

# FROM A NOTEBOOK TO EXPERIMENTS ON A CLUSTER

## DATA LAB — MASTER IASD

**Alexandre Vérine**

Université Paris Dauphine – PSL / ENS – PSL

8 October 2026



WHERE ARE YOU NOW?

**Where are you now?**

---

## THREE THINGS TO CHECK TODAY

### **Identity**

Complete Name / Mail  
in the GitHub Sheet

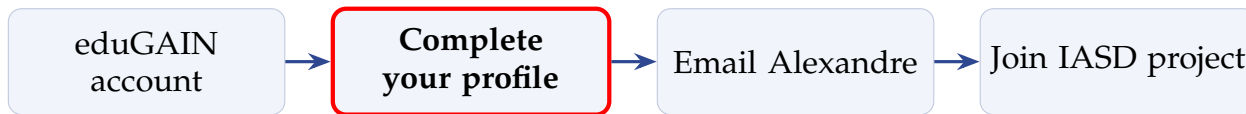
### **Your group**

Check “Find  
your A1 group”  
Exactly three students

### **Compute access**

Check the MesoNET  
follow-up Sheet

## MESONET: AN ACCOUNT IS ONLY THE FIRST STEP



alexandre.verine@dauphine.psl.eu

**Live:** I will show you the teacher's portal and why an incomplete profile prevents me from adding you.

# WHERE ARE YOUR COMPUTE HOURS?



Resource portal

## Liste des projets

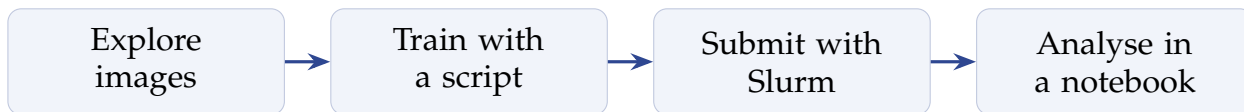
Project membership, machine account,  
SSH key and compute allocation.

**Live walkthrough** of the IASD project.  
Students use their own accounts.

The screenshot shows the MesoNET website interface. The top navigation bar includes links for 'Documentation', 'Portail', and 'Site web'. The sidebar on the left contains a list of links: 'Accueil', 'Le portail MesoNET', 'Machine Code/Formation', 'Les machines', 'Juliet', 'Description', 'Se connecter à Juliet', 'Le stockage', 'Lancer un calcul', 'Environnements logiciels', 'Réserver des noeuds de calcul sur Juliet', 'Troubleshooting', 'Utiliser VS code server', 'Faire une demande de formation sur Juliet', 'Utiliser les noeuds de visualisation Juliet visu', 'Conditions Générales d'Utilisation des ressources de calcul de Juliet', 'Vesta', 'Zen', and 'Phileas'. The main content area is titled 'Juliet, machine IA/GPU' and contains the following text: 'La machine Juliet du projet MesoNET est dédiée aux projets scientifiques, techniques et de formation nécessitant des GPUs.', 'Description de la machine', 'Se connecter à juliet', 'Lancer un calcul', 'Pour les formations', and 'Problèmes fréquents'. A light blue box contains the section 'EN CAS DE PROBLÈME' with the following text: 'Pour les utilisateurs non-académiques (entreprises et collectivités), vous pouvez solliciter le support de l'équipe ROMEO ici', 'Pour les utilisateurs académiques (universitaires, étudiants et instituts de recherche) vous pouvez ouvrir un ticket ou écrire un mail à support@mesonet.fr', and 'Vous pouvez également regarder les solutions aux problèmes les plus courants'. The bottom of the page features navigation links: 'Previous', '« Les machines', 'Next', and 'Description »'.

Public MesoNET documentation, captured 7 October 2026.

## ONE PROJECT, FROM IMAGES TO A CLUSTER



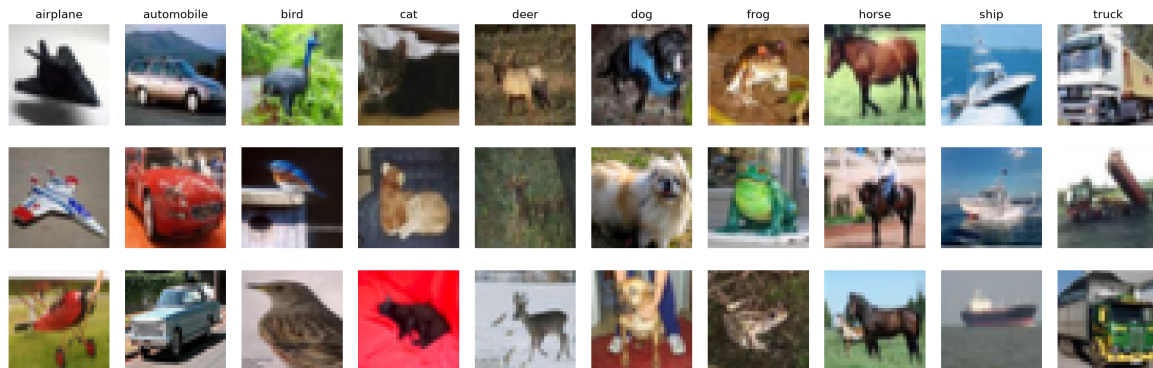
Two hours. One small image classifier. The same experiment at every step.

BUILD EXPERIMENTS YOU CAN EXPLAIN

**Build experiments you can explain**

---

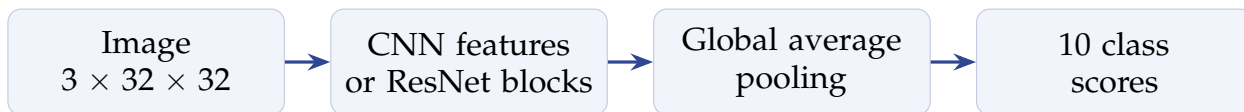
## OUR EXAMPLE: CLASSIFY CIFAR-10 IMAGES



Small CNN vs ResNet-18 adapted to  $32 \times 32$  images.

CIFAR-10: Alex Krizhevsky, Vinod Nair and Geoffrey Hinton. Images from the training split.

## TWO ARCHITECTURES, THE SAME EXPERIMENTAL QUESTION



### Small CNN

3 convolutional layers. Fast baseline. Easy to inspect.

### ResNet-18

Residual blocks; CIFAR-sized input stem; 10 outputs.  
Train from scratch.

“IT WORKED IN MY NOTEBOOK...”

```
In [8]: learning_rate = 0.01  
In [2]: model = make_model()  
In [15]: train(model)  
In [4]: plot(history)
```

Hidden state

Which model? Which configuration?  
Can a fresh kernel reproduce this?

Illustrative notebook excerpt. A simple notebook is accepted in A1; code itself is not graded.

## A NOTEBOOK KERNEL KEEPS MORE THAN YOU THINK

### Memory persists

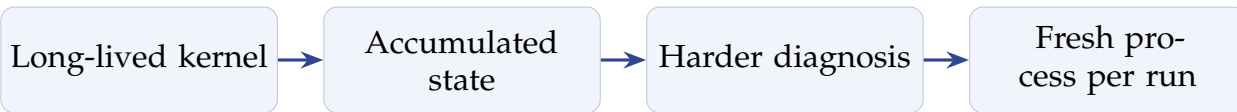
Old models, tensors and outputs can remain referenced across cells. Rerunning a cell does not reset the process. Restart the kernel for a clean state.

### Execution order matters

A saved notebook may depend on hidden state. Restart + Run All, fixed inputs, configs and environment are needed to reproduce it.

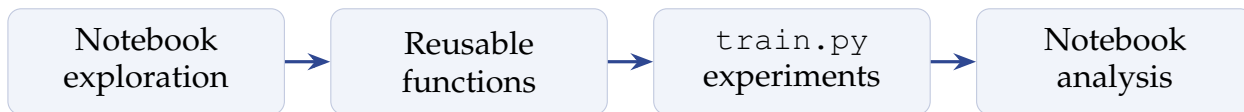
### Laptop-dependent training

If training runs on your laptop, sleep or shutdown interrupts it. A remote kernel may continue; a Slurm job is independent of your browser.



