NLP 2025
CINECA HPC

Slides provided by:
- Luca Moroni
- Roberto Navigli

Cineca

Non profit Consortium, made up of 119 members:

- **2 Ministries**
- **70 Italian Universities**
- **48 Italian National Institutions and Agencies**

CINECA

Cineca: CLUSTER

Cineca is one of the Large Scale Facilities in Europe

- **LEONARDO:** pre-exascale Tier-0 EuroHPC supercomputer. LEONARDO is classified in the 6° position in the Top500 list

Cineca: CLUSTER

Cineca is one of the Large Scale Facilities in Europe

- **LEONARDO:** pre-exascale Tier-0 EuroHPC supercomputer. LEONARDO is classified in the 6° position in the Top500 list
- **MARCONI:** Tier-0 EuroHPC supercomputer less powerful than LEONARDO
- **GALILEO100:** Tier-1 EuroHPC

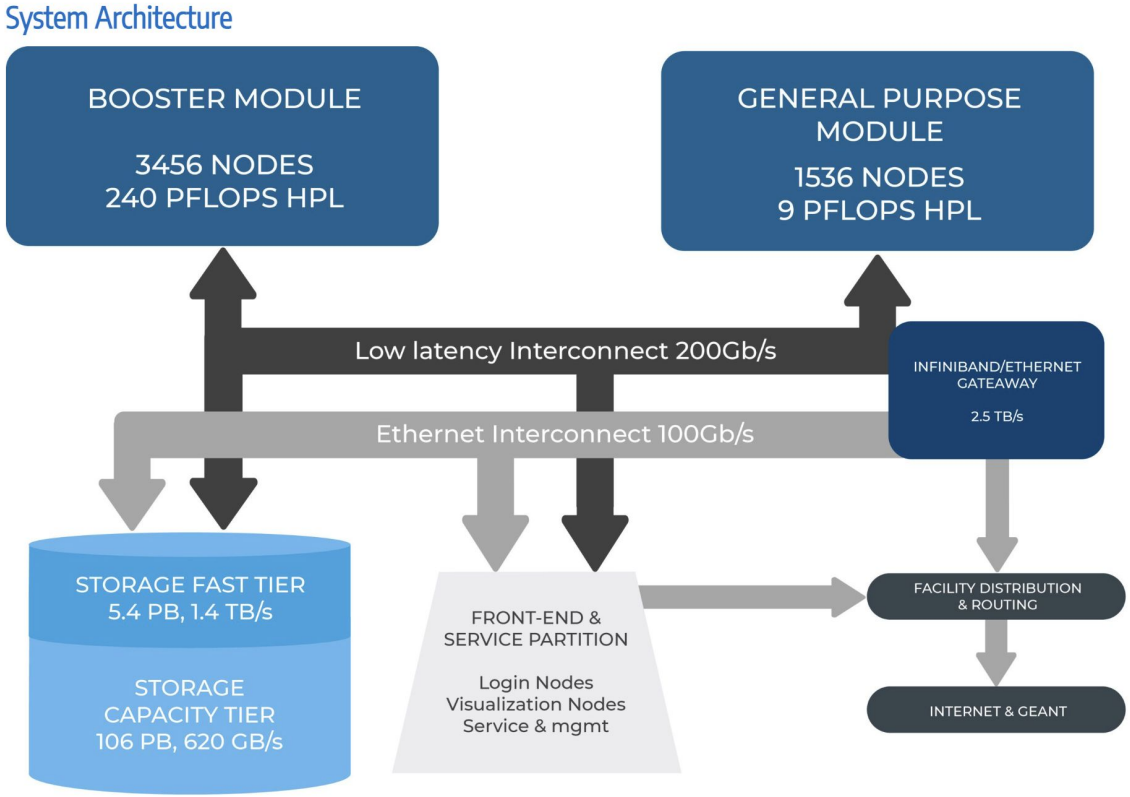
CINECA DOCUMENTATION

The CINECA HPC is well documented, and the **helpdesk is always active** to help you!

