First AI training job on LUMI

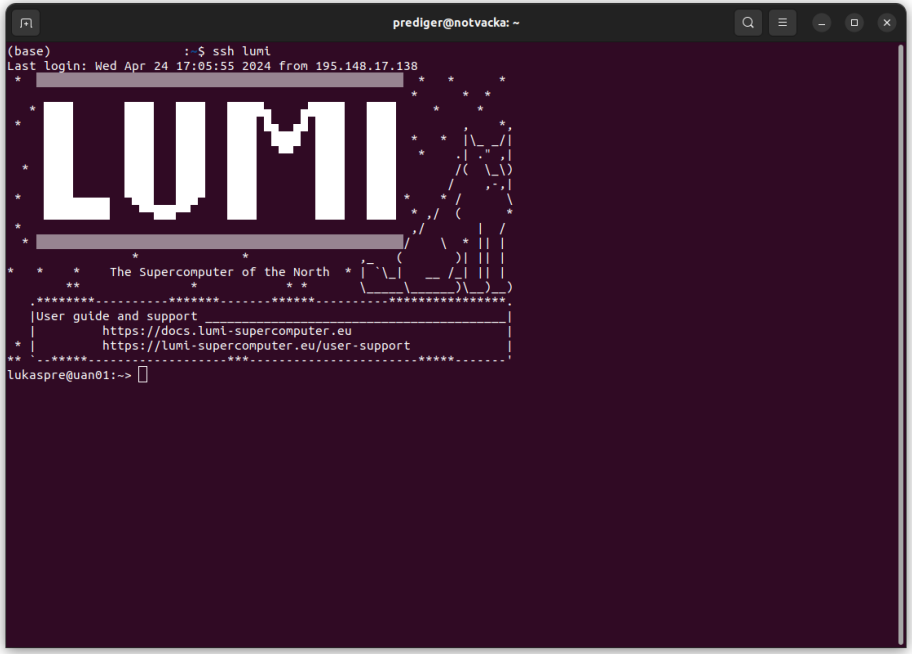
Mats Sjöberg, Lukas Prediger – CSC – IT Center for Science, Finland