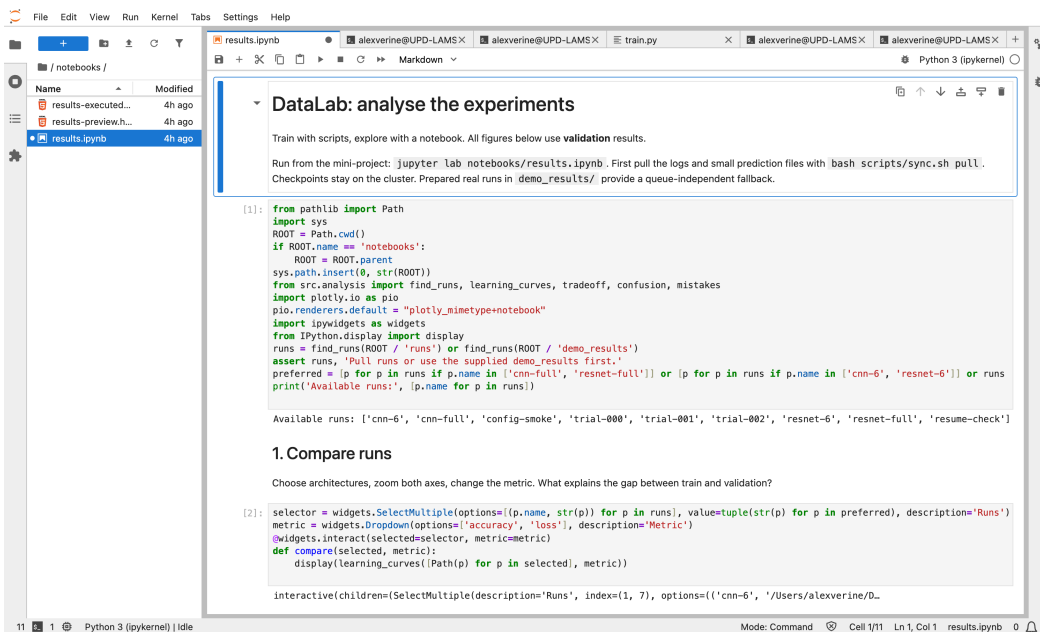
Notebook risks concern state and lifecycle, not an inherent slowdown of the same PyTorch operations.

KEEP THE NOTEBOOK. GIVE EXPERIMENTS A SCRIPT.



Twenty configurations should not require twenty copied notebooks.

# OUR NOTEBOOK STAYS FOCUSED ON ANALYSIS



Actual JupyterLab notebook: load saved metrics and predictions; training is launched by a separate script.

## A SMALL REPO WITH CLEAR BOUNDARIES

```
mini_project/  
  train.py  
  src/models.py  
  configs/  
  scripts/train.sbatch  
  notebooks/results.ipynb  
  requirements.txt  
  README.md
```

### **Version in Git**

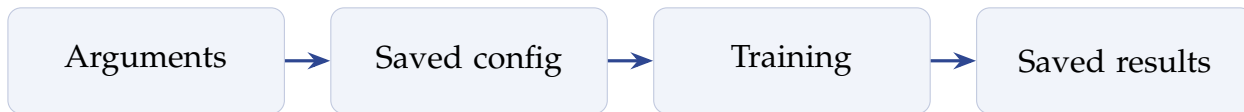
Code, configs,  
launch scripts, docs

### **Keep outside Git**

Data, environments,  
logs, large checkpoints

## ONE COMMAND, ONE CONFIGURATION, ONE RUN

```
python train.py --model small_cnn --lr 0.001 \  
--seed 42 --epochs 6 --run-dir runs/cnn-001
```



Use a new run directory for a new experiment. Keep the seed and software versions.

The same seed is useful, but does not guarantee identical results on different hardware.

## THERE ARE MANY WAYS TO CONFIGURE A RUN

### argparse: standard library

```
p = argparse.ArgumentParser()
p.add_argument("--lr", type=
    float,
                default=0.001)
args = p.parse_args()
```

### Click: decorators

```
@click.command()
@click.option("--lr", type=float,
              default=0.001)
def train(lr):
    ...
```

### YAML / JSON: saved settings

```
model: resnet18
lr: 0.001
seed: 42
epochs: 12
```

**Our example:** argparse + JSON config. CLI values override the config. Save the final resolved settings with every run. YAML needs a parser (e.g. PyYAML's safe loader); Click is a separate dependency. Hydra/OmegaConf and other tools can help with larger projects. Choose one convention and document it.

## ARGPARSE: TURN A SCRIPT INTO A REUSABLE EXPERIMENT

```
import argparse
p = argparse.ArgumentParser()
p.add_argument("--model", choices=["small_cnn", "resnet18"])
p.add_argument("--lr", type=float, default=0.001)
p.add_argument("--epochs", type=int, default=12)
p.add_argument("--run-dir", required=True)
p.add_argument("--tensorboard", action="store_true")
args = p.parse_args()
# Use args.lr, args.epochs, args.run_dir, ...
```

```
python train.py --model resnet18 --lr 0.001 --epochs 12 \
--run-dir runs/resnet-001 --tensorboard
```

Simplified teaching excerpt. Save the resolved configuration, including defaults, with the results.

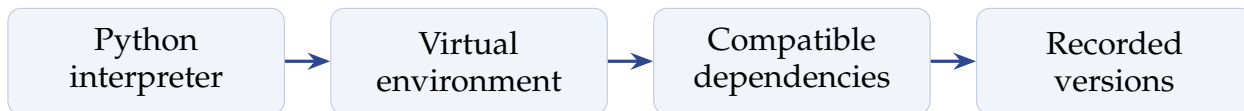
`-help`: document options.

`type`: convert and validate input.

`choices`: reject unknown models.

`store_true`: a Boolean flag.

## YOUR ENVIRONMENT IS PART OF THE EXPERIMENT



```
python3 --version
python3 -m venv .venv
source .venv/bin/activate
python -m pip install -r requirements.txt
python -m pip freeze > environment.txt
```

**A1 lesson:** old Torch pins and a new Python can be incompatible; missing packages such as `tqdm` must be declared.

## CONDA: MANAGE PYTHON AND PACKAGES TOGETHER

```
conda create -n cifar-demo python=3.11 pip
conda activate cifar-demo
python -m pip install -r requirements.txt
conda env export > environment.yml
# Recreate the saved environment:
conda env create -f environment.yml
```

**venv**: isolates packages for an existing Python interpreter.

**conda**: manages the interpreter and packages, including native dependencies.

Install conda packages first, then pip packages. Record versions and channels.

Choose one environment for a run. Verify which `python` and `python -version` after activation. A saved environment may still need adaptation to another OS or GPU stack.

## HPC MODULES: SELECT THE SERVER'S SOFTWARE STACK

```
module avail
module show Python/EXAMPLE_VERSION
module load Python/EXAMPLE_VERSION
module list
python --version
python -m venv .venv
source .venv/bin/activate
```

**Environment Modules / Lmod** expose software installed by the cluster team.

A module adjusts paths and libraries; it is not your project's isolated environment.

Load the same modules in the batch script, then activate your venv or conda environment.

Module names are cluster-specific: replace `Python/EXAMPLE_VERSION`. Follow the site's documentation for Python, CUDA and containers. Our Juliet demo uses a venv; `module` is not available in the tested session.

## EVERY RUN LEAVES EVIDENCE

```
runs/cnn-001/  
  config.json  
  environment.txt  
  train.log  
  metrics.jsonl  
  tensorboard/  
  last.pt  
  best.pt  
  predictions.npz  
  summary.json
```

Command + config + job ID



Curve + result

What changed, what was measured, and which model produced the prediction?

A log explains events. Structured metrics allow comparison. A checkpoint allows recovery.

## A LOGGER RECORDS EVENTS; A METRIC RECORDS A MEASUREMENT

```
import logging
logging.basicConfig(
    level=logging.INFO,
    format="%(asctime)s %(levelname)s %(message)s",
    handlers=[logging.StreamHandler(),
              logging.FileHandler("train.log")])
log = logging.getLogger(__name__)
log.info("epoch=%d val_accuracy=%.3f", epoch, acc)
```

Logger: time +  
level + message



Console / Slurm log + file

Measurements: JSONL  
/ TensorBoard / W&B  
Structured metrics for plots

Use DEBUG for diagnostics, INFO for progress, WARNING for unexpected conditions and ERROR for failures.

`log.exception(...)` records the traceback inside an exception handler.

Our console handler writes to `stderr`: progress is also visible in Slurm's `.err` file; the application keeps `train.log`.

## WRITE OUTPUTS THAT ANOTHER PROGRAM CAN READ

```
from pathlib import Path
import json
root = Path(args.run_dir)
root.mkdir(parents=True, exist_ok=False)
(root / "config.json").write_text(
    json.dumps(vars(args), indent=2))
# At the end of each epoch:
row = {"epoch": epoch, "val_accuracy": acc,
       "val_loss": loss, "seconds": elapsed}
with (root / "metrics.jsonl").open("a") as f:
    f.write(json.dumps(row) + "\n")
```

Minimal standalone pattern, not the entire training script. The actual demo also records environment, source files and job ID.

