsonschema, fastapi, xformers, transformers, torchaudio, ray, outlines_core, opentelemetry-sdk, mistral-common, fastapi-cloud-cli, fastapi-cli, xgrammar, outlines, opentelemetry-exporter-otlp-proto-http, opentelemetry-exporter-otlp-proto-grpc, compressed-tensors, opentelemetry-exporter-otlp, vllm
Successfully installed MarkupSafe-3.0.3 aiohappyeyeballs-2.6.1 aiohttp-3.13.3 aiosignal-1.4.0 airportsdata-20250909 annotated-doc-0.0.4 annotated-types-0.7.0 anyio-4.12.1 astor-0.8.1 attrs-25.4.0 blake3-1.0.8 cachetools-6.2.4 certifi-2026.1.4 charset-normalizer-3.4.4 click-8.3.1 cloudpickle-3.1.2 compressed-tensors-0.9.3 cupy-cuda12x-13.6.0 deprecated-1.3.1 depyf-0.18.0 dill-0.4.1 diskcache-5.6.3 distro-1.9.0 dnspython-2.8.0 einops-0.8.1 email-validator-2.3.0 fastapi-0.128.0 fastapi-cli-0.0.20 fastapi-cloud-cli-0.11.0 fastar-0.8.0 fastrlock-0.8.3 filelock-3.20.3 frozenlist-1.8.0 fsspec-2026.1.0 gguf-0.17.1 googleapis-common-protos-1.72.0 grpcio-1.76.0 h11-0.16.0 hf-xet-1.2.0 httpcore-1.0.9 httpx-0.28.1 httptools-0.7.1 huggingface-hub-0.36.0 idna-3.11 importlib_metadata-8.0.0 interegular-0.3.3 Jinja2-3.1.6 jiter-0.12.0 jsonschema-4.26.0 jsonschema-specifications-2025.9.1 lark-1.2.2 llguidance-0.7.30 llvmlite-0.44.0 lm-format-enforcer-0.10.12 markdown-it-py-4.0.0 mdurl-0.1.2 mistral-common-1.8.8 mpmath-1.3.0 msgpack-1.1.2 msgspec-0.20.0 multidict-6.7.0 networkx-3.6.1 ninja-1.13.0 numba-0.61.2 numpy-2.2.6 nvidia-cublas-cu12-12.4.5.8 nvidia-cuda-cupti-cu12-12.4.127 nvidia-cuda-nvrtc-cu12-12.4.127 nvidia-cuda-runtime-cu12-12.4.127 nvidia-cudnn-cu12-9.1.0.70 nvidia-cufft-cu12-11.2.1.3 nvidia-curand-cu12-10.3.5.147 nvidia-cusolver-cu12-11.6.1.9 nvidia-cusparselt-cu12-12.3.1.170 nvidia-cusparselt-cu12-0.6.2 nvidia-nccl-cu12-2.21.5 nvidia-nvjitlink-cu12-12.4.127 nvidia-nvtx-cu12-12.4.127 openai-2.15.0 opencv-python-headless-4.13.0.90 opentelemetry-api-1.26.0 opentelemetry-exporter-otlp-1.26.0 opentelemetry-exporter-otlp-proto-common-1.26.0 opentelemetry-exporter-otlp-proto-grpc-1.26.0 opentelemetry-exporter-otlp-proto-http-1.26.0 opentelemetry-proto-1.26.0 opentelemetry-sdk-1.26.0 opentelemetry-semantic-conventions-0.47b0 opentelemetry-semantic-conventions-ai-0.4.13 outlines-0.1.11 outlines_core-0.1.26 partial-json-parser-0.2.1.1.post7 pillow-12.1.0 prometheus-fastapi-instrumentator-7.1.0 prometheus_client-0.24.1 propcache-0.4.1 protobuf-4.25.8 py-cpuinfo-9.0.0 pycountry-24.6.1 pydantic-2.12.5 pydantic-core-2.41.5 pydantic-extra-types-2.11.0 pydantic-settings-2.12.0 python-dotenv-1.2.1 python-json-logger-4.0.0 python-multipart-0.0.21 pyyaml-6.0.3 ray-2.53.0 referencing-0.37.0 regex-2026.1.15 requests-2.32.5 rich-14.2.0 rich-toolkit-0.17.1 rignore-0.7.6 rpdpy-0.30.0 safetensors-0.7.0 scipy-1.17.0 sentencepiece-0.2.1 sentry-sdk-2.50.0 shellingham-1.5.4 sniffio-1.3.1 starlette-0.50.0 sympy-1.13.1 tiktoken-0.12.0 tokenizers-0.22.2 torch-2.6.0 torchaudio-2.6.0 torchvision-0.21.0 tqdm-4.67.1 transformers-4.57.6 triton-3.2.0 typer-0.21.1 typing-inspection-0.4.2 urllib3-2.6.3 uvicorn-0.40.0 uvloop-0.22.1 vllm-0.8.5.post1 watchfiles-1.1.1 websockets-16.0 wrapt-2.0.1 xformers-0.0.29.post2 xgrammar-0.1.18 yarl-1.22.0 zipp-3.23.0
[notice] A new release of pip is available: 24.0 -> 25.3
[notice] To update, run: pip install --upgrade pip

```

La préparation du venv est terminée.

8) Exécuter la commande ci-dessous pour désactiver le venv temporairement :

```
$> deactivate
```

```
(vllm) [hpc_admin@ood ~]$ deactivate
[hpc_admin@ood ~]$
```

A partir de ce point, la ré-activation du venv « vllm » nous permettra d'utiliser la librairie vllm dans notre code python.

Utilisation du code python depuis les noeuds de calculs GPU

Dans les prochains chapitres, nous allons voir comment utiliser notre code python depuis les nœuds de calculs GPU.

Plusieurs options sont possibles, selon vos habitudes :

Mise en oeuvre des calculs via une console Linux

Détaillée au chapitre 3

Mise en oeuvre des calculs via une GUI Linux

Détaillée au chapitre 4

Mise en oeuvre des calculs via jupyterhub

Détaillée au chapitre 5

A noter : Jupyterhub offre plus de fonctionnalités pour le debugage et test du code

3 Mise en œuvre des calculs via une console Linux

1) Cliquer sur le menu Clusters > HPC Shell Access pour accéder au home directory depuis la plateforme OOD HPC

Inserm Portail HPC GitLab Apps Files Jobs Clusters Interactive Apps My Interactive Sessions Help Logged in as hpc_admin Log Out

>_ HPC Shell Access

HPC Shell Access

Inserm

La science pour la santé

 From science to health

OnDemand provides an integrated, single access point for all of your HPC resources.

Pinned Apps A featured subset of [all available apps](#)

Interactive Apps

2) Exécuter la commande ci-dessous pour réserver un nœud de calcul de la partition 2xH100-96 :

```
$> srun -p 2xH100-96 --pty /bin/bash
```

Le prompt prend alors le nom du nœud de calcul, nous indiquant que nous sommes relocalisé sur le nœud de calcul :

```
Host: localhost Themes: Default
[saucouturier@ood ~]$ srun -p 2xH100-96 --pty /bin/bash
[saucouturier@podsgpu-h100-ts-1 ~]$
```

3) Exécuter la commande ci-dessous pour vérifier la présence des gpus :

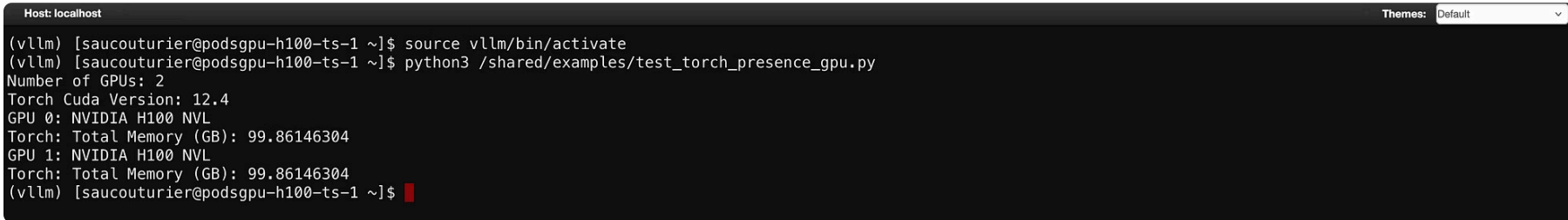
Nous pouvons vérifier la présence des 2 GPUs car nous avons réservé un nœud de la partition 2xH100-96 :