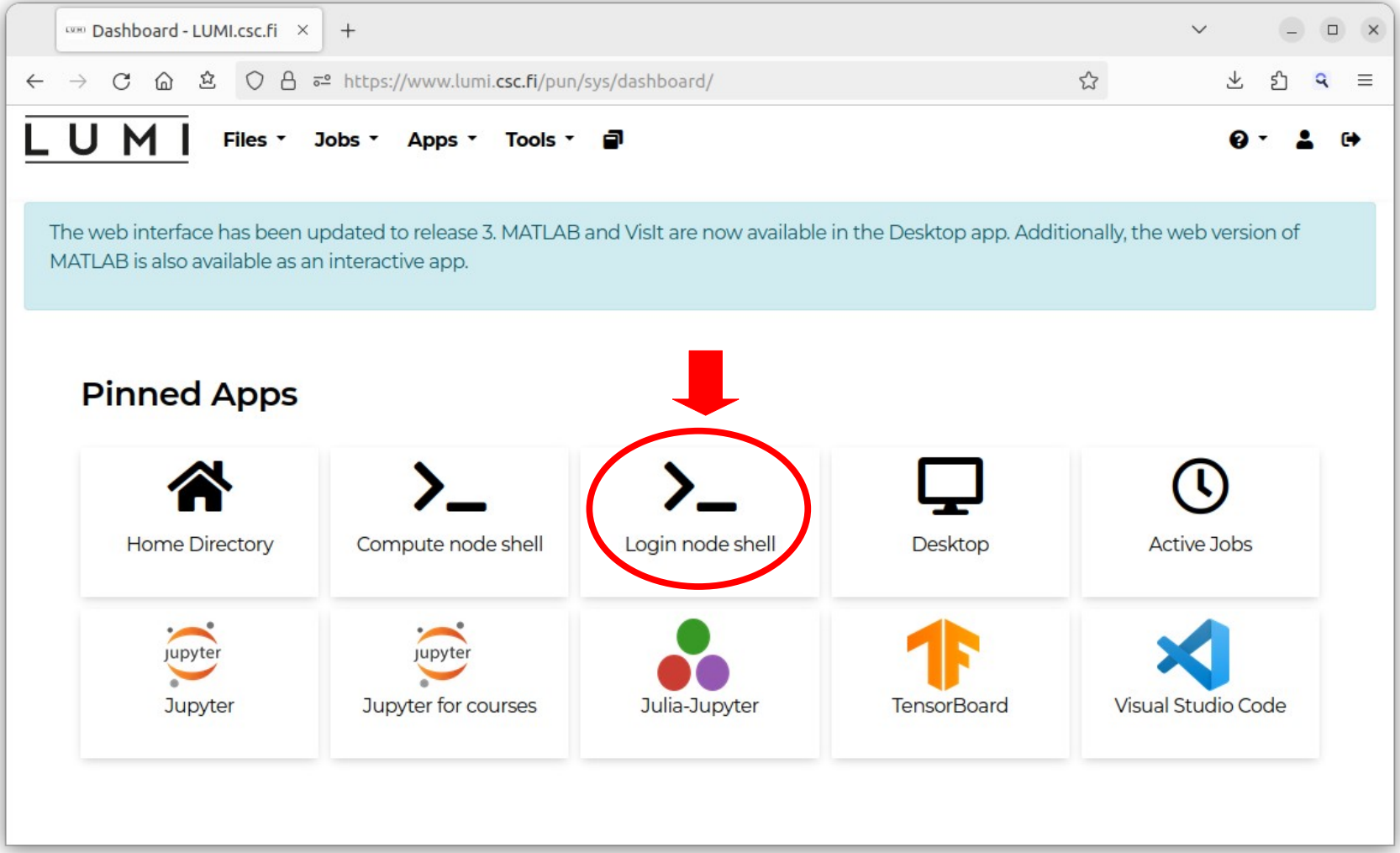
Accessing LUMI via SSH

- No passwords
  → SSH key pair **required**
- register your public key in your MyAccessID user profile

```
ssh -i <path-to-private-key> \
       <username>@lumi.csc.fi
```

https://docs.lumi-supercomputer.eu/firststeps/




Dashboard - LUMI.csc.fi
https://www.lumi.csc.fi/pun/sys/dashboard/
LUMI
Files
Jobs
Apps
Tools
The web interface has been updated to release 3. MATLAB and VisIt are now available in the Desktop app. Additionally, the web version of MATLAB is also available as an interactive app.
Pinned Apps
Home Directory
Compute node shell
Login node shell
Desktop
Active Jobs
Jupyter
Jupyter for courses
Julia-Jupyter
TensorBoard
Visual Studio Code

Setting up the Software Environment

https://pytorch.org/

Official PyTorch ROCM version does not currently work for LUMI

environment-minimal.yml

```yaml
channels:
  - pytorch
  - conda-forge
  - defaults
dependencies:
  - accelerate=0.29.1
  - datasets=2.18.0
  - python=3.10.14
  - pytorch=2.2.2
  - transformers=4.39.3
```

~47k files

↓

bad for the shared network filesystem

Container icon: Iconjam @ flaticon.com

https://docs.lumi-supercomputer.eu/software/

Setting up the Software Environment

LUMI

- Singularity/Apptainer containers recommended for ML applications on LUMI
- Similar to Docker, better suited for multi-user supercomputers
  - Isolated environment
  - Easier to manage complex software dependencies
- For now we'll provide a pre-installed container
  - Later lectures will discuss how to create your own containers

`python`  `srun singularity exec $CONTAINER python`

Setting up the Software Environment

Set up the software environment

```
$ module purge
$ module use /appl/local/training/modules/AI-20240529/
$ module load singularity-userfilesystems singularity-CPEbits
```

Modules should be available from base system after July 2024

Launch the PyTorch container

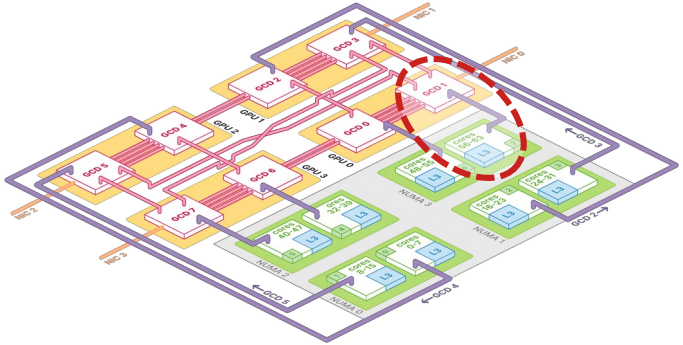
```
$ CONTAINER=/scratch/project_465001063/containers/pytorch_transformers.sif
$ srun singularity exec $CONTAINER python
```