- **Official Documentation:** https://docs.hpc.cineca.it/index.html
- **Support E-Mail:** superc@cineca.it

LEONARDO

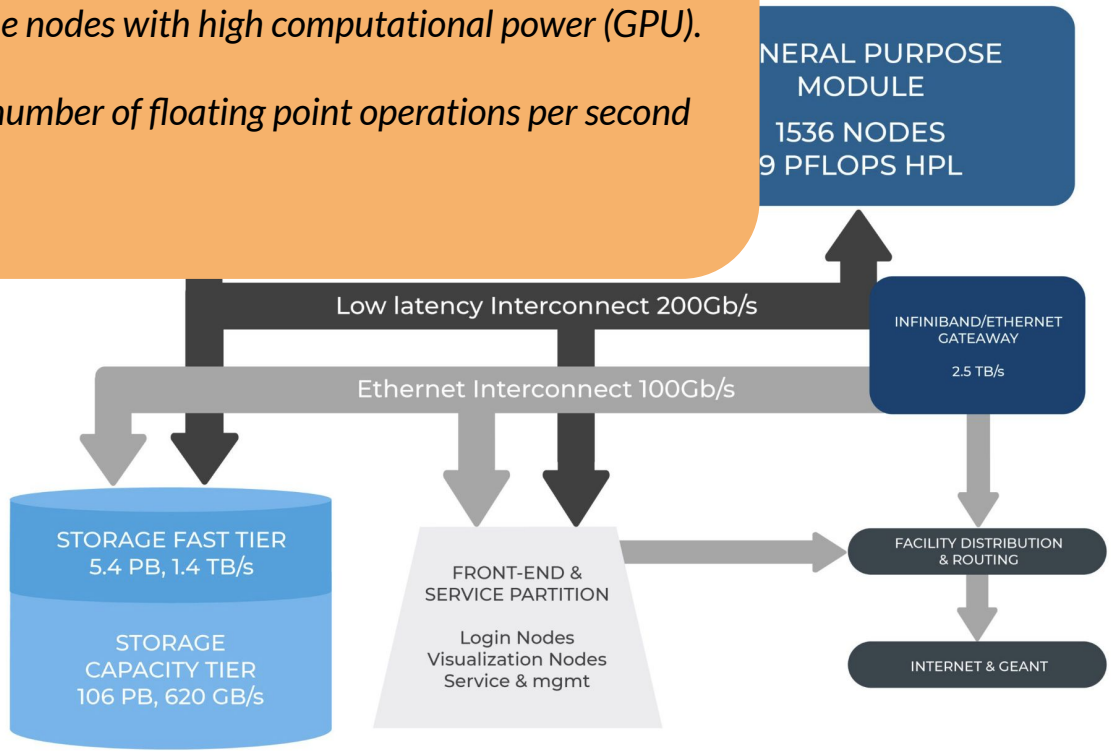
Leonardo



SAPIENZA NLP

Booster Modules: *the nodes with high computational power (GPU).*

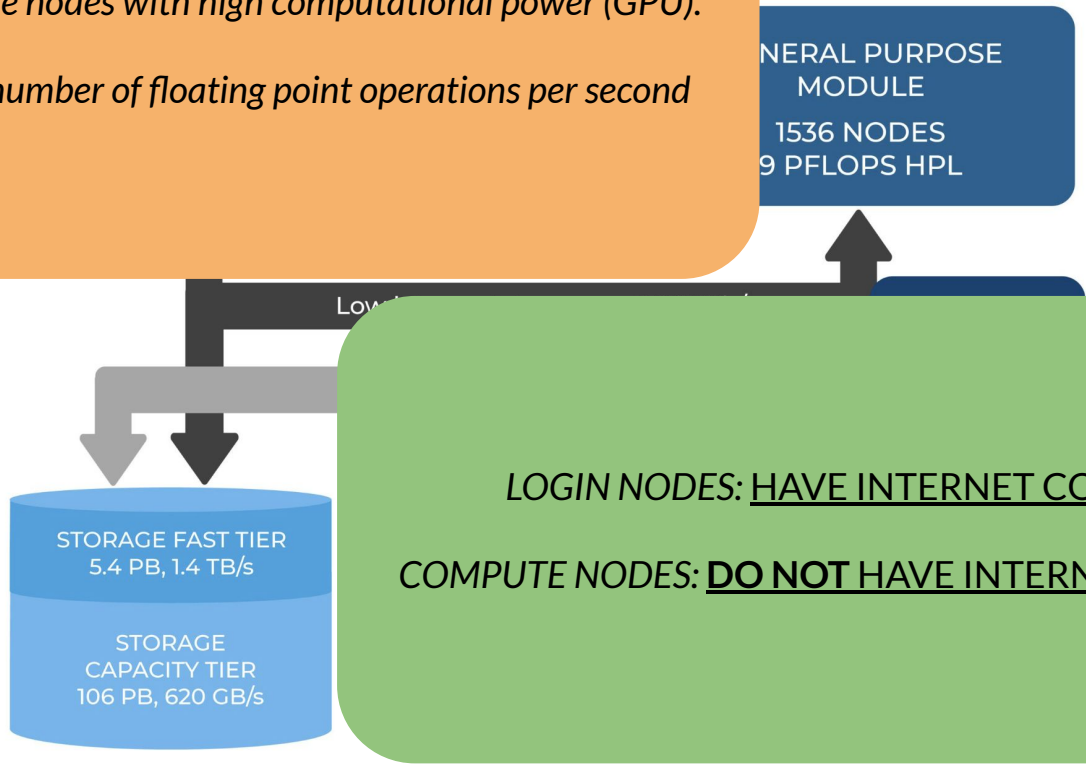
PetaFlops: 10^15 number of floating point operations per second