**config.json:** one complete configuration.

**metrics.jsonl:** one JSON record per epoch,  
easy to append and parse.

**train.log:** events and diagnosis.

**last.pt / best.pt:** resume / evaluate.

## THE SAME METRICS CAN FEED DIFFERENT LOGGERS

```
from torch.utils.tensorboard import SummaryWriter
writer = SummaryWriter(root / "tensorboard")
writer.add_scalar("val/accuracy", acc, epoch)
writer.add_scalar("val/loss", loss, epoch)
writer.close() # after training
```

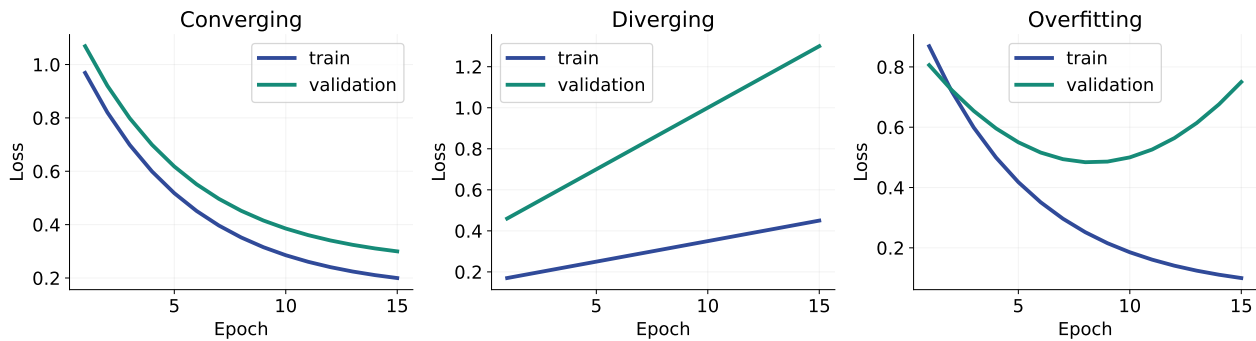
```
tensorboard --logdir runs
```

```
import wandb
run = wandb.init(project="cifar-demo",
                 mode="offline", config=vars(args),
                 dir=str(root))
run.log({"epoch": epoch,
        "val_accuracy": acc})
run.finish() # after training
```

**One measurement, several destinations.** Log scalars each epoch; close the writer after the loop. Keep local JSONL results even if a tracking service is unavailable.

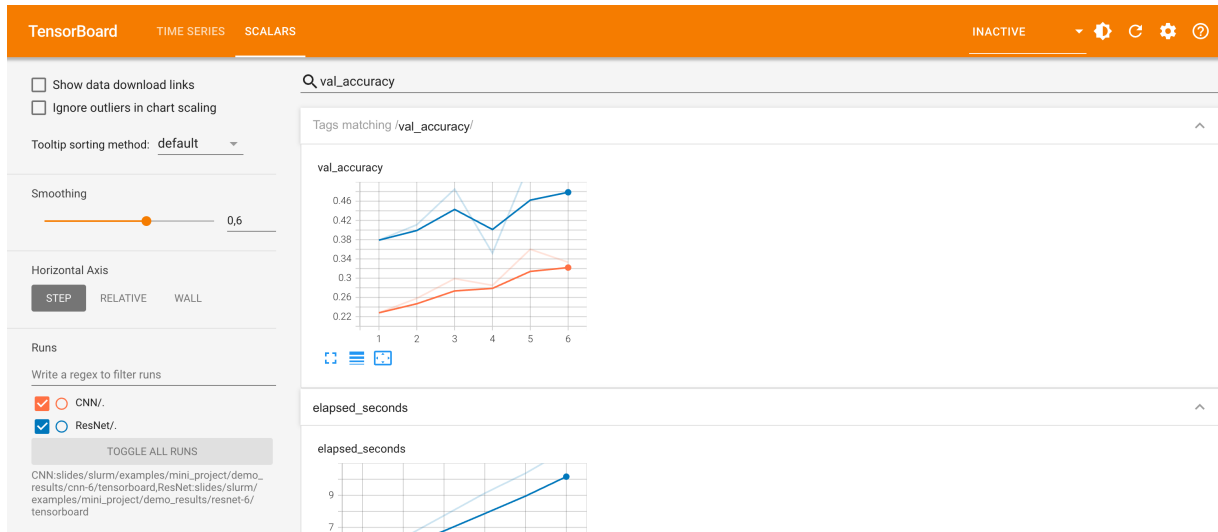
Illustrative integration excerpts; TensorBoard and W&B require their own dependencies. No hosted W&B account is needed in offline mode.

## READ THE CURVES BEFORE CELEBRATING



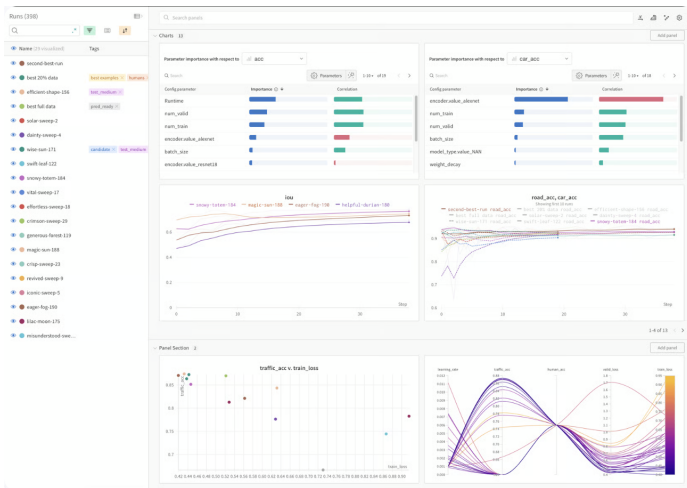
Illustrative curves, not measured results.

# TENSORBOARD: COMPARE EXPERIMENTS LOCALLY



Actual TensorBoard dashboard: validation accuracy for our two short CIFAR-10 runs.

## WEIGHTS & BIASES: SHARED EXPERIMENT TRACKING



Official W&B documentation; interface example, not our hosted results.

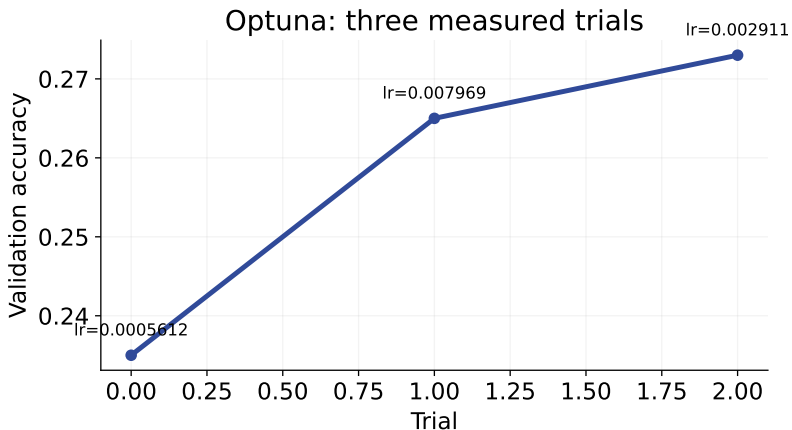
## Configs + metrics + run comparisons.

Our example logs **offline** on the cluster.

```
wandb.init(mode="offline", ...)
run.log({"val_accuracy": acc})
```

Sync later only if you want a hosted dashboard. No API key needed for our offline demo.

## OPTUNA CHOOSES TRIALS; YOU INTERPRET RESULTS



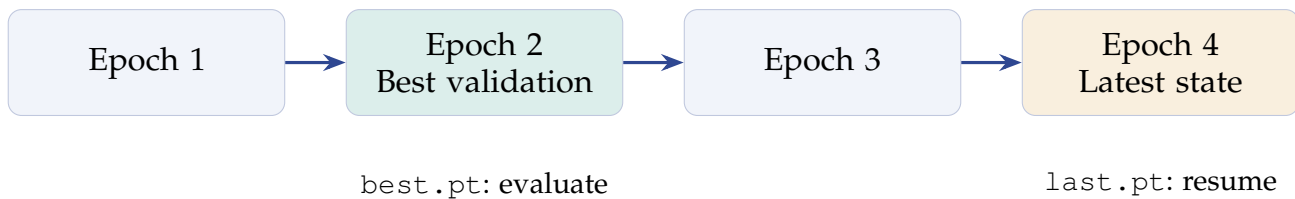
1. **Suggest:** learning rate sampled on a log scale, from  $10^{-4}$  to  $10^{-2}$ .
2. **Evaluate:** train the same CNN for two epochs; return validation accuracy.
3. **Adapt:** TPE uses previous trials to propose the next setting.

Our budget: three trials, fixed data split and run seed. Save parameters, scores and the study; a single winner is not evidence of a robust gain.

**Tracking  $\neq$  tuning.** Tuning can change multiple settings; causal conclusions need controlled comparisons. Never tune on the test set.

Measured small study. Pruning can stop unpromising trials; our minimal example does not implement it.

## CHECKPOINTS: LAST IS NOT BEST



Resume needs model, optimizer, epoch and relevant random states. Best is selected on **validation**.

TERMINAL, SSH AND A COMFORTABLE WORKFLOW

# Terminal, SSH and a comfortable workflow

---

## A TERMINAL IS JUST ANOTHER WAY TO WORK WITH FILES

```
pwd                # current directory
ls                 # its contents
cd mini_project    # change directory
python train.py --help # available options
```

Terminal = application

Shell = command interpreter

**Mac/Linux:** Terminal. **Windows:** PowerShell for SSH; WSL/Linux shell for the Bash and rsync scripts.

Tab completes names. Up recalls commands. `Ctrl+C` interrupts a command.