```
$> nvidia-smi -L
```

```
Host: localhost Themes: Default
[saucouturier@podsgpu-h100-ts-1 ~]$ nvidia-smi -L
GPU 0: NVIDIA H100 NVL (UUID: GPU-c67fa2ea-cb54-ba8e-0e81-c103ccb9262e)
GPU 1: NVIDIA H100 NVL (UUID: GPU-0740c156-7f64-8738-03c4-a140a0140cf7)
[saucouturier@podsgpu-h100-ts-1 ~]$
```

4) Exécuter la commande ci-dessous pour activer le venv et exécuter un test de présence gpu via python :

```
$> source vllm/bin/activate  
$> python3 /shared/examples/test_torch_presence_gpu.py
```



```
Host: localhost Themes: Default  
(vllm) [saucouturier@podsgpu-h100-ts-1 ~]$ source vllm/bin/activate  
(vllm) [saucouturier@podsgpu-h100-ts-1 ~]$ python3 /shared/examples/test_torch_presence_gpu.py  
Number of GPUs: 2  
Torch Cuda Version: 12.4  
GPU 0: NVIDIA H100 NVL  
Torch: Total Memory (GB): 99.86146304  
GPU 1: NVIDIA H100 NVL  
Torch: Total Memory (GB): 99.86146304  
(vllm) [saucouturier@podsgpu-h100-ts-1 ~]$
```

5) Exécuter la commande « vllm » sur 2 GPUs :

❏ **Attention, la configuration GPU nécessite de positionner les variables d'environnement suivantes :**

- NCCL_IB_DISABLE=1
- NCCL_NET=socket
- NCCL_P2P_DISABLE=1

Dans le cas contraire, les GPUS crash et il faut rebooter physiquement le serveur GPUH100 (bug nvidia).

```
$> NCCL_IB_DISABLE=1 NCCL_NET=socket NCCL_P2P_DISABLE=1 CUDA_VISIBLE_DEVICES=0,1 python3 -m vllm.entrypoints.openai.api_server --model  
"/shared/huggingface/hub/models--meta-llama--Meta-Llama-3.1-8B-Instruct/snapshots/0e9e39f249a16976918f6564b8830bc894c89659" --tensor-parallel-size 2
```

Voici le résultat obtenu :

```
Host: localhost Themes: Default
(vllm) [saucouturier@podsgpu-h100-ts-1 ~]$ NCCL_IB_DISABLE=1 NCCL_NET=socket NCCL_P2P_DISABLE=1 CUDA_VISIBLE_DEVICES=0,1 python3 -m vllm.entrypoints.openai.api_server
--model "/shared/huggingface/hub/models--meta-llama--Meta-Llama-3.1-8B-Instruct/snapshots/0e9e39f249a16976918f6564b8830bc894c89659" --tensor-parallel-size 2
INFO 01-21 10:12:51 [__init__.py:239] Automatically detected platform cuda.
INFO 01-21 10:12:58 [api_server.py:1043] vLLM API server version 0.8.5.post1
INFO 01-21 10:12:58 [api_server.py:1044] args: Namespace(host=None, port=8000, uvicorn_log_level='info', disable_uvicorn_access_log=False, allow_credentials=False, allow
ed_origins=['*'], allowed_methods=['*'], allowed_headers=['*'], api_key=None, lora_modules=None, prompt_adapters=None, chat_template=None, chat_template_content_format='
auto', response_role='assistant', ssl_keyfile=None, ssl_certfile=None, ssl_ca_certs=None, enable_ssl_refresh=False, ssl_cert_reqs=0, root_path=None, middleware=[], retur
n_tokens_as_token_ids=False, disable_frontend_multiprocessing=False, enable_request_id_headers=False, enable_auto_tool_choice=False, tool_call_parser=None, tool_parser_p
login='', model='/shared/huggingface/hub/models--meta-llama--Meta-Llama-3.1-8B-Instruct/snapshots/0e9e39f249a16976918f6564b8830bc894c89659', task='auto', tokenizer=None,
hf_config_path=None, skip_tokenizer_init=False, revision=None, code_revision=None, tokenizer_revision=None, tokenizer_mode='auto', trust_remote_code=False, allowed_loca
l_media_path=None, load_format='auto', download_dir=None, model_loader_extra_config={}, use_tqdm_on_load=True, config_format=<ConfigFormat.AUTO: 'auto'>, dtype='auto', m
ax_model_len=None, guided_decoding_backend='auto', reasoning_parser=None, logits_processor_pattern=None, model_impl='auto', distributed_executor_backend=None, pipeline_p
arallel_size=1, tensor_parallel_size=2, data_parallel_size=1, enable_expert_parallel=False, max_parallel_loading_workers=None, ray_workers_use_nsight=False, disable_cust
om_all_reduce=False, block_size=None, gpu_memory_utilization=0.9, swap_space=4, kv_cache_dtype='auto', num_gpu_blocks_override=None, enable_prefix_caching=None, prefix_c
aching_hash_algo='builtin', cpu_offload_gb=0, calculate_kv_scales=False, disable_sliding_window=False, use_v2_block_manager=True, seed=None, max_logprobs=20, disable_log
_stats=False, quantization=None, rope_scaling=None, rope_theta=None, hf_token=None, hf_overrides=None, enforce_eager=False, max_seq_len_to_capture=8192, tokenizer_pool_s
ize=0, tokenizer_pool_type='ray', tokenizer_pool_extra_config={}, limit_mm_per_prompt={}, mm_processor_kwargs=None, disable_mm_preprocessor_cache=False, enable_lora=None
, enable_lora_bias=False, max_loras=1, max_lora_rank=16, lora_extra_vocab_size=256, lora_dtype='auto', long_lora_scaling_factors=None, max_cpu_loras=None, fully_sharded
loras=False, enable_prompt_adapter=None, max_prompt_adapters=1, max_prompt_adapter_token=0, device='auto', speculative_config=None, ignore_patterns=[], served_model_name
=None, qlora_adapter_name_or_path=None, show_hidden_metrics_for_version=None, otlp_traces_endpoint=None, collect_detailed_traces=None, disable_async_output_proc=False, m
ax_num_batched_tokens=None, max_num_seqs=None, max_num_partial_prefills=1, max_long_partial_prefills=1, long_prefill_token_threshold=0, num_lookahead_slots=0, scheduler_
delay_factor=0.0, preemption_mode=None, num_scheduler_steps=1, multi_step_stream_outputs=True, scheduling_policy='fcfs', enable_chunked_prefill=None, disable_chunked_mm
input=False, scheduler_cls='vllm.core.scheduler.Scheduler', override_neuron_config=None, override_pooler_config=None, compilation_config=None, kv_transfer_config=None, w
orker_cls='auto', worker_extension_cls='', generation_config='auto', override_generation_config=None, enable_sleep_mode=False, additional_config=None, enable_reasoning=F
alse, disable_cascade_attn=False, disable_log_requests=False, max_log_len=None, disable_fastapi_docs=False, enable_prompt_tokens_details=False, enable_server_load_tracki
ng=False)
```