SAPIENZA
NLP

Booster Modules: *the nodes with high computational power (GPU).*

PetaFlops: 10^15 number of floating point operations per second



LOGIN NODES: HAVE INTERNET CONNECTION

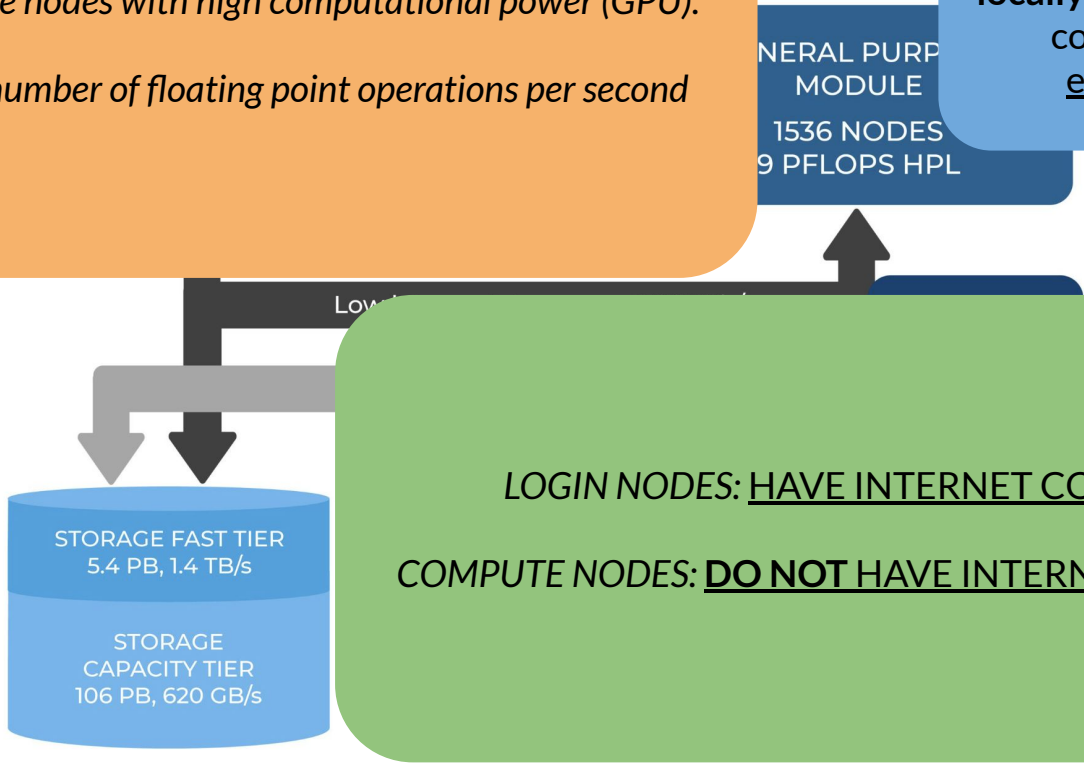
COMPUTE NODES: **DO NOT** HAVE INTERNET CONNECTION

SAPIENZA NLP

Booster Modules: *the nodes with high computational power (GPU).*

PetaFlops: 10^15 number of floating point operations per second

You have to **save datasets and models locally** to use them in the compute nodes.
e.g. Cache-HF



LOGIN NODES: HAVE INTERNET CONNECTION

COMPUTE NODES: **DO NOT** HAVE INTERNET CONNECTION

SAPIENZA NLP

Leonardo: Compute Nodes

Booster partition

Model: BullSequana X2135 “Da Vinci” single node GPU Blade

Nodes: 3456

Processors: single socket 32-core Intel Xeon Platinum 8358 CPU, 2.60GHz (Ice Lake)

Cores: 110592

RAM: 8×64 GB DDR4 3200 MHz (512 GB), per node!

Accelerators: 4x NVIDIA custom Ampere A100 GPU 64GB HBM2e, NVLink 3.0 (200GB/s), per node!

Network: 2 x dual port HDR100 per node (400Gbps/node)

First Login

Documentation: https://docs.hpc.cineca.it/general/access.html

1. Registration on UserDB: https://userdb.hpc.cineca.it/.
   a. Prof. Navigli should add you on the project: PROJECT_NAME.
2. Correct configuration of 2FA and OTP: LINK.

Then you can login into one of the 8 Login Nodes:

1. `step ssh login 'your@email.com' --provisioner cineca-hpc`
2. `ssh your-username@login01-ext.leonardo.cineca.it` [note you can use login02, ... login07]

SAPIENZA NLP

LEONARDO Budget and Accounting

Documentation:

https://docs.hpc.cineca.it/hpc/hpc_intro.html#budget-and-accounting

Command: `saldo -b`

Gives you the usage and the CPU hour availability of your account, for the booster partition. [NOTE: The hours are SHARED]

```
(.env) [lmoroni0@login01 llm-foundry]$ saldo -b
--------------------------------------------------------------------------------------------------------------------------------