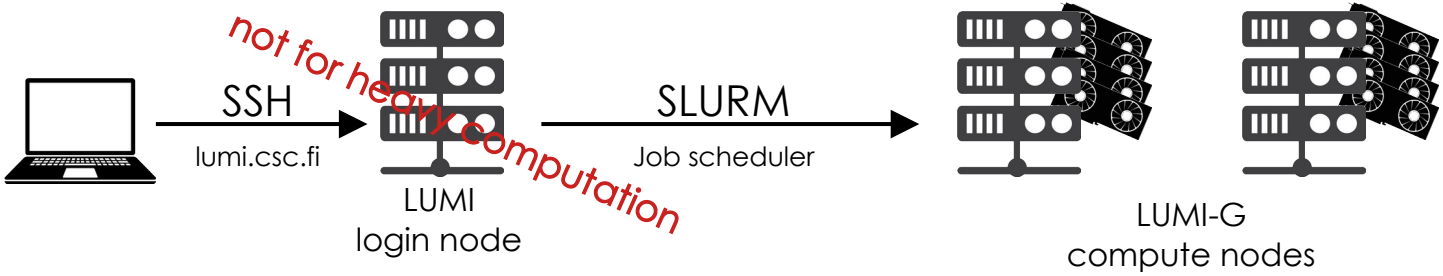
https://docs.lumi-supercomputer.eu/software/
https://docs.lumi-supercomputer.eu/runjobs/lumi_env/Lmod_modules/

Don't run computation on the login node – use the Slurm job scheduler!