## LOCAL: INSPECT THE PROJECT AND ITS CONFIGURATION

```
DataLab · local terminal → Juliet Live shell - real SSH commands

configs/:
resnet18.json
small_cnn.json

notebooks/:
results-executed.html
results-preview.html
results.ipynb

scripts/:
check_resume.sbatch
pull.sh
push.sh
submit.sh
sync-target.sh
sync.sh
train.sbatch
tune.sbatch

src/:
__init__.py
pycache
analysis.py
models.py
{
  "model": "resnet18",
  "epochs": 12,
  "train_size": 45000,
  "val_size": 5000,
  "lr": 0.001,
  "batch_size": 128,
  "seed": 42
}
local $
```

Actual local shell in a live xterm.js terminal: pwd, project files and JSON configuration. The shell is the same kind of shell used in Terminal.

## REMOTE: CHECK THE MACHINE, ENVIRONMENT AND SCRIPT OPTIONS

DataLab · local terminal → Juliet    Live shell · real SSH commands

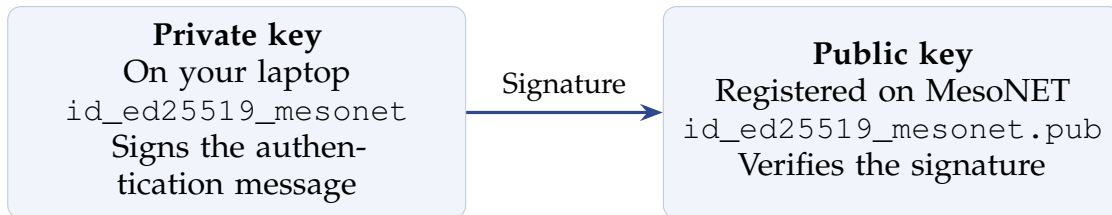
```
(.venv) juliet $ hostname; python --version; python train.py --help | head -22
juliet1
Python 3.9.25
usage: train.py [-h] [--config CONFIG] [--model {small_cnn,resnet18}]
               [--data-dir DATA_DIR] --run-dir RUN_DIR [--epochs EPOCHS]
               [--train-size TRAIN_SIZE] [--val-size VAL_SIZE]
               [--batch-size BATCH_SIZE] [--lr LR] [--seed SEED]
               [--device {auto,cpu,cuda,mps}] [--workers WORKERS]
               [--threads THREADS] [--resume] [--tensorboard] [--wandb]

CIFAR-10 teaching experiment: script launches; notebook analyses.

optional arguments:
  -h, --help            show this help message and exit
  --config CONFIG        JSON defaults, overridden by explicit CLI arguments
  --model {small_cnn,resnet18}
  --data-dir DATA_DIR
  --run-dir RUN_DIR
  --epochs EPOCHS        Total epochs, including resumed epochs
  --train-size TRAIN_SIZE
  --val-size VAL_SIZE
  --batch-size BATCH_SIZE
  --lr LR
  --seed SEED
  --device {auto,cpu,cuda,mps}
(.venv) juliet $
```

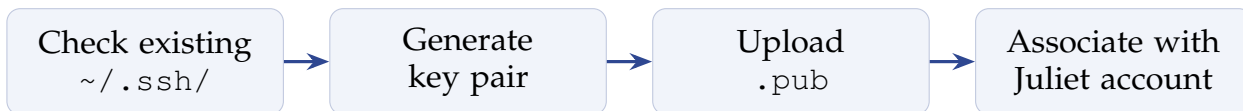
Actual SSH command on Juliet: `hostname`, Python version and `train.py -help`. These checks tell you where you are and what you can launch.

## AN SSH KEY PAIR: PROVE WHO YOU ARE WITHOUT SHARING THE SECRET



- ▶ The server verifies that you own the matching private key; the private key is never sent.
- ▶ Share or upload only the `.pub` file. A passphrase protects the private key on your laptop.
- ▶ Registering a key does not grant project access: Alexandre must also add your account to the IASD project.

## SSH: KEEP THE PRIVATE KEY; REGISTER THE PUBLIC KEY



```
ls ~/.ssh
# Use a new name; do not overwrite an existing key.
ssh-keygen -t ed25519 -f ~/.ssh/id_ed25519_mesonet
cat ~/.ssh/id_ed25519_mesonet.pub
```

Use a passphrase. Never share the private file. Alexandre must first have added your account to the IASD project. Add the **public** key to the portal, then associate it with the project's Juliet account and accept the machine's terms.

## THE INSTRUCTIONS ARE PUBLIC AND ILLUSTRATED



The screenshot shows a web interface for managing SSH keys. At the top, there is a blue button labeled "Ajouter une clé ssh". Below it, a blue box contains instructions: "Veuillez faire un copier-coller depuis votre fichier de clé publique ssh" and "Algorithmes autorisés: RSA-2048 ED25519-256". On the left, there is a tab labeled "Clés ssh". Below the instructions, there is a form with two fields: "Nom de la clé:" with a dropdown menu showing "choisissez un nom", and "Votre clé publique ssh" with a text input field containing a long SSH public key. At the bottom of the form, there are two buttons: "ajouter" and "reset".

Official MesoNET documentation screenshot.

### MesoNET SSH-key guide

1. User menu → SSH keys.
2. Add the public key.
3. Project list → associate key with the machine account.
4. Wait for deployment.

## CONNECT WITH AN ALIAS

**Before connecting:** Alexandre must have added you to the IASD MesoNET project; your public key must be associated with your Juliet account.

In your laptop's ~/.ssh/config:

```
Host juliet
  HostName juliet.mesonet.fr
  User YOUR_MESONET_ID
  IdentityFile ~/.ssh/id_ed25519_mesonet
  IdentitiesOnly yes
```

```
ssh juliet
hostname
pwd
exit
```

**Permission denied?** Check username, selected key, project membership and key deployment.

Do not disable host-key checks. Verify a new host fingerprint through an official channel.

## EDIT REMOTELY WHEN NECESSARY; KEEP YOUR MAIN EDITOR LOCALLY

### nano

```
nano job.sbatch  
  
Ctrl+O, Enter: save  
Ctrl+X: exit
```

### vim

```
vim job.sbatch  
  
i: insert  
Esc, :wq: save/exit  
Esc, :q!: discard
```

### VS Code Remote SSH

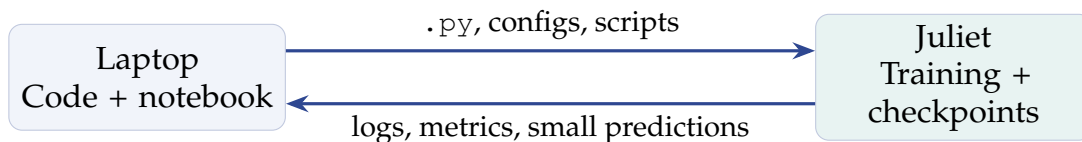
Open a remote folder and edit there.

Check the host in the status bar.  
Terminal commands run remotely.

**My recommended workflow:** local editor + selective rsync. Running in a remote editor still does not allocate a GPU: submit through Slurm.

VS Code Remote SSH documentation; Juliet also documents VS Code server, a distinct option.

## RSYNC: SEND CODE, BRING BACK EVIDENCE



```
bash scripts/push.sh juliet --dry-run
bash scripts/push.sh juliet
bash scripts/pull.sh juliet --dry-run
bash scripts/pull.sh juliet
```

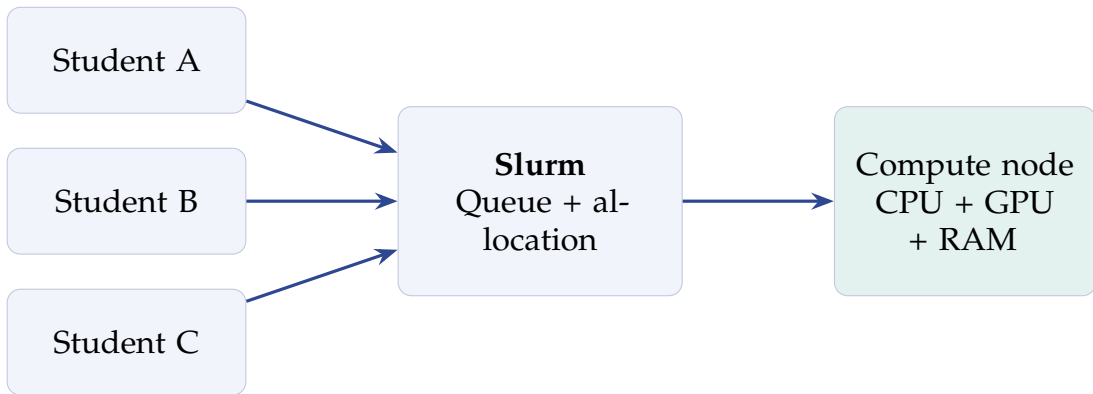
.pt / .pth / .ckpt stay remote. No -delete. Dry-run first. Keep valuable checkpoints on durable storage separately.