```
INFO 01-21 10:15:06 [kv_cache_utils.py:634] GPU KV cache size: 1,150,720 tokens
INFO 01-21 10:15:06 [kv_cache_utils.py:637] Maximum concurrency for 131,072 tokens per request: 8.78x
INFO 01-21 10:15:06 [kv_cache_utils.py:634] GPU KV cache size: 1,150,720 tokens
INFO 01-21 10:15:06 [kv_cache_utils.py:637] Maximum concurrency for 131,072 tokens per request: 8.78x
(VLLMWorker rank=1 pid=23053) INFO 01-21 10:15:26 [custom_all_reduce.py:195] Registering 4355 cuda graph addresses
(VLLMWorker rank=0 pid=23052) INFO 01-21 10:15:26 [custom_all_reduce.py:195] Registering 4355 cuda graph addresses
(VLLMWorker rank=1 pid=23053) INFO 01-21 10:15:26 [gpu_model_runner.py:1686] Graph capturing finished in 21 secs, took 0.60 GiB
(VLLMWorker rank=0 pid=23052) INFO 01-21 10:15:26 [gpu_model_runner.py:1686] Graph capturing finished in 21 secs, took 0.60 GiB
INFO 01-21 10:15:26 [core.py:159] init engine (profile, create kv cache, warmup model) took 80.49 seconds
INFO 01-21 10:15:26 [core_client.py:439] Core engine process 0 ready.
WARNING 01-21 10:15:27 [config.py:1239] Default sampling parameters have been overridden by the model's Hugging Face generation config recommended from the model creator. If this is not intended, please relaunch vLLM instance with `--generation-config vllm`.
INFO 01-21 10:15:27 [serving_chat.py:118] Using default chat sampling params from model: {'temperature': 0.6, 'top_p': 0.9}
INFO 01-21 10:15:27 [serving_completion.py:61] Using default completion sampling params from model: {'temperature': 0.6, 'top_p': 0.9}
INFO 01-21 10:15:27 [api_server.py:1090] Starting vLLM API server on http://0.0.0.0:8000
INFO 01-21 10:15:27 [launcher.py:28] Available routes are:
INFO 01-21 10:15:27 [launcher.py:36] Route: /openapi.json, Methods: GET, HEAD
INFO 01-21 10:15:27 [launcher.py:36] Route: /docs, Methods: GET, HEAD
INFO 01-21 10:15:27 [launcher.py:36] Route: /docs/oauth2-redirect, Methods: GET, HEAD
INFO 01-21 10:15:27 [launcher.py:36] Route: /redoc, Methods: GET, HEAD
INFO 01-21 10:15:27 [launcher.py:36] Route: /health, Methods: GET
INFO 01-21 10:15:27 [launcher.py:36] Route: /load, Methods: GET
INFO 01-21 10:15:27 [launcher.py:36] Route: /ping, Methods: POST, GET
INFO 01-21 10:15:27 [launcher.py:36] Route: /tokenize, Methods: POST
INFO 01-21 10:15:27 [launcher.py:36] Route: /detokenize, Methods: POST
INFO 01-21 10:15:27 [launcher.py:36] Route: /v1/models, Methods: GET
INFO 01-21 10:15:27 [launcher.py:36] Route: /version, Methods: GET
INFO 01-21 10:15:27 [launcher.py:36] Route: /v1/chat/completions, Methods: POST
INFO 01-21 10:15:27 [launcher.py:36] Route: /v1/completions, Methods: POST
INFO 01-21 10:15:27 [launcher.py:36] Route: /v1/embeddings, Methods: POST
INFO 01-21 10:15:27 [launcher.py:36] Route: /pooling, Methods: POST
INFO 01-21 10:15:27 [launcher.py:36] Route: /score, Methods: POST
INFO 01-21 10:15:27 [launcher.py:36] Route: /v1/score, Methods: POST
INFO 01-21 10:15:27 [launcher.py:36] Route: /v1/audio/transcriptions, Methods: POST
INFO 01-21 10:15:27 [launcher.py:36] Route: /rerank, Methods: POST
INFO 01-21 10:15:27 [launcher.py:36] Route: /v1/rerank, Methods: POST
INFO 01-21 10:15:27 [launcher.py:36] Route: /v2/rerank, Methods: POST
INFO 01-21 10:15:27 [launcher.py:36] Route: /invocations, Methods: POST
INFO 01-21 10:15:27 [launcher.py:36] Route: /metrics, Methods: GET
INFO: Started server process [22650]
INFO: Waiting for application startup.
INFO: Application startup complete.
```