LUMI

```
$ python my_pytorch_script.py
```

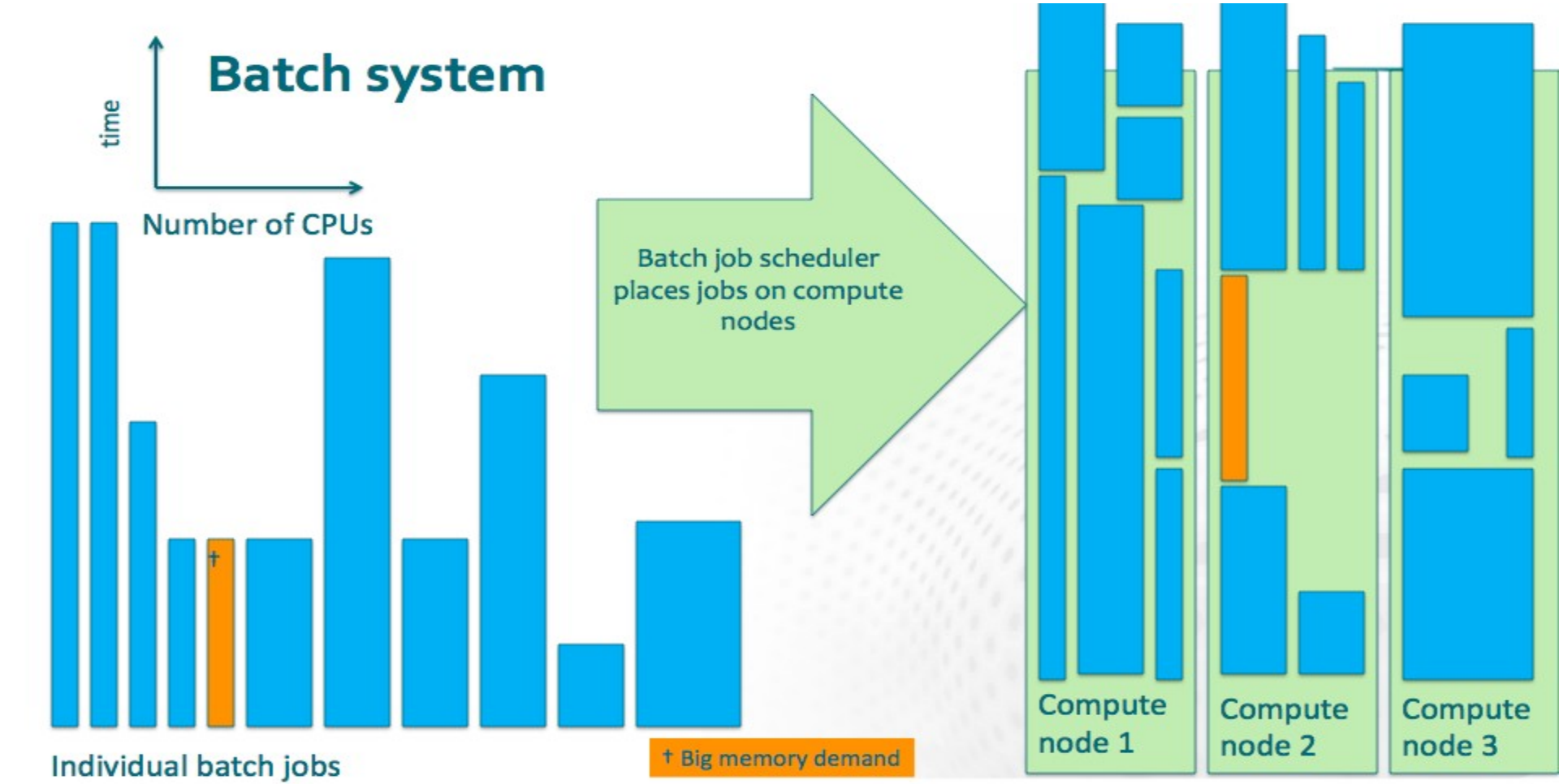




SLURM is used to reserve resources and submit scripts for running on the compute nodes

Icons: vecteezy.com

https://docs.lumi-supercomputer.eu/runjobs/


Batch system
time
Number of CPUs
†
Individual batch jobs
Batch job scheduler places jobs on compute nodes
† Big memory demand
Compute node 1
Compute node 2
Compute node 3

Training on a single Graphics Compute Die

SLURM batch script (run.sh)

```
#!/bin/bash
#SBATCH --account=project_123456
#SBATCH --partition=small-g
#SBATCH --gpus-per-node=1
#SBATCH --ntasks-per-node=1
#SBATCH --cpus-per-task=7
#SBATCH --mem-per-gpu=60G
#SBATCH --time=1:00:00

module purge
module use /appl/local/training/modules/AI-20240529/
module load singularity-userfilesystems singularity-CPEbits
CONTAINER=/scratch/project_465001063/containers/pytorch_transformers.sif

srun singularity exec $CONTAINER python my_pytorch_script.py
```

What resources requested?

What software to load?

Good rule-of-thumb:

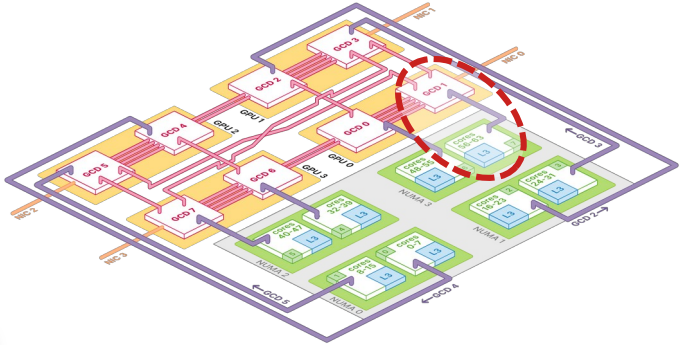
- 1 GPU = 1/8 of node
- Use also ≤ 1/8 of CPU cores and memory

Available GPU partitions

- **standard-g**
  ≤ 48h, whole nodes only, max 1024 nodes/job
- **small-g**
  ≤ 72h, individual GCDs, max 4 nodes/job
- **dev-g**
  ≤ 3h, individual GCDs, max 32 nodes/job, *max 1 job running*

https://docs.lumi-supercomputer.eu/runjobs/scheduled-jobs/partitions/

Submit the job to the queue



Submit the SLURM batch script

```
$ sbatch run.sh
Submitted batch job 987654
```

https://docs.lumi-supercomputer.eu/runjobs/scheduled-jobs/slurm-quickstart/

Useful SLURM commands for monitoring job progress

Check the SLURM queue

```
$ squeue –me
JOBID    PARTITION NAME    ST TIME   NODES NODELIST
987654   small-g   run.sh  R  0:18       1 compute_node
```