HPC: ASK FOR RESOURCES, THEN RUN

**HPC: ask for resources, then run**

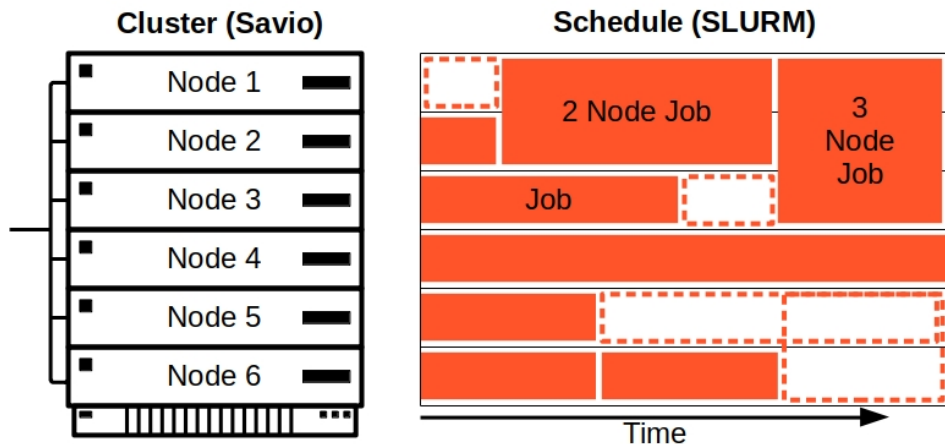
---

## WHY DO WE NEED A SCHEDULER?



Shared resources, fair scheduling, bounded jobs. Request what you need, release it when done.

## BACKFILL: USE GAPS WITHOUT DELAYING HIGHER-PRIORITY JOBS



Horizontal axis: time.

Vertical axis: nodes.

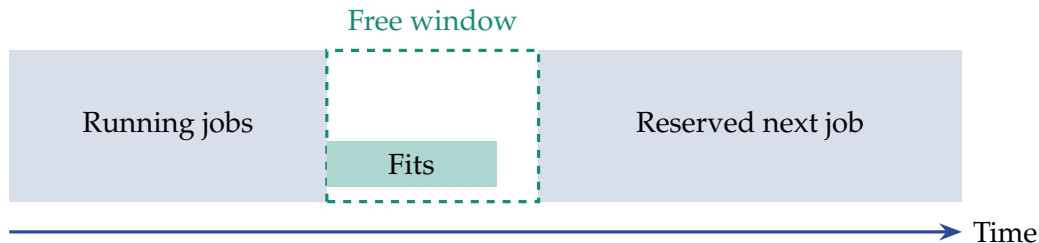
Empty rectangles are opportunities for jobs that fit.

A smaller job can start if it does not delay a higher-priority job.

Savio cluster scheduling illustration, reused from the former Slurm deck; original asset author not identified.

**Your request determines where the job fits:** realistic CPU/GPU count, RAM and wall time.

## REQUEST ENOUGH TO FINISH, NOT EVERYTHING AVAILABLE



### Overestimate

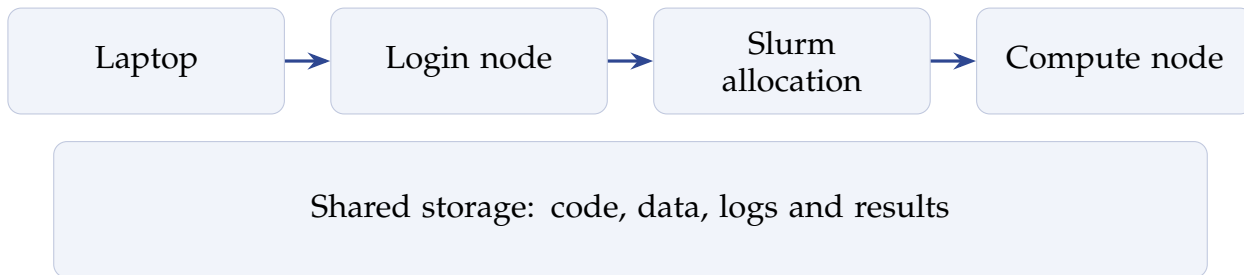
An unnecessarily long time limit or large allocation may prevent the job from fitting in a free window.

### Underestimate

The time limit can kill the job; insufficient memory can cause failure. Measure a pilot, keep margin, save checkpoints.

Conceptual diagram. Actual start time depends on cluster policies, priority and available resources; this is not a guarantee.

## THE LOGIN NODE IS NOT A TRAINING GPU



**Login:** edit, prepare files, submit, monitor.

**Compute:** train inside an allocation.

On this account: home is `/home/alverine`; project directory `/projects/m25146` exists. Check your actual paths; the documentation's generic spelling differs.

A JOB = RESOURCES + A COMMAND

Account  
Who pays?

Partition  
Which queue?

CPU + GPU  
How many?

RAM + time  
How much?

Our example: 1 GPU, 4 CPU cores, 8 GB host RAM, 15 minutes maximum.

Host RAM is not GPU memory. Reserving a GPU does not move your model onto it.

## THE BATCH SCRIPT: SMALL, EXPLICIT, READABLE

```
#!/bin/bash
#SBATCH --partition=mesonet
#SBATCH --ntasks=1
#SBATCH --cpus-per-task=4
#SBATCH --gres=gpu:1
#SBATCH --mem=8G
#SBATCH --time=00:15:00
#SBATCH --output=logs/%x-%j.out
#SBATCH --error=logs/%x-%j.err
set -euo pipefail
source .venv/bin/activate
srun python -u train.py ...
```

Shortened teaching example; the stdout/stderr paths and monitoring commands are explained in the following slides.

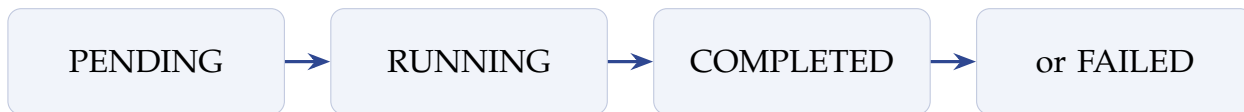
#SBATCH: allocation request.

Shell commands: executed when the job starts.

-u: flush Python output for live logs.

Create logs/ **before** submission.

## SUBMIT, WAIT, RUN, FINISH



```
export DATALAB_ACCOUNT=YOUR_PROJECT_ACCOUNT
bash scripts/submit.sh --model small_cnn \
  --run-dir runs/cnn-001 --epochs 6 --tensorboard
squeue -u "$USER"
```

**Keep the job ID.** Pending can mean resources are occupied or other jobs have priority. An invalid account or partition is a different problem.

## SINFO: WHAT RESOURCES AND PARTITIONS EXIST?

```
sinfo  
sinfo -p mesonet  
sinfo -p mesonet -o "%P %a %l %D %t %G"
```

**Partition:** a queue / set of nodes.

**Time limit:** maximum allowed duration.

**GRES:** generic resources, e.g. GPUs.

**idle:** available node.

**mix:** some resources allocated.

**alloc:** allocated. **down/drain:** unavailable for new jobs.

**sinfo describes the cluster.** It does not tell you that your job will start immediately.

## THE REAL PARTITION AND ITS GPU RESOURCES

DataLab · local terminal → Juliet    Live shell · real SSH commands

```
(.venv) juliet $ sinfo -p mesonet -o "%P %a %l %D %t %G"; squeue -u "$USER"
PARTITION AVAIL TIMELIMIT NODES STATE GRES
mesonet* up 10-00:00:00 1 inval gpu:a100:7(S:0-1)
mesonet* up 10-00:00:00 1 mix gpu:a100:8(S:0-1)
mesonet* up 10-00:00:00 1 idle gpu:a100:8(S:0-1)
      JOBID PARTITION      NAME      USER ST      TIME  NODES NODELIST(REASON)
(.venv) juliet $ █
```

Actual sinfo and squeue on Juliet, 7 October 2026. A node can be partly allocated; read the state and resources together.

## SQUEUE: WHERE IS MY JOB?

```
squeue -u "$USER"  
squeue -j JOBID  
squeue -u "$USER" \  
-o "%.12i %.9P %.18j %.10T %.10M %R"
```

### PENDING / PD

Read the reason: Resources, Priority, BeginTime, etc.  
The job has not started training yet.

### RUNNING / R

Resources have been allocated. Inspect your Slurm logs and the application's metrics.

## SCANCEL STOPS A JOB; SACCT TELLS YOU HOW IT ENDED

```
scancel JOBID  
sacct -j JOBID --format=JobID,State,ExitCode,Elapsed
```

### Cancel deliberately

Check the job ID first. Cancel your own unused pending jobs or a wrong experiment. Closing your SSH terminal does not cancel a batch job.