Le code a parfaitement fonctionné et a utilisé la parallélisation des calculs sur deux GPUS.

Utiliser "CTRL-C" pour arrêter le code.

4 Mise en œuvre des calculs via une GUI Linux

Pour bénéficier d'une interface graphique, nous pouvons réserver un nœud de calcul via le widget Inserm Cluster Desktop et y accéder avec une session GUI.

1) Choisir le widget Inserm Cluster Desktop

The screenshot shows the Inserm HPC Portal interface. At the top, there is a navigation bar with the following items: "Inserm Portail HPC", "GitLab", "Apps", "Files", "Jobs", "Clusters", "Interactive Apps", "My Interactive Sessions", "Help", "Logged in as hpc_admin", and "Log Out". The main header features the "Inserm" logo and the tagline "La science pour la santé From science to health". Below this, a message states: "OnDemand provides an integrated, single access point for all of your HPC resources." A section titled "Pinned Apps A featured subset of all available apps" is followed by a "Interactive Apps" header. Underneath, there are seven app tiles arranged in two rows. A large red arrow points to the "INSERM Cluster Desktop" tile in the top row, which is labeled as a "System Installed App".

Interactive Apps						
 BioContainer Shell BioContainer Desktop	 Code Server System Installed App	 APPTAINER Container Shell Container Desktop	 INSERM Cluster Desktop System Installed App			
 JupyterLab System Installed App	 NVIDIA NGC Catalog NVIDIA NGC Container Shell Container Desktop	 RStudio RStudio Server System Installed App				

2) Sélectionner la partition GPU :

Inserm Portail HPC Apps Files Jobs Clusters Interactive Apps My Interactive Sessions

Home / My Interactive Sessions / INSERM Cluster Desktop

Interactive Apps

Desktops

- BioContainer Shell
- Container Shell
- INSERM Cluster Desktop**
- NVIDIA NGC Container Shell

Servers

- Code Server
- JupyterLab

INSERM Cluster Desktop

This app will launch an interactive desktop on one or more compute nodes. You will have full access to the resources these nodes provide. This is analogous to an interactive batch job.

Runtime (in hours) (required / Max 30Hours)

1

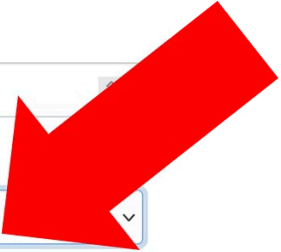
Partition

2xH100-96

Please select a partition from the drop-down.

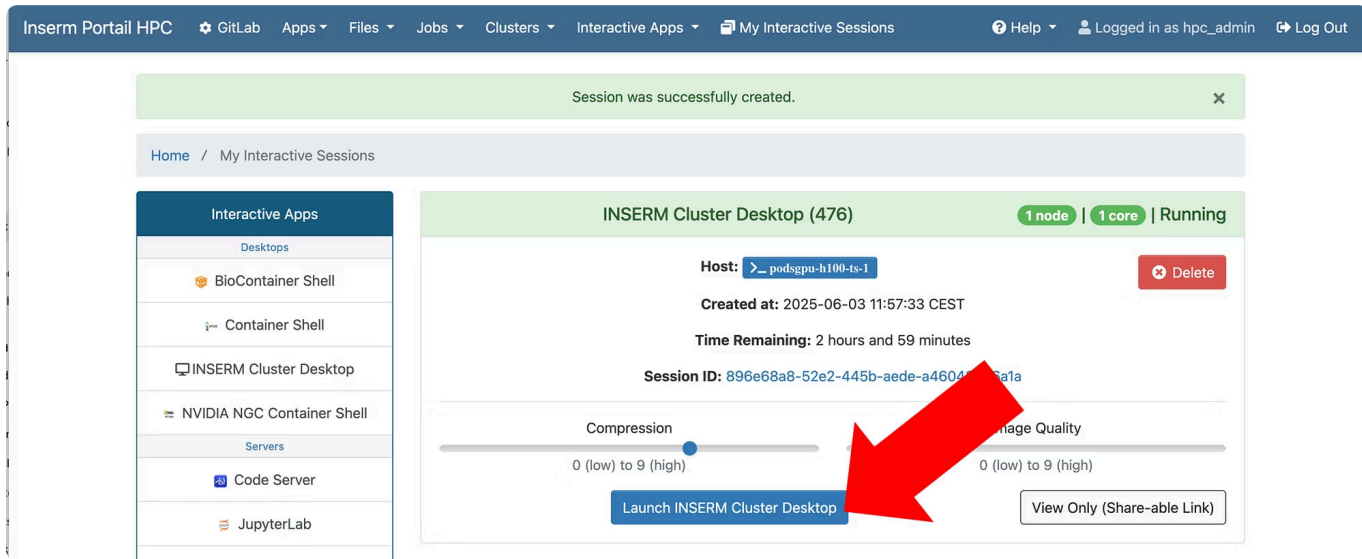
Name of HDS Project:

Launch



* The INSERM Cluster Desktop session data for this session can be accessed under the [data root directory](#).

3) Cliquer sur Launch Inserm Cluster Desktop pour accéder à l'interface



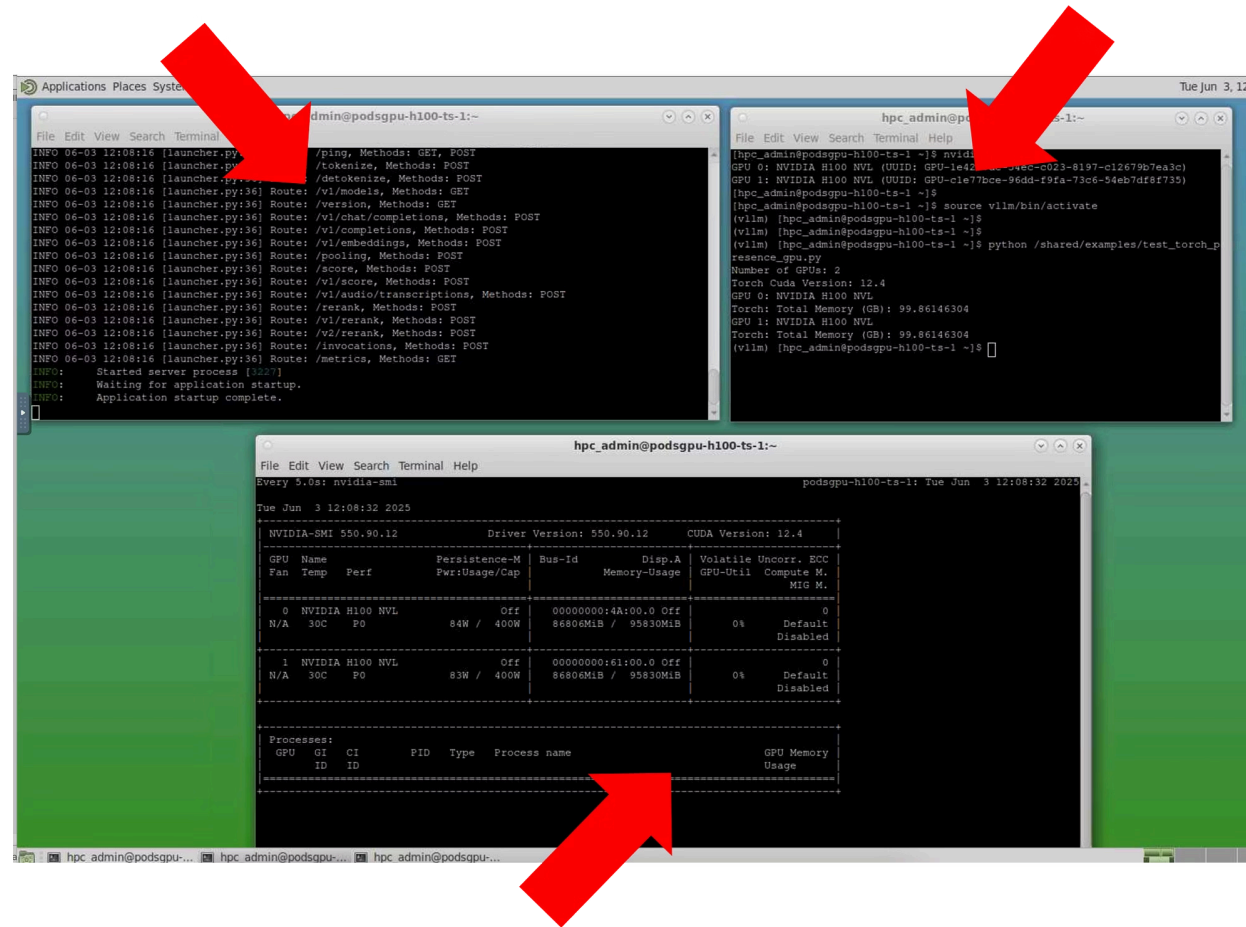
The screenshot shows the 'Inserm Portail HPC' interface. At the top, a navigation bar includes 'GitLab', 'Apps', 'Files', 'Jobs', 'Clusters', 'Interactive Apps', and 'My Interactive Sessions'. A green notification banner states 'Session was successfully created.' Below this, the breadcrumb 'Home / My Interactive Sessions' is visible. On the left, a sidebar titled 'Interactive Apps' lists several options: 'Desktops' (BioContainer Shell, Container Shell, **INSERM Cluster Desktop**, NVIDIA NGC Container Shell) and 'Servers' (Code Server, JupyterLab). The main content area displays details for an 'INSERM Cluster Desktop (476)' session, which is '1 node | 1 core | Running'. Key information includes: Host: podsgpu-h100-ts-1, Created at: 2025-06-03 11:57:33 CEST, Time Remaining: 2 hours and 59 minutes, and Session ID: 896e68a8-52e2-445b-aede-a4604... A red arrow points to the 'Launch INSERM Cluster Desktop' button. Other controls include a 'Delete' button, 'Compression' and 'Image Quality' sliders, and a 'View Only (Share-able Link)' button.