account                start         end         total       localCluster    totConsumed     totConsumed     monthTotal     monthConsumed
                                                (local h)   Consumed(local h)   (local h)               %      (local h)         (local h)
--------------------------------------------------------------------------------------------------------------------------------
IscrB_medit          20231116    20250616       3566080            3473137       3473137            97.4        185090            306414
```

LEONARDO Budget and Accounting

Documentation:

https://docs.hpc.cine … nting

Command: `saldo -`

Gives you the usage … t, for the booster partition. [NOTE: Th…

IMPORTANT:

1 GPU HOUR == 8 CPU HOURS

```
(.env) [lmoroni0@login01 llm-foundry
```

account	start	e…	…	…	…	…nsumed %	monthTotal (local h)	monthConsumed (local h)
IscrB_medit	20231116	20250616	3566080	3473137	3473137	97.4	185090	306414

SAPIENZA NLP

Leonardo: Data Areas

Name	Area Attributes	Quota	Backup	Note
$HOME	permanent,user specific	50 GB	daily	
$WORK	permanent,shared	1 TB	no	Large data to be shared with project's colla
$FAST	permanent, shared	1 TB	no	Only on Leonardo. Faster I/O compared with outer areas.
$SCRATCH	temporary,user specific	-/20 TB	no	files older than 40 days are deleted
$TMPDIR	temporary,user specific	(-)	no	directory removed at job completion
$PUBLIC	permanent,open	50 GB	no	Only on Leonardo.
$DRES	permanent,shared	defined by project	no	

Leonardo: Data Areas

Name	Area Attributes			
$HOME	permanent,use			
$WORK	permanent,sha			ith project's colla
$FAST	permanent, sh			outer areas.
$SCRATCH	temporary,use			
$TMPDIR	temporary,use			
$PUBLIC	permanent,open			
$DRES	permanent,shared	defined by project	no	

HOME: your home directory, where you will write your code, download the github repos and do lightweight stuff.