Slurm command roles: `sinfo` = resources; `squeue` = current jobs; `scancel` = cancel; `sacct` = job accounting.

### Read the final state

COMPLETED + ExitCode 0: successful process exit.  
FAILED, TIMEOUT, OUT\_OF\_MEMORY,  
CANCELLED: inspect logs. Scientific validity still needs checking.

## SUBMITTED DOES NOT MEAN RUNNING

```
DataLab · local terminal → Juliet  Live shell · real SSH commands

(.venv) juliet $ cancel_job=$(sbatch --parsable --begin=now+3minutes .capture-cancel.sbatch); echo "Submitted batch job
$cancel_job"; squeue -j "$cancel_job"
Submitted batch job 123016
      JOBID PARTITION   NAME     USER ST       TIME  NODES NODELIST(REASON)
      123016   mesonet datalab- alverine PD    0:00      1 (BeginTime)
(.venv) juliet $ █
```

Actual submission and squeue. For this capture we requested a future start with `-begin`: the reason is `BeginTime`, not congestion. This tiny job is cancelled next.

## CANCEL THE UNUSED JOB AND CHECK ITS STATE

DataLab · local terminal → Juliet    Live shell · real SSH commands

```
(.venv) juliet $ scancel "$cancel_job"; sacct -j "$cancel_job" --format=JobID,State,ExitCode,Elapsed; squeue -j "$cancel_job"
```

| JobID  | State      | ExitCode | Elapsed  |
|--------|------------|----------|----------|
| 123016 | CANCELLED+ | 0:0      | 00:00:00 |

| JOBID | PARTITION | NAME | USER | ST | TIME | NODES | ODELIST(REASON) |
|-------|-----------|------|------|----|------|-------|-----------------|
|-------|-----------|------|------|----|------|-------|-----------------|

```
(.venv) juliet $ █
```

Actual scancel and sacct for our demonstration job. Only our own deliberately delayed job is cancelled.

## THE ACTUAL TRAINING JOB RECEIVES A GPU

DataLab · local terminal → Juliet    Live shell · real SSH commands

```
(.env) juliet $ job=$(sbatch --parsable --account=m25146 scripts/train.sbatch --config configs/resnet18.json --run-dir runs/capture-walkthrough-visual --tensorboard); echo "Submitted batch job $job"; echo "$job" > .capture-visual-job-id
Submitted batch job 123017
(.env) juliet $ █
```

DataLab · local terminal → Juliet    Live shell · real SSH commands

```
(.env) juliet $ squeue -j "$job" -o "%.12i %.9P %.18j %.10T %.10M %R"
      JOBID PARTITION          NAME      STATE      TIME NODELIST(REASON)
      123017 mesonet      datalab-cifar  RUNNING      0:05 juliet2
(.env) juliet $ █
```

Actual ResNet submission and squeue: job 123017 running on Juliet. Same one-GPU batch script; the job continues independently of this SSH terminal.

## FOLLOW THE JOB; INVESTIGATE ITS END

```
squeue -u "$USER"  
tail -f logs/datalab-cifar-JOBID.err  
# Ctrl+C stops tail, not the job.  
sacct -j JOBID \  
    --format=JobID,State,ExitCode,Elapsed  
scancel JOBID
```

No longer in queue



Check sacct + stdout/stderr

**Disappeared**  $\neq$  **succeeded**. Cancel only your own unneeded jobs; check the recorded exit code.

## WHERE DOES EACH OUTPUT GO?

```
#SBATCH --output=logs/%x-%j.out
#SBATCH --error=logs/%x-%j.err
# Shell commands in the batch script:
hostname
nvidia-smi
srun python -u train.py ...
```

```
tail -f logs/datalab-cifar-JOBID.err
tail -f runs/resnet-001/train.log
```

**.out:** stdout (e.g. hostname, GPU details).

**.err:** stderr, including our logger's INFO messages. A nonempty file does not necessarily mean failure.

**train.log:** application log inside the run folder.

**%x / %j:** job name / job ID.

## REAL SLURM LOGS: HARDWARE, PROGRESS AND EXIT STATUS

DataLab · local terminal → Juliet    Live shell · real SSH commands

```
(.venv) juliet $ head -4 logs/datalab-cifar-121734.out; grep INFO logs/datalab-cifar-121734.err | tail -6; sacct -j 121734 --format=JobID,State,ExitCode,Elapsed
juliet2
name, memory.total [MiB]
NVIDIA A100-SXM4-80GB, 81920 MiB
2026-10-07 11:13:05,566 INFO epoch=7 loss=0.1487 val_accuracy=0.809 time=43.4s
2026-10-07 11:13:11,629 INFO epoch=8 loss=0.1108 val_accuracy=0.807 time=49.5s
2026-10-07 11:13:17,665 INFO epoch=9 loss=0.0875 val_accuracy=0.820 time=55.5s
2026-10-07 11:13:23,765 INFO epoch=10 loss=0.0715 val_accuracy=0.820 time=61.6s
2026-10-07 11:13:29,885 INFO epoch=11 loss=0.0606 val_accuracy=0.827 time=67.8s
2026-10-07 11:13:35,984 INFO epoch=12 loss=0.0560 val_accuracy=0.809 time=73.9s
JobID          State      ExitCode    Elapsed
-----
121734         COMPLETED 0:0         00:01:21
121734.batch   COMPLETED 0:0         00:01:21
121734.exte+   COMPLETED 0:0         00:01:21
121734.0       COMPLETED 0:0         00:01:21
(.venv) juliet $
```

Actual stdout, stderr and sacct from our completed ResNet run. INFO progress goes to stderr because logging.StreamHandler() uses it by default.

## DIAGNOSE BEFORE REQUESTING MORE RESOURCES

### Import error

Wrong interpreter?  
Environment activated?  
Dependency declared?

### CUDA out of memory

Lower batch size.  
Inspect GPU memory.  
Host `-mem` will not increase GPU  
memory.

### Time limit

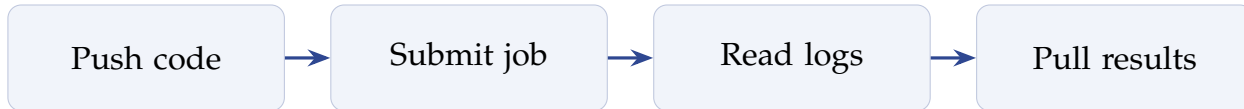
Inspect progress.  
Save checkpoints.  
Resume; request a realistic duration.

LIVE: CLUSTER TO NOTEBOOK

**Live: cluster to notebook**

---

## WATCH ONE EXPERIMENT TRAVEL TO JULIET



We will follow one job ID from submission to metrics.

If the queue is busy, we inspect a prepared run from the same code and cancel an unused pending job.

## A RUN FOLDER CONTAINS THE EVIDENCE

DataLab · local terminal → Juliet    Live shell · real SSH commands

```
(.venv) juliet $ ls -d runs/cnn-full runs/resnet-full; ls -l runs/resnet-full; cat runs/resnet-full/summary.json
runs/cnn-full  runs/resnet-full
best.pt
config.json
environment.txt
last.pt
metrics.jsonl
predictions.npz
status.json
summary.json
tensorboard
train.log
{
  "best_val_accuracy": 0.8268,
  "epochs": 12,
  "elapsed_seconds": 75.29652385041118,
  "model": "resnet18",
  "split": "validation"
}
(.venv) juliet $ █
```

Actual Juliet run folders: logs, metrics, checkpoints, predictions and summary. The summary reports validation performance and measured time.