4) Rejouer les commandes du chapitre 3 pour activer le venv, vérifier la présence GPU et lancer le vllm.

Test VLLM via le venv "vllm"

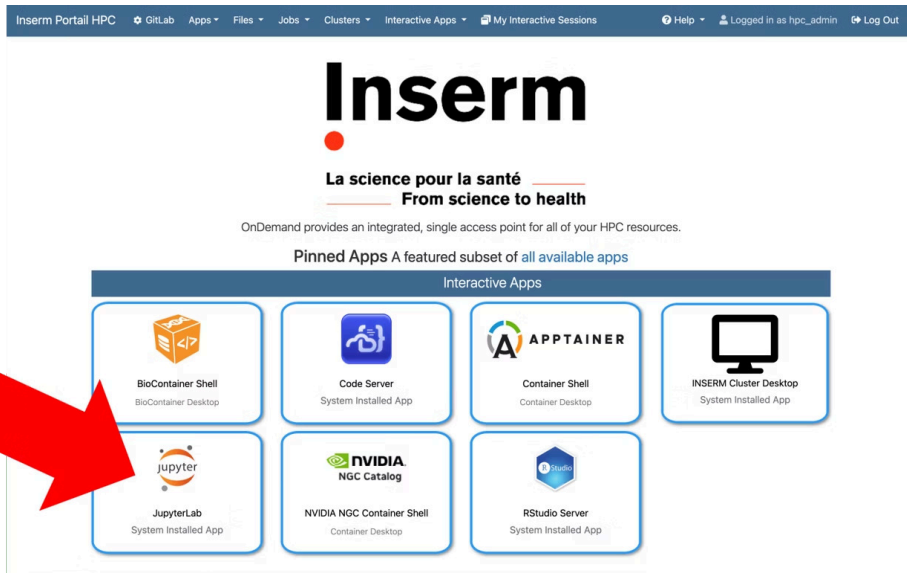
Test Presence GPU via le venv "vllm"



Surveillance GPUs via un "watch -n 5 nvidia-smi »

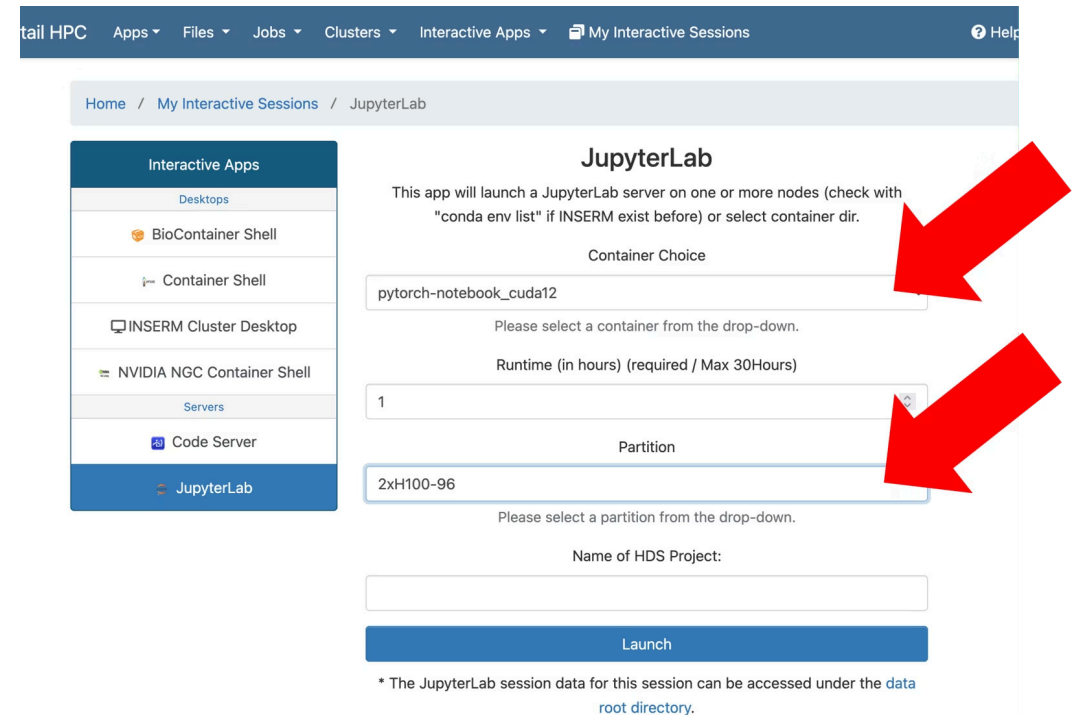
5 Mise en œuvre des calculs via jupyterhub

1) Choisir le widget JupyterLab



The screenshot shows the Inserm portal interface. At the top, there is a navigation bar with 'Inserm Portail HPC', 'GitLab', 'Apps', 'Files', 'Jobs', 'Clusters', 'Interactive Apps', and 'My Interactive Sessions'. Below the navigation bar is the Inserm logo and the tagline 'La science pour la santé From science to health'. The main content area is titled 'Pinned Apps A featured subset of all available apps' and 'Interactive Apps'. A grid of seven app widgets is displayed: BioContainer Shell, Code Server, Container Shell, INSERM Cluster Desktop, JupyterLab, NVIDIA NGC Container Shell, and RStudio Server. A large red arrow points to the JupyterLab widget.

2) Sélectionner le container choice pytorch-notebook_cuda12 et la partition 2xH100-96



The screenshot shows the configuration page for JupyterLab. The navigation bar includes 'tail HPC', 'Apps', 'Files', 'Jobs', 'Clusters', 'Interactive Apps', and 'My Interactive Sessions'. The page title is 'JupyterLab'. A sidebar on the left lists 'Interactive Apps' with categories: Desktops (BioContainer Shell, Container Shell, INSERM Cluster Desktop, NVIDIA NGC Container Shell, Servers (Code Server, JupyterLab)). The main content area contains the following configuration options:

- Container Choice:** A dropdown menu with 'pytorch-notebook_cuda12' selected. A red arrow points to this dropdown.
- Runtime (in hours) (required / Max 30Hours):** A dropdown menu with '1' selected.
- Partition:** A dropdown menu with '2xH100-96' selected. A red arrow points to this dropdown.
- Name of HDS Project:** An empty text input field.
- Launch:** A blue button.

At the bottom, there is a note: '* The JupyterLab session data for this session can be accessed under the [data root directory](#).'

3) Cliquer sur Connect to Jupyter pour accéder à l'interface

Inserm Portail HPC ⚙️ GitLab Apps ▾ Files ▾ Jobs ▾ Clusters ▾ Interactive Apps ▾ 📄 My Interactive Sessions 🔗 Help ▾ 👤 Logged in as hpc_admin 🚪 Log Out

Session was successfully deleted. ✕

Home / My Interactive Sessions

Interactive Apps

Desktops

- BioContainer Shell
- Container Shell
- INSERM Cluster Desktop
- NVIDIA NGC Container Shell

Servers

- Code Server
- JupyterLab
- RStudio Server

JupyterLab (477) 1 node | 1 core | Running

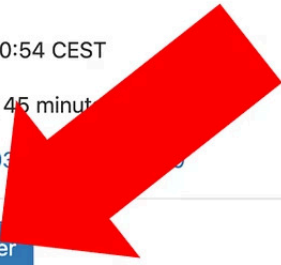
Host: `>_ podsgpu-h100-ts-0` ✕ Delete

Created at: 2025-06-03 12:10:54 CEST

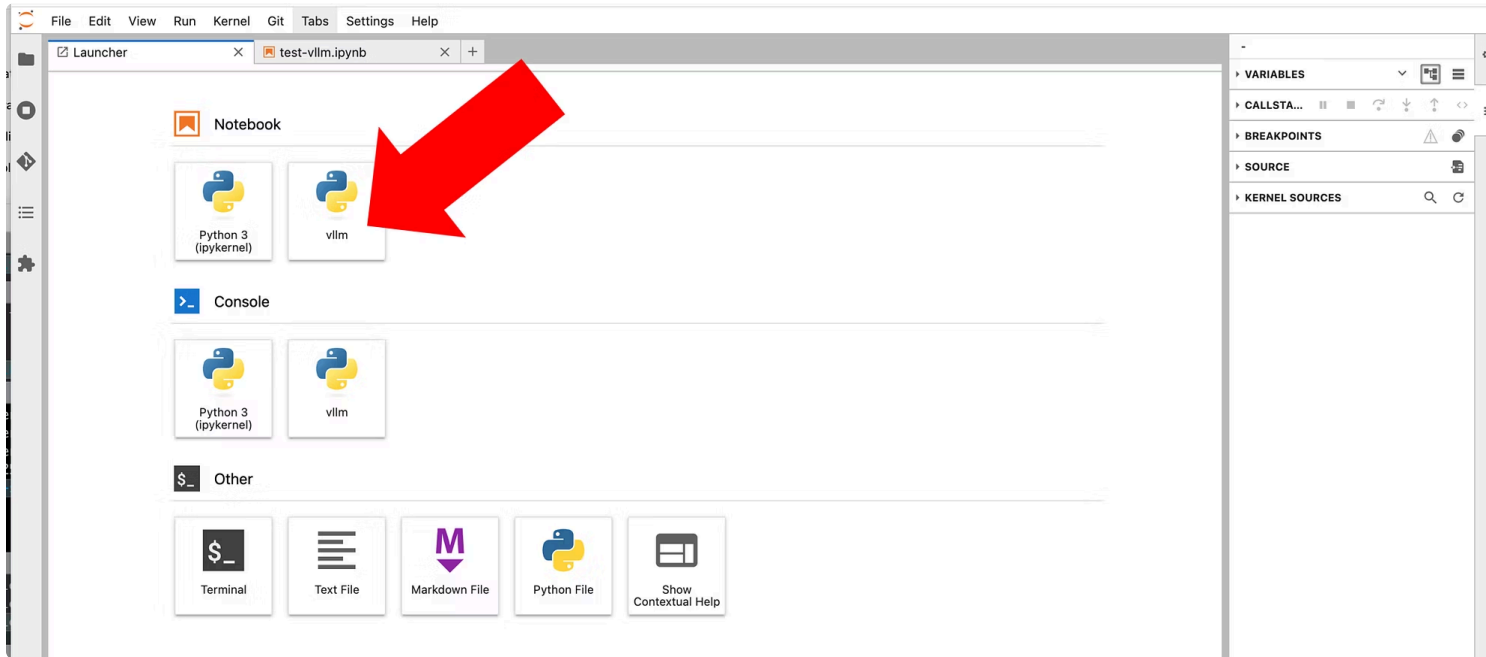
Time Remaining: 2 hours and 45 minutes

Session ID: 0061d016-06fa-45ce-b93...

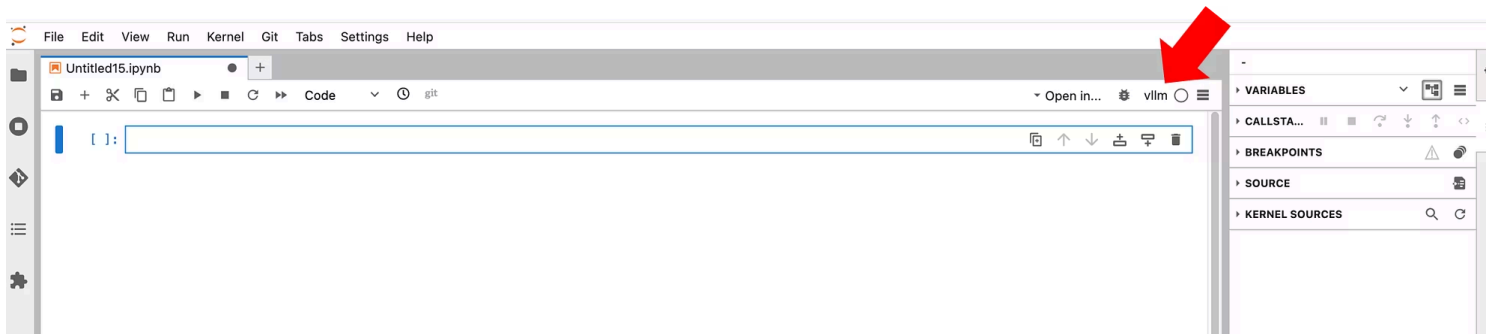
[👁️ Connect to Jupyter](#)



4) Cliquer sur "vllm" disponible dans l'interface jupyterhub de notre compte grâce à l'enregistrement de l'environnement virtuel au chapitre 2.

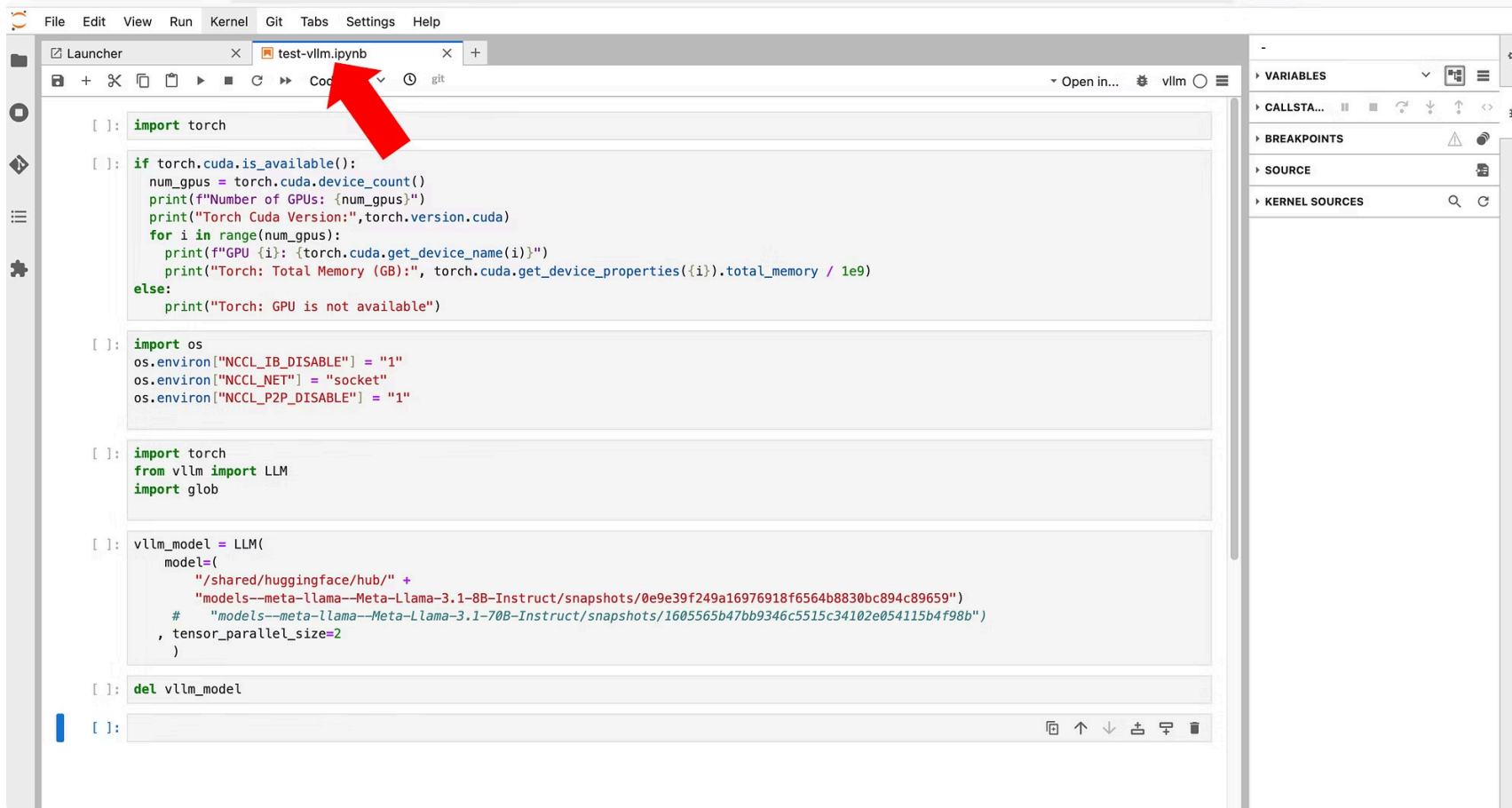


4 bis) Bien s'assurer d'être dans le venv vllm dans un notebook python vide (donc à python 3.11)



5) Cliquer sur l'onglet test vllm

Le code applicatif est recopié sous forme de bloc exécutif et le fichier est sauvegardé :