Leonardo: Data Areas

Name	Area Attributes			
$HOME	permanent,use			
$WORK	permanent,sha			ith project's colla
$FAST	permanent, sh			outer areas.
$SCRATCH	temporary,use			
$TMPDIR	temporary,use			
$PUBLIC	permanent,open			
$DRES	permanent,shared	defined by project	no	

SCRATCH: your high capable directory, non-permanent, but with a lot of space. save here the hf_cache, models, datasets, and heavy stuffs.

Remember that the data here will be removed after 40 days.

Environment

Environment

Once you are logged-in, you will land in your $HOME directory

Leonardo works with **modules**

Modules are containerized ready-to-use packages and tools.

https://docs.hpc.cineca.it/hpc/hpc_enviroment.html

Command	Action
module avail	show the available modules on the machine
module load <appl>	load the module in the current shell session, preparing the enviroment for the application.
module load autoload <appl>	load the module and all dependencies in the current session
module help <appl>	show specific information and basic help on the application
module list	show the module currently loaded in the shell session
module purge	unload all the loaded modules
module unload <appl>	unload a specific module

Environment: install python libraries

To install and use a Python library you have to load the following modules:

- module load profile/deeplrn
- module load cuda/12.1 # available even 12.3

Once you have loaded the modules you can create a virtual environment and install the libraries:

```
$ python -m venv .env       # create the environment
$ source .env/bin/activate # activate the environment
$ pip install transformers # install library
```

Execution

Once you have created the environment (usually in your $HOME or whatever) you can execute your Python script

You can execute the scripts with the the SLURM system

SLURM -> allows you to execute programs on the compute nodes

https://docs.hpc.cineca.it/hpc/hpc_scheduler.html

SLURM Scheduling

There are different SLURM directives:

- `sbatch <slurm_script.sh>`: submit the job to the scheduler
- `squeue --me`: monitoring your submitted jobs
- `scancel job_id`: cancel the job

SLURM Scripts

You can create a .sh script with the following **SBATCH instructions**:

```
#!/bin/bash

#SBATCH --nodes=1                    # 1 node
#SBATCH --ntasks-per-node=32         # 32 tasks per node
#SBATCH --time=1:00:00               # time limit: 1 hour
#SBATCH --error=myJob.err            # standard error file
#SBATCH --output=myJob.out           # standard output file
#SBATCH --account=<Project Account>  # project account
#SBATCH --partition=<partition_name> # partition name
#SBATCH --qos=<qos_name>             # quality of service

./my_application
```

SLURM partitions

Partition	QOS	#Cores/#GPU per job	Walltime	Max Nodes/cores/GPUs/
lrd_all_serial (**default**)	normal	4 cores (8 logical cores)	04:00:00	1 node / 4 cores (30800 MB RAM)
boost_usr_prod	normal	64 nodes	24:00:00	
	boost_qos_dbg	2 nodes	00:30:00	2 nodes / 64 cores / 8 GP
	boost_qos_bprod	min = 65 nodes max = 256 nodes	24:00:00	256 nodes
	boost_qos_lprod	3 nodes	4-00:00:00	3 nodes / 12 GPUs
boost_fua_dbg	normal	2 nodes	00:10:00	2 nodes / 64 cores / 8 GP
boost_fua_prod	normal	16 nodes	24:00:00	4 running jobs per user ac 32 nodes / 3584 cores
	boost_qos_fuabprod	min = 17 nodes max = 32 nodes	24:00:00	32 nodes / 3584 cores
	qos_fualowprio	16 nodes	08:00:00	