## PULL RESULTS WITHOUT DOWNLOADING THE CHECKPOINTS

DataLab · local terminal → Juliet    Live shell · real SSH commands

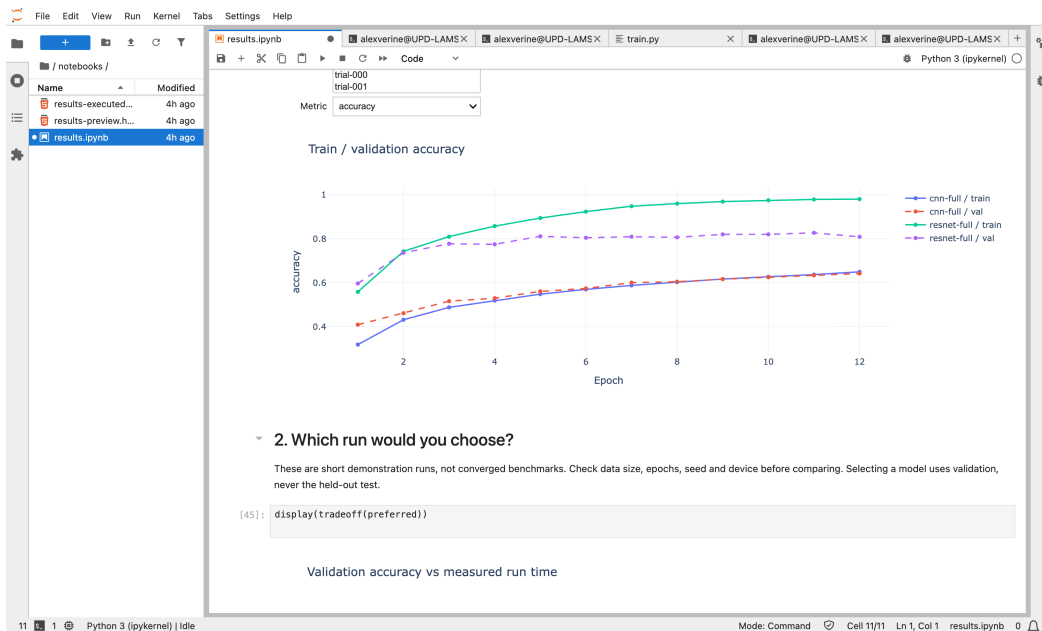
```
local $ bash scripts/pull.sh juliet --dry-run | tail -18
cd+++++++ resume-test-121741/resumed/source/
>f+++++++ resume-test-121741/resumed/source/requirements.txt
>f+++++++ resume-test-121741/resumed/source/sha256.json
cd+++++++ resume-test-121741/resumed/source/src/

sent 520 bytes  received 4249 bytes  3179.33 bytes/sec
total size is 59915852  speedup is 12563.61
receiving file list ... done
.d..t.... ./
>f+++++++ datalab-cifar-123010.err
>f+++++++ datalab-cifar-123010.out
>f+++++++ datalab-cifar-123017.err
>f+++++++ datalab-cifar-123017.out
>f+++++++ datalab-resume-test-121741.err
>f+++++++ datalab-resume-test-121741.out

sent 58 bytes  received 550 bytes  243.20 bytes/sec
total size is 13729  speedup is 22.58
local $ █
```

Actual rsync dry-run: preview the transfer of metrics, logs and small predictions. Large model checkpoints stay on the cluster.

# A NOTEBOOK FOR ANALYSIS, NOT HIDDEN TRAINING STATE



Actual JupyterLab notebook, executed locally with real Juliet results. Change the metric and selected runs; zoom both axes. Checkpoints stay on Juliet.

## LOOK AT THE MISTAKES, THEN PROPOSE THE NEXT EXPERIMENT



Real validation predictions from the teaching run. Short runs illustrate a workflow, not state-of-the-art performance.

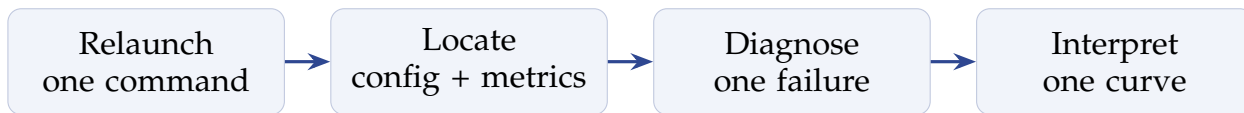
## DOWNLOAD THE COMPLETE TEACHING EXAMPLE

### Download the mini-project (ZIP)

- ▶ Python training code, CNN / ResNet models, configurations and Slurm / rsync scripts.
- ▶ Interactive analysis notebook, real metrics, logs and validation predictions.
- ▶ The two trained `best.pt` models; no raw CIFAR-10 archive or local environment.

Start with `README.md`. Open the analysis notebook without retraining; download CIFAR-10 separately if you want to train. The larger `last.pt` resume checkpoints are not included.

## CAN YOU DO THIS ON YOUR OWN?



MesoNET docs · Slurm Quick Start

**In your own project:** change one setting, use a new run directory, compare validation curves, and explain the result. Keep the test set for the final evaluation.

REFERENCE AND OPTIONAL TOOLS

## Reference and optional tools

---

## INSPECT TENSORBOARD LOCALLY AFTER PULLING THE LOGS

```
bash scripts/pull.sh juliet
python -m pip install tensorboard
tensorboard --logdir runs --host 127.0.0.1 --port 6006
```

Open `http://127.0.0.1:6006` on your laptop. This avoids running a web service on the login node and does not require a tunnel.

Remote visualization is possible with an appropriate allocation and SSH tunnel; follow the cluster's policy and topology.

## RESUME A TRUSTED CHECKPOINT

```
bash scripts/submit.sh --model small_cnn \  
  --run-dir runs/cnn-001 --epochs 12 \  
  --resume --tensorboard
```

`--epochs` is the total target, including completed epochs. Preserve architecture, data split, batch size, learning rate and seed.

Only load your own trusted checkpoints. Changing the experiment means starting a new run.

## OPTUNA: A SMALL, EXPLICIT BUDGET

```
python -m pip install -r requirements-tracking.txt  
# Run this inside an appropriate Slurm allocation.  
python tune.py --trials 3 --epochs 2
```

Each trial calls the same `train.py`; the objective is validation accuracy. The SQLite study can be reopened. Three short trials demonstrate the workflow, not reliable hyperparameter conclusions.

Pruning requires intermediate measurements and explicit stop/report logic; it is not implemented in this minimal example.

## TWO EVERYDAY COMMANDS; PROJECT-SPECIFIC FILTERS

```
bash scripts/push.sh juliet --dry-run
bash scripts/pull.sh juliet --dry-run
# Other SSH aliases: lam (Lamsade), jz (Jean Zay).
# For jz, configure YOUR work directory first.
```

### Push: inputs

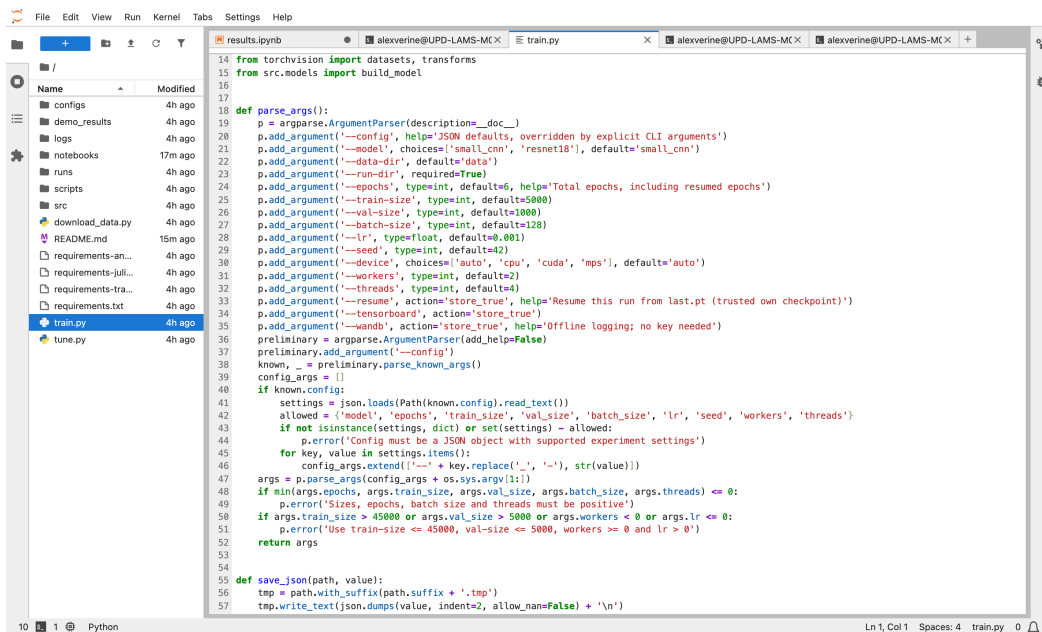
Python, scripts, configs and requirements. Keep `.json` configs and `requirements.txt`.

### Pull: evidence

Logs, metrics, TensorBoard events and small predictions. Our notebook needs `predictions.npz`.

Checkpoints stay remote. First matching filter wins. Dry-run previews transfers; same-name files may be overwritten. No automatic conflict resolution.