```
[ ]: import torch

[ ]: if torch.cuda.is_available():
    num_gpus = torch.cuda.device_count()
    print(f"Number of GPUs: {num_gpus}")
    print("Torch Cuda Version:", torch.version.cuda)
    for i in range(num_gpus):
        print(f"GPU {i}: {torch.cuda.get_device_name(i)}")
        print("Torch: Total Memory (GB):", torch.cuda.get_device_properties(i).total_memory / 1e9)
    else:
        print("Torch: GPU is not available")

[ ]: import os
os.environ["NCCL_IB_DISABLE"] = "1"
os.environ["NCCL_NET"] = "socket"
os.environ["NCCL_P2P_DISABLE"] = "1"

[ ]: import torch
from vllm import LLM
import glob

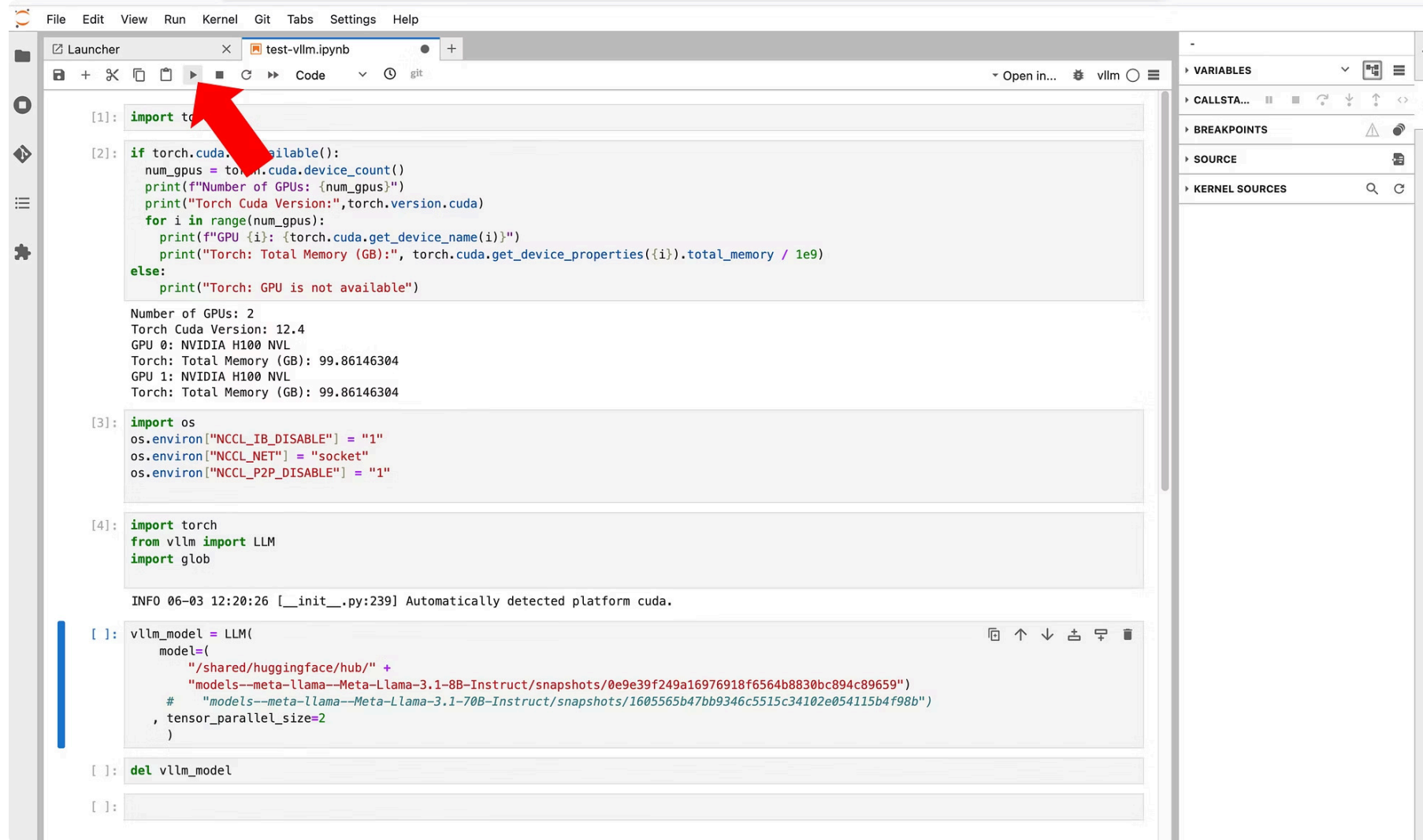
[ ]: vllm_model = LLM(
    model=(
        "/shared/huggingface/hub/" +
        "models--meta-llama--Meta-Llama-3.1-8B-Instruct/snapshots/@e9e39f249a16976918f6564b8830bc894c89659")
        # "models--meta-llama--Meta-Llama-3.1-70B-Instruct/snapshots/1605565b47bb9346c5515c34102e054115b4f98b")
    , tensor_parallel_size=2
)