LLM generation

Example: LLM generation

A simple step-by-step guide:

1. **Create your environment**
2. **Load the environment** and **install** the Transformers library
3. **Create the Python script** with your code
4. **Create the slurm script** with all the correct SBATCH directives

Python Script

my_fancy_generation.py

my_fancy_generation.py > ...

```python
import transformers
import torch # type: ignore

model_id = "meta-llama/Meta-Llama-3.1-8B-Instruct" # <- YOUR FANCY MODEL HERE!

pipeline = transformers.pipeline(
    "text-generation",
    model=model_id,
    model_kwargs={"torch_dtype": torch.bfloat16},
    device_map="auto",
)

messages = [
    {"role": "system", "content": "You are a pirate chatbot who always responds in pirate speak!"},
    {"role": "user", "content": "Who are you?"},
]

outputs = pipeline(
    messages,
    max_new_tokens=256,
)
print(outputs[0]["generated_text"][-1])
```

SLURM Script

my_fancy_generation.slurm.sh

$ my_fancy_generation.slurm.sh

```
#!/bin/bash
#SBATCH --job-name=my_fancy_gen                                  # Job name
#SBATCH --output=/path/to/your/log/folder/log_name-%x-%j.out     # Name of stdout output file. %x job_name, %j job_number
#SBATCH --error=/path/to/your/log/folder/log_name-%x-%j.err      # Name of stdout output file. %x job_name, %j job_number

#SBATCH -A <Account_Name>                                        # account name
#SBATCH -p boost_usr_prod                                        # partition usually production
#SBATCH -q boost_qos_bprod                                       # quality of service
#SBATCH --time 24:00:00                                          # timing: HH:MM:SS for booster maximum 24H
#SBATCH -N 1                                                     # number of nodes, one should be enough
#SBATCH --ntasks=1                                               # number of tasks
#SBATCH --ntasks-per-node=1                                      # number of tasks per node
#SBATCH --cpus-per-task=1                                        # number of cpu per tasks
#SBATCH --gres=gpu:1                                             # number of GPU per node, 1 should be enough


# load the modules
module load profile/deeplrn cuda/12.1

# activate your environment
source /path/to/your/environment/folder/bin/activate

# load and set the Huggingface CACHE, to retrieve cached informations in a no-internet environment

export HF_DATASETS_CACHE=/path/to/hf/cache
export HUGGINGFACE_HUB_CACHE=/path/to/hf/cache
export WANDB_MODE=offline # set the wandb offline (no needed for generation...)
# read Huggingface token from .env file
export HF_TOKEN=$(python -c "import huggingface_hub; print(huggingface_hub.HfFolder.get_token() or '')")

# execute the python script
python /path/to/your/python/script/my_fancy_generation.py
```

SAPIENZA NLP

SLURM Script

my_fancy_generation.slurm.sh

```
$ my_fancy_generation.slurm.sh
#!/bin/bash
#SBATCH --job-name=my_fancy_gen                    # job name
#SBATCH --output=/path/to/your/log/folder/log_name-%x-%j.out    # Name of stdout output file. %x job_name, %j job_number
#SBATCH --error=/path/to/your/log/folder/log_name-%x-%j.err     # Name of stdout output file. %x job_name, %j job_number

#SBATCH -A <Account_Name>                          # account name
#SBATCH -p boost_usr_prod                          # partition usually production
#SBATCH -q boost_qos_bprod                         ce
#SBATCH --time 24:00:00                            for booster maximum 24H
#SBATCH -N 1                                       ne should be enough
#SBATCH --ntasks=1
#SBATCH --ntasks-per-node=1                        r node
#SBATCH --cpus-per-task=1                          tasks
#SBATCH --gres=gpu:1                               node, 1 should be enough


# load the modules
module load profile/deeplrn cuda/

# activate your environment
source /path/to/your/environment/folder/bin/activate

# load and set the Huggingface CACHE, to retrieve cached informations in a no-internet environment

export HF_DATASETS_CACHE=/path/to/hf/cache
export HUGGINGFACE_HUB_CACHE=/path/to/hf/cache
export WANDB_MODE=offline # set the wandb offline (no needed for generation...)
# read Huggingface token from .env file
export HF_TOKEN=$(python -c "import huggingface_hub; print(huggingface_hub.HfFolder.get_token() or '')")

# execute the python script
python /path/to/your/python/script/my_fancy_generation.py
```