Check GPU utilization

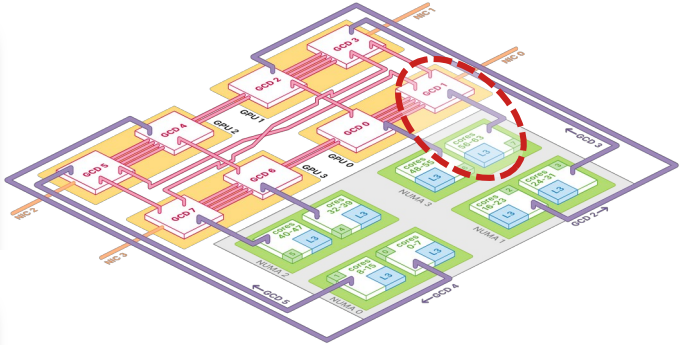
```
$ srun --overlap --pty --jobid=987654 bash
@compute_node$ rocm-smi
```

Read job outputs

```
$ tail -f slurm-987654.out
```

Cancel a job

```
$ scancel 987654
```



https://docs.lumi-supercomputer.eu/runjobs/scheduled-jobs/slurm-quickstart/

Useful SLURM commands for monitoring job progress

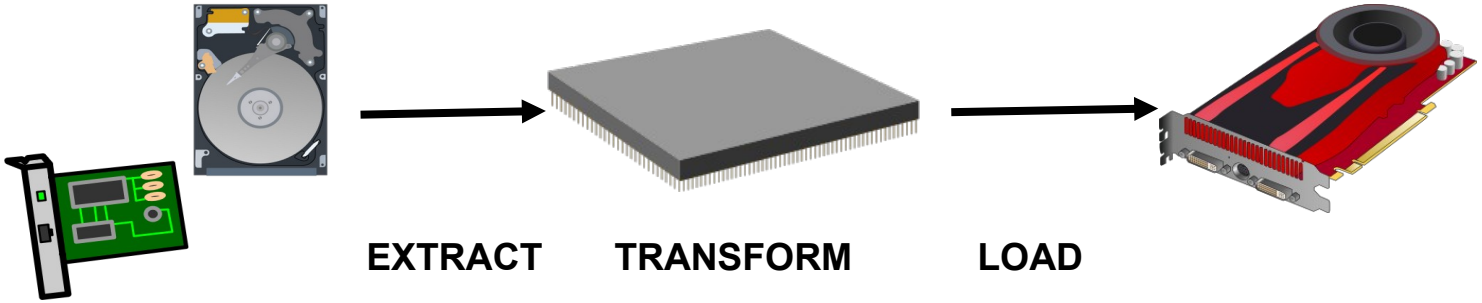
LUMI

Check the status of partitions

A = allocated
I = idle
O = other states
T = total

```
$ sinfo -s
PARTITION   AVAIL  TIMELIMIT   NODES(A/I/O/T) N                    
debug          up      30:00          8/0/0/8 nid[                002595-002597]
interactive    up    8:00:00          4/0/0/4 nid[002502,002507,002594,002599]
q_fiqci     inact      15:00          0/1/0/1 nid002598
q_industry     up      15:00          0/1/0/1 nid002598
q_nordiq       up      15:00          0/1/0/1 nid002503
small          up 3-00:00:00     280/6/20/306 nid[002280-002499,002508-002593]
standard       up 2-00:00:00 1493/111/124/172 nid[001000-002279,002600-003047]
dev-g          up    3:00:00       30/17/1/48 nid[005002-005025,007954-007977]
small-g        up 3-00:00:00      193/2/3/198 nid[005026-005123,007852-007951]
standard-g     up 2-00:00:00 2555/23/150/2728 nid[005124-007851]
largemem       up 1-00:00:00          1/5/0/6 nid[000101-000106]
lumid          up    4:00:00          1/6/1/8 nid[000016-000023]
```

Using multiple CPUs for ETL



- Reserve enough CPU cores per GPU, 7 cores/GPU on LUMI

```
#SBATCH --cpus-per-task=7
```

- Use multiple workers (processes) in PyTorch DataLoader:

```
train_loader = torch.utils.data.DataLoader( ... ,
  num_workers=N)
```

Use checkpointing!

Checkpointing = save the model weights every now and then

- E.g. after each epoch, or every N batches
- If your run crashes, simply restart from the latest checkpoint
- Train for longer than the “short” 2-3 days limit on LUMI partitions
- Exploit scheduling “backfilling”

HuggingFace Trainer example

```
training_args =
  TrainingArguments(
    ... ,
    overwrite_output_dir=False,
    ...
  )

trainer = Trainer( ... )
trainer.train(
  resume_from_checkpoint=True
)
```