[ ]: del vllm_model

[ ]:
```

Point d'attention : Le bloc 3 du code configure les variables d'environnement essentielles pour prévenir tout crash des GPUs (voir détails au chapitre 3.4).

6) Cliquer sur le bouton play pour procéder au run, bloc après bloc



The screenshot shows a Jupyter Notebook interface with a menu bar (File, Edit, View, Run, Kernel, Git, Tabs, Settings, Help) and a toolbar with various icons. The notebook has a single tab titled "test-vllm.ipynb". The code is organized into several cells:

- Cell [1]:** `import torch`
- Cell [2]:** Checks for GPU availability and prints details:

```
if torch.cuda.is_available():
    num_gpus = torch.cuda.device_count()
    print(f"Number of GPUs: {num_gpus}")
    print("Torch Cuda Version:", torch.version.cuda)
    for i in range(num_gpus):
        print(f"GPU {i}: {torch.cuda.get_device_name(i)}")
        print("Torch: Total Memory (GB):", torch.cuda.get_device_properties(i).total_memory / 1e9)
else:
    print("Torch: GPU is not available")
```

Output: `Number of GPUs: 2`, `Torch Cuda Version: 12.4`, `GPU 0: NVIDIA H100 NVL`, `Torch: Total Memory (GB): 99.86146304`, `GPU 1: NVIDIA H100 NVL`, `Torch: Total Memory (GB): 99.86146304`
- Cell [3]:** Sets environment variables:

```
import os
os.environ["NCCL_IB_DISABLE"] = "1"
os.environ["NCCL_NET"] = "socket"
os.environ["NCCL_P2P_DISABLE"] = "1"
```
- Cell [4]:** Imports vllm and glob:

```
import torch
from vllm import LLM
import glob
```

Output: `INFO 06-03 12:20:26 [__init__.py:239] Automatically detected platform cuda.`
- Cell []:** Defines the vllm model:

```
vllm_model = LLM(
    model={
        "/shared/huggingface/hub/" +
        "models--meta-llama--Meta-Llama-3.1-8B-Instruct/snapshots/0e9e39f249a16976918f6564b8830bc894c89659")
        # "models--meta-llama--Meta-Llama-3.1-70B-Instruct/snapshots/1605565b47bb9346c5515c34102e054115b4f98b")
    , tensor_parallel_size=2
    )
```
- Cell []:** `del vllm_model`
- Cell []:** (Empty)

A red arrow points to the play button (a right-pointing triangle) in the toolbar above the first code cell. On the right side of the interface, there are panels for "VARIABLES", "CALLSTACK", "BREAKPOINTS", "SOURCE", and "KERNEL SOURCES".

Voici le run du bloc test VLLM ci-dessous :

```
[5]: vllm_model = LLM(
      model=(
        "/shared/huggingface/hub/" +
        "models--meta-llama--Meta-Llama-3.1-8B-Instruct/snapshots/0e9e39f249a16976918f6564b8830bc894c89659")
        # "models--meta-llama--Meta-Llama-3.1-70B-Instruct/snapshots/1605565b47bb9346c5515c34102e054115b4f98b")
      , tensor_parallel_size=2
    )

INFO 06-03 12:21:28 [config.py:717] This model supports multiple tasks: {'generate', 'classify', 'embed', 'reward', 'score'}. Defaulting to 'generate'.
INFO 06-03 12:21:28 [config.py:1770] Defaulting to use mp for distributed inference
INFO 06-03 12:21:28 [config.py:2003] Chunked prefill is enabled with max_num_batched_tokens=16384.
WARNING 06-03 12:21:31 [utils.py:2382] We must use the 'spawn' multiprocessing start method. Overriding VLLM_WORKER_MULTIPROC_METHOD to 'spawn'. See https://docs.vllm.ai/en/latest/getting_started/troubleshooting.html#python-multiprocessing for more information. Reason: CUDA is initialized
INFO 06-03 12:21:41 [__init__.py:239] Automatically detected platform cuda.
INFO 06-03 12:21:47 [core.py:58] Initializing a V1 LLM engine (v0.8.5.post1) with config: model='/shared/huggingface/hub/models--meta-llama--Meta-Llama-3.1-8B-Instruct/snapshots/0e9e39f249a16976918f6564b8830bc894c89659', speculative_config='/shared/huggingface/hub/models--meta-llama--Meta-Llama-3.1-8B-Instruct/snapshots/0e9e39f249a16976918f6564b8830bc894c89659', skip_tokenizer_init=False, tokenizer_mode=auto, revision=None, override_neuron_config=None, tokenizer_revision=None, trust_remote_code=False, dtype=torch.bfloat16, max_seq_len=131072, download_dir=None, load_format=LoadFormat.AUTO, tensor_parallel_size=2, pipeline_parallel_size=1, disable_custom_all_reduce=False, quantization=None, enforce_eager=False, kv_cache_dtype=auto, device_config=cuda, decoding_config=DecodingConfig(guided_decoding_backend='auto'), reasoning_backend=None, observability_config=ObservabilityConfig(show_hidden_metrics=False, otlp_traces_endpoint=None, collect_model_forward_time=False, collect_model_execute_time=False), seed=None, served_model_name='/shared/huggingface/hub/models--meta-llama--Meta-Llama-3.1-8B-Instruct/snapshots/0e9e39f249a16976918f6564b8830bc894c89659', num_scheduler_steps=1, multi_step_stream_outputs=True, enable_prefix_caching=True, chunked_prefill_enabled=True, use_async_output_proc=True, disable_mm_preprocessor_cache=False, mm_processor_kwargs=None, pooler_config=None, compilation_config={'level': 3, 'custom_ops': ['none'], 'splitting_ops': ['vllm.unified_attention', 'vllm.unified_attention_without']}, 'use_inductor': True, 'compile_sizes': [], 'use_cudagraph': True, 'cudagraph_num_of_warmups': 1, 'cudagraph_capture_sizes': [512, 504, 496, 488, 480, 472, 464, 456, 448, 440, 432, 424, 416, 408, 400, 392, 384, 376, 368, 360, 352, 344, 336, 328, 320, 312, 304, 296, 288, 280, 272, 264, 256, 248, 240, 232, 224, 216, 208, 200, 192, 184, 176, 168, 160, 152, 144, 136, 128, 120, 112, 104, 96, 88, 80, 72, 64, 56, 48, 40, 32, 24, 16, 8, 4, 2, 1], 'max_capture_size': 512}
WARNING 06-03 12:21:47 [multiproc_worker_utils.py:306] Reducing Torch parallelism from 64 threads to 1 to avoid unnecessary CPU contention. Set OMP_NUM_THREADS in the external environment to tune this value as needed.
INFO 06-03 12:21:47 [shm_broadcast.py:266] vLLM message queue communication handle: Handle(local_reader_ranks=[0, 1], buffer_handle=(2, 10485760, 10, 'psm_0dc09b84'), local_subscribe_addr='ipc:///tmp/146f61aa-93d3-491a-b460-defffa2d3900', remote_subscribe_addr=None, remote_addr_ipv6=False)
INFO 06-03 12:21:57 [__init__.py:239] Automatically detected platform cuda.
INFO 06-03 12:21:57 [__init__.py:239] Automatically detected platform cuda.
WARNING 06-03 12:22:02 [utils.py:2522] Methods determine_num_available_blocks, device_config, get_cache_block_size_bytes, initialize_cache not implemented in <vllm.v1.worker.gpu_worker.Worker object at 0x7fd2fb3d750>
(VLLMWorker rank=0 pid=10308) INFO 06-03 12:22:02 [shm_broadcast.py:266] vLLM message queue communication handle: Handle(local_reader_ranks=[0], buffer_handle=(1, 10485760, 10, 'psm_6b49a6ce'), local_subscribe_addr='ipc:///tmp/f659772e-834c-4753-a61c-081ebe0f95e1', remote_subscribe_addr=None, remote_addr_ipv6=False)
WARNING 06-03 12:22:02 [utils.py:2522] Methods determine_num_available_blocks, device_config, get_cache_block_size_bytes, initialize_cache not implemented in <vllm.v1.worker.gpu_worker.Worker object at 0x7f24a8596890>
(VLLMWorker rank=1 pid=10309) INFO 06-03 12:22:02 [shm_broadcast.py:266] vLLM message queue communication handle: Handle(local_reader_ranks=[0], buffer_handle=(1, 10485760, 10, 'psm_1c6ab220'), local_subscribe_addr='ipc:///tmp/a2855015-99c5-45a4-a149-419cddf97ea8', remote_subscribe_addr=None, remote_addr_ipv6=False)
(VLLMWorker rank=0 pid=10308) INFO 06-03 12:22:03 [utils.py:1055] Found nccl from library libnccl.so.2
(VLLMWorker rank=0 pid=10308) INFO 06-03 12:22:03 [pynnccl.py:69] vLLM is using nccl=2.21.5
```