Log folder should exists otherwise the process doesn't work

SAPIENZA NLP

SLURM Script

my_fancy_generation.slurm.sh

```
$ my_fancy_generation.slurm.sh
#!/bin/bash
#SBATCH --job-name=my_fancy_gen                                  # Job name
#SBATCH --output=/path/to/your/log/folder/log_name-%x-%j.out     # Name of stdout output file. %x job_name, %j job_number
#SBATCH --error=/path/to/your/log/folder/log_name-%x-%j.err      # Name of stdout output file. %x job_name, %j job_number

#SBATCH -A <Account_Name>                                        # account name
#SBATCH -p boost_usr_prod                                        # partition usually production
#SBATCH -q boost_qos_bprod                                       # quality of service
#SBATCH --time 24:00:00                                                  for booster maximum 24H
#SBATCH -N 1                                                             one should be enough
#SBATCH --ntasks=1
#SBATCH --ntasks-per-node=1                                              er node
#SBATCH --cpus-per-task=1                                                tasks
#SBATCH --gres=gpu:1                                                     node, 1 should be enough


# load the modules
module load profile/deeplrn cu

# activate your environment
source /path/to/your/environment/folder/bin/activate

# load and set the Huggingface CACHE, to retrieve cached informations in a no-internet environment

export HF_DATASETS_CACHE=/path/to/hf/cache
export HUGGINGFACE_HUB_CACHE=/path/to/hf/cache
export WANDB_MODE=offline # set the wandb offline (no needed for generation...)
# read Huggingface token from .env file
export HF_TOKEN=$(python -c "import huggingface_hub; print(huggingface_hub.HfFolder.get_token() or '')")

# execute the python script
python /path/to/your/python/script/my_fancy_generation.py
```

Look at your output file to see the generated text!

Llama Factory

Llama Factory

Llama Factory is a ready-to-use library for LLM training

You could use it to instruction-tune a LLM with new conversations

https://github.com/hiyouga/LLaMA-Factory

To install:

```
$ git clone --depth 1 https://github.com/hiyouga/LLaMA-Factory.git
$ cd LLaMA-Factory
$ source /path/to/env/bin/activate
$ pip install -e ".[torch,metrics]" --no-build-isolation
```

Llama Factory

After installed, run:

```
$ llamafactory-cli train /path/to/config.yaml
```

Inside a slurm script.

Llama Factory: Config

examples > train_full > ! sft_example_blank.yaml

```yaml
### model
model_name_or_path: path_of_model # no hf_id but the absolute path, save it in a specific directory
new_special_tokens: <|start_header_id|>,<|end_header_id|>,<|eot_id|> # token added with the chat template

### method
stage: sft # Supervised Fine-Tuning
do_train: true
finetuning_type: full
use_badam: true # use BADAM not classical full-finetuning, faster!
badam_mode: layer
badam_switch_mode: ascending
badam_switch_interval: 50
badam_verbose: 2
flash_attn: fa2
deepspeed: examples/deepspeed/ds_z3_config.json # deepspeeed

### dataset
dataset: magpielm-v0.1 # comma separated datasets, defined in: data/dataset_info.json
template: llama3 # chat_template
cutoff_len: 2048 #
max_samples: 560000
overwrite_cache: true
preprocessing_num_workers: 16

### output
output_dir: /path/to/output # output directory
logging_steps: 10
save_steps: 500
plot_loss: true
overwrite_output_dir: true

### train
per_device_train_batch_size: 4
gradient_accumulation_steps: 8
learning_rate: 1.0e-5
num_train_epochs: 1.0
lr_scheduler_type: cosine
warmup_ratio: 0.05
bf16: true
ddp_timeout: 180000000

### eval
val_size: 0.01
per_device_eval_batch_size: 4
eval_strategy: steps
eval_steps: 100

### logging
report_to: wandb
run_name: run_name # the name of the run
```