```
File Edit View Run Kernel Git Tabs Settings Help
test-vllm.ipynb
(VllmWorker rank=1 pid=10309) INFO 06-03 12:22:04 [parallel_state.py:1004] rank 1 in world size 2 is assigned as DP rank 0, PP rank 0, TP rank 1
(VllmWorker rank=1 pid=10309) INFO 06-03 12:22:04 [cuda.py:221] Using Flash Attention backend on V1 engine.
(VllmWorker rank=1 pid=10309) WARNING 06-03 12:22:04 [topk_top_sampler.py:69] FlashInfer is not available. Falling back to the PyTorch-native implementation of top-p & top-k sampling. For the best performance, please install FlashInfer.
(VllmWorker rank=0 pid=10308) INFO 06-03 12:22:04 [parallel_state.py:1004] rank 0 in world size 2 is assigned as DP rank 0, PP rank 0, TP rank 0
(VllmWorker rank=0 pid=10308) INFO 06-03 12:22:04 [cuda.py:221] Using Flash Attention backend on V1 engine.
(VllmWorker rank=0 pid=10308) WARNING 06-03 12:22:04 [topk_top_sampler.py:69] FlashInfer is not available. Falling back to the PyTorch-native implementation of top-p & top-k sampling. For the best performance, please install FlashInfer.
(VllmWorker rank=1 pid=10309) INFO 06-03 12:22:04 [gpu_model_runner.py:1329] Starting to load model /shared/huggingface/hub/models--meta-llama--Meta-Llama-3.1-8B-Instruct/snapshots/0e9e39f249a16976918f6564b8830bc894c89659...
(VllmWorker rank=0 pid=10308) INFO 06-03 12:22:04 [gpu_model_runner.py:1329] Starting to load model /shared/huggingface/hub/models--meta-llama--Meta-Llama-3.1-8B-Instruct/snapshots/0e9e39f249a16976918f6564b8830bc894c89659...
Loading safetensors checkpoint shards: 0% Completed | 0/4 [00:00<7, 7it/s]
Loading safetensors checkpoint shards: 25% Completed | 1/4 [00:00<00:00, 8.09it/s]
Loading safetensors checkpoint shards: 50% Completed | 2/4 [00:00<00:00, 2.96it/s]
Loading safetensors checkpoint shards: 75% Completed | 3/4 [00:01<00:00, 2.31it/s]
Loading safetensors checkpoint shards: 100% Completed | 4/4 [00:01<00:00, 2.09it/s]
Loading safetensors checkpoint shards: 100% Completed | 4/4 [00:01<00:00, 2.35it/s]
(VllmWorker rank=0 pid=10308)
(VllmWorker rank=1 pid=10309) INFO 06-03 12:22:06 [loader.py:458] Loading weights took 1.75 seconds
(VllmWorker rank=0 pid=10308) INFO 06-03 12:22:06 [loader.py:458] Loading weights took 1.75 seconds
(VllmWorker rank=0 pid=10308) INFO 06-03 12:22:07 [gpu_model_runner.py:1347] Model loading took 7.5123 GiB and 1.959231 seconds
(VllmWorker rank=1 pid=10309) INFO 06-03 12:22:07 [gpu_model_runner.py:1347] Model loading took 7.5123 GiB and 1.965942 seconds
(VllmWorker rank=1 pid=10309) INFO 06-03 12:22:16 [backends.py:420] Using cache directory: /home/hpc_admin/.cache/vllm/torch_compile_cache/89ffa88695/rank_1_0 for vLLM's torch.compile
(VllmWorker rank=1 pid=10309) INFO 06-03 12:22:16 [backends.py:430] Dynamo bytecode transform time: 9.74 s
(VllmWorker rank=0 pid=10308) INFO 06-03 12:22:16 [backends.py:420] Using cache directory: /home/hpc_admin/.cache/vllm/torch_compile_cache/89ffa88695/rank_0_0 for vLLM's torch.compile
(VllmWorker rank=0 pid=10308) INFO 06-03 12:22:16 [backends.py:430] Dynamo bytecode transform time: 9.74 s
(VllmWorker rank=1 pid=10309) INFO 06-03 12:22:23 [backends.py:118] Directly load the compiled graph(s) for shape None from the cache, took 5.939 s
(VllmWorker rank=0 pid=10308) INFO 06-03 12:22:23 [backends.py:118] Directly load the compiled graph(s) for shape None from the cache, took 6.475 s
(VllmWorker rank=1 pid=10309) INFO 06-03 12:22:24 [monitor.py:33] torch.compile takes 9.74 s in total
(VllmWorker rank=0 pid=10308) INFO 06-03 12:22:24 [monitor.py:33] torch.compile takes 9.74 s in total
INFO 06-03 12:22:26 [kv_cache_utils.py:634] GPU KV cache size: 1,153,280 tokens
INFO 06-03 12:22:26 [kv_cache_utils.py:637] Maximum concurrency for 131,072 tokens per request: 8.80x
INFO 06-03 12:22:26 [kv_cache_utils.py:634] GPU KV cache size: 1,153,280 tokens
INFO 06-03 12:22:26 [kv_cache_utils.py:637] Maximum concurrency for 131,072 tokens per request: 8.80x
(VllmWorker rank=1 pid=10309) INFO 06-03 12:22:41 [custom_all_reduce.py:195] Registering 4355 cuda graph addresses
(VllmWorker rank=0 pid=10308) INFO 06-03 12:22:41 [custom_all_reduce.py:195] Registering 4355 cuda graph addresses
(VllmWorker rank=1 pid=10309) INFO 06-03 12:22:41 [gpu_model_runner.py:1686] Graph capturing finished in 16 secs, took 0.60 GiB
(VllmWorker rank=0 pid=10308) INFO 06-03 12:22:41 [gpu_model_runner.py:1686] Graph capturing finished in 16 secs, took 0.60 GiB
INFO 06-03 12:22:41 [core.py:159] init engine (profile, create kv cache, warmup model) took 34.71 seconds
INFO 06-03 12:22:41 [core_client.py:439] Core engine process 0 ready.
[6]: del vllm_model
```

☑ Succès : Le bloc test VLLM s'exécute parfaitement ! Les ressources GPU sont ensuite correctement libérées grâce à l'instruction « del vllm_model ».

**Si vous avez des questions,
contacter vos référents
habituels sur le HPC.**