Llama Factory: Config

examples > train_full > ! sft_example_blank.yaml

```yaml
### model
model_name_or_path: path_of_model # no hf_id but the absolute path, save it in a specific directory
new_special_tokens: <|start_header_id|>,<|end_header_id|>,<|eot_id|> # token added with the chat template

### method
stage: sft # Supervised Fine-Tuning
do_train: true
finetuning_type: full
use_badam: true # use BADAM not classical
badam_mode: layer
badam_switch_mode: ascending
badam_switch_interval: 50
badam_verbose: 2
flash_attn: fa2
deepspeed: examples/deepspeed/ds_z3_confi

### dataset
dataset: magpielm-v0.1 # comma separated
template: llama3 # chat_template
cutoff_len: 2048 #
max_samples: 560000
overwrite_cache: true
preprocessing_num_workers: 16

### output
output_dir: /path/to/output # output directory
logging_steps: 10
save_steps: 500
plot_loss: true
overwrite_output_dir: true
```

```yaml
### train
per_device_train_batch_size: 4
gradient_accumulation_steps: 8
learning_rate: 1.0e-5
num_train_epochs: 1.0
lr_scheduler_type: cosine
warmup_ratio: 0.05
: true
_timeout: 180000000

eval
_size: 0.01
_device_eval_batch_size: 4
_strategy: steps
_steps: 100

logging
rt_to: wandb
_name: run_name # the name of the run
```

The *chat_template* is the text structure used by the chat models.

Starting from a *base models* we should add to the vocabulary the *special tokens* used to structure the *chat_template*.

Add your custom dataset

To add a conversation dataset, you can load it from HF, and add it to the `data/dataset_info.json` LlamaFactory file

```
"conversations-it": {
  "hf_hub_url": "sapienzanlp/conversations_italian",
  "formatting": "sharegpt",
  "columns": {
    "messages": "messages"
  },
  "tags": {
    "role_tag": "role",
    "content_tag": "content",
    "user_tag": "user",
    "assistant_tag": "assistant",
    "system_tag": "system"
  }
},
```

Create your custom dataset

Once you have collected your beautiful and useful conversations you can upload them to your HF profile

Make sure you follow a standardized format. We suggest you to use the **SHAREGPT** format.

Create your custom dataset

Once you have collected your beautiful and useful conversations you can upload them to your HF profile

Make sure you follow a standardized format. We suggest you to use the **SHAREGPT** format.

Each conversation is encoded in a list field named “messages”.
The “messages” field contains a list of dictionaries with user and assistant (LLM) messages.

Create your custom dataset

Once you have collected your beautiful and useful conv
to your HF profile

Make sure you follow a standardized format. We sugges
format.

Each conversation is encoded in a list field named “mes
The “messages” field contains a list of dictionaries wit
messages.

```
messages
list
```


```
[
{
"content": "These instructions apply to
section-based themes (Responsive 6.0+, Retina
4.0+, Parallax 3.0+ Turbo 2.0+, Mobilia 5.0+).
What theme version am I using?\nOn your
Collections pages & Featured Collections
sections, you can easily show the secondary
image of a product on hover by enabling one of
the theme's built-in settings!\nYour
Collection pages & Featured Collections
sections will now display the secondary
product image just by hovering over that
product image thumbnail.\nDoes this feature
apply to all sections of the theme or just
specific ones as listed in the text
material?",
"role": "user"
},
{
"content": "This feature only applies to
Collection pages and Featured Collections
sections of the section-based themes listed in
the text material.",
"role": "assistant"
```

wandb

wandb is an useful tool to monitor your ML training

Some of you already used it in the first homework

you have to install it in your environment: `pip install wandb`

Create an account on wandb.ai then login locally by `wandb login`

wandb will be used offline by the compute nodes (see the slurm script) you have to sync the run online after the execution: `wandb sync path_to_run`