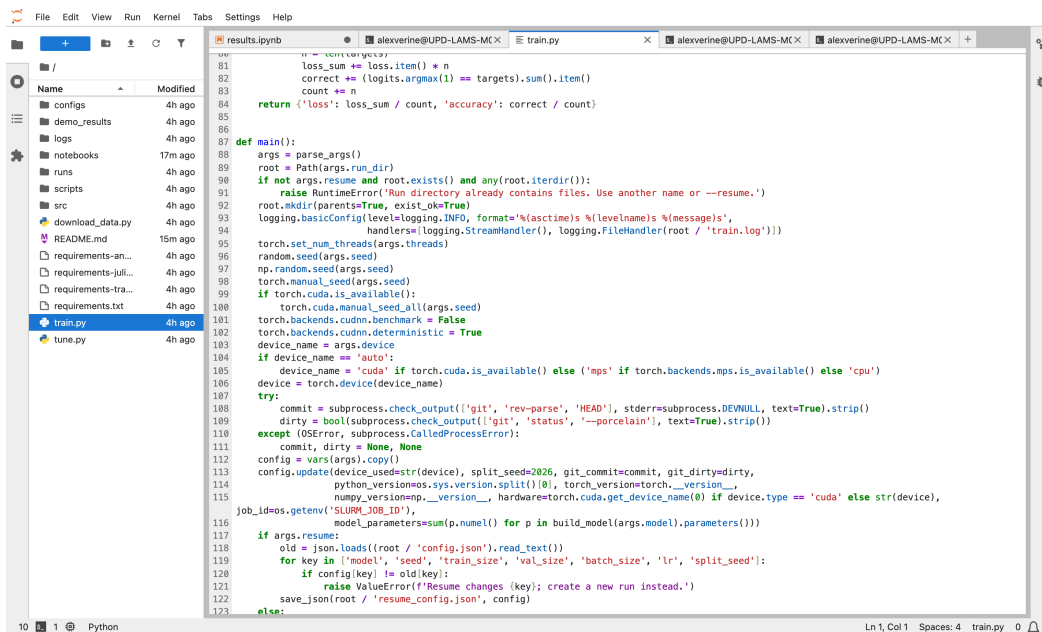
## EXAMPLE SOURCE: DECLARE THE EXPERIMENT'S ARGUMENTS



```
14 from torchvision import datasets, transforms
15 from src.models import build_model
16
17
18 def parse_args():
19     p = argparse.ArgumentParser(description=__doc__)
20     p.add_argument('--config', help='JSON defaults, overridden by explicit CLI arguments')
21     p.add_argument('--model', choices=['small_cnn', 'resnet18'], default='small_cnn')
22     p.add_argument('--data-dir', default='data')
23     p.add_argument('--run-dir', required=True)
24     p.add_argument('--epochs', type=int, default=6, help='Total epochs, including resumed epochs')
25     p.add_argument('--train-size', type=int, default=5000)
26     p.add_argument('--val-size', type=int, default=1000)
27     p.add_argument('--batch-size', type=int, default=128)
28     p.add_argument('--lr', type=float, default=0.001)
29     p.add_argument('--seed', type=int, default=42)
30     p.add_argument('--device', choices=['auto', 'cpu', 'cuda', 'mps'], default='auto')
31     p.add_argument('--workers', type=int, default=2)
32     p.add_argument('--threads', type=int, default=4)
33     p.add_argument('--resume', action='store_true', help='Resume this run from last.pt (trusted own checkpoint)')
34     p.add_argument('--tensorboard', action='store_true')
35     p.add_argument('--wandb', action='store_true', help='Offline logging; no key needed')
36     preliminary = argparse.ArgumentParser(add_help=False)
37     preliminary.add_argument('--config')
38     known, _ = preliminary.parse_known_args()
39     config_args = []
40     if known.config:
41         settings = json.loads(Path(known.config).read_text())
42         allowed = {'model', 'epochs', 'train_size', 'val_size', 'batch_size', 'lr', 'seed', 'workers', 'threads'}
43         if not isinstance(settings, dict) or set(settings) - allowed:
44             p.error('Config must be a JSON object with supported experiment settings')
45         for key, value in settings.items():
46             config_args.extend(['--' + key.replace('_', '-') + str(value)])
47     args = p.parse_args(config_args + os.sys.argv[1:])
48     if min(args.epochs, args.train_size, args.val_size, args.batch_size, args.threads) <= 0:
49         p.error('Sizes, epochs, batch size and threads must be positive')
50     if args.train_size > 45000 or args.val_size > 500 or args.workers <= 0 or args.lr <= 0:
51         p.error('Use train-size <= 45000, val-size <= 5000, workers >= 0 and lr > 0')
52     return args
53
54
55 def save_json(path, value):
56     tmp = path.with_suffix(path.suffix + '.tmp')
57     tmp.write_text(json.dumps(value, indent=2, allow_nan=False) + '\n')
```

Actual train.py opened in JupyterLab. The slide with the minimal argparse pattern explains how to reuse these ideas in your own project.

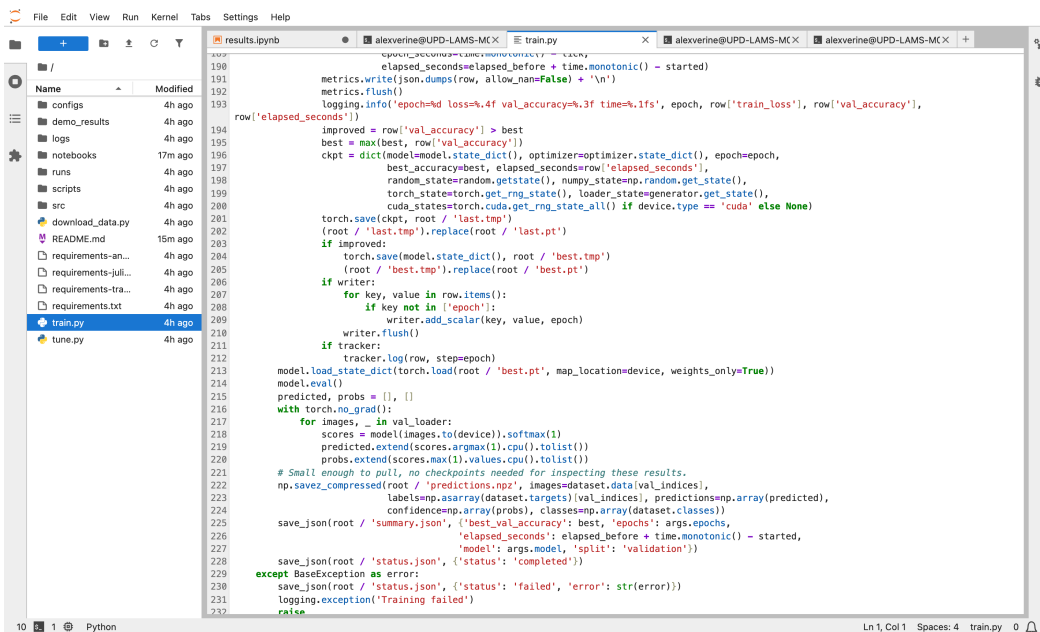
## EXAMPLE SOURCE: CREATE THE RUN AND CONFIGURE LOGGING



```
81     loss_sum += loss.item() * n
82     correct += (logits.argmax(1) == targets).sum().item()
83     count += n
84
85     return ('loss': loss_sum / count, 'accuracy': correct / count)
86
87 def main():
88     args = parse_args()
89     root = Path(args.run_dir)
90     if not args.resume and root.exists() and any(root.iterdir()):
91         raise RuntimeError('Run directory already contains files. Use another name or --resume.')
92     root.mkdir(parents=True, exist_ok=True)
93     logging.basicConfig(level=logging.INFO, format='%(asctime)s %(levelname)s %(message)s',
94                         handlers=[logging.StreamHandler(), logging.FileHandler(root / 'train.log')])
95     torch.set_num_threads(args.threads)
96     random.seed(args.seed)
97     np.random.seed(args.seed)
98     torch.manual_seed(args.seed)
99     if torch.cuda.is_available():
100         torch.cuda.manual_seed_all(args.seed)
101     torch.backends.cudnn.benchmark = False
102     torch.backends.cudnn.deterministic = True
103     device_name = args.device
104     if device_name == 'auto':
105         device_name = 'cuda' if torch.cuda.is_available() else ('mps' if torch.backends.mps.is_available() else 'cpu')
106     device = torch.device(device_name)
107     try:
108         commit = subprocess.check_output(['git', 'rev-parse', 'HEAD'], stderr=subprocess.DEVNULL, text=True).strip()
109         dirty = bool(subprocess.check_output(['git', 'status', '--porcelain'], text=True).strip())
110     except (OSError, subprocess.CalledProcessError):
111         commit, dirty = None, None
112     config = vars(args).copy()
113     config.update(device_used=str(device), split_seed=2026, git_commit=commit, git_dirty=dirty,
114                  python_version=os.sys.version.split()[0], torch_version=torch.__version__,
115                  numpy_version=np.__version__, hardware=torch.cuda.get_device_name(0) if device.type == 'cuda' else str(device),
116                  job_id=os.getenv('SLURM_JOB_ID'),
117                  model_parameters=sum(p.numel() for p in build_model(args.model).parameters()))
118     if args.resume:
119         old = json.loads((root / 'config.json').read_text())
120         for key in ['model', 'seed', 'train_size', 'val_size', 'batch_size', 'lr', 'split_seed']:
121             if config[key] != old[key]:
122                 raise ValueError(f'Resume changes {key}; create a new run instead.')
123         save_json(root / 'resume_config.json', config)
124     else:
```

Actual source: a fresh run directory, console + file logging, random seeds and device selection.

## EXAMPLE SOURCE: SAVE METRICS AND CHECKPOINTS



```
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000
```

Actual training loop: record measurements, track progress, write last / best checkpoints, and save predictions for later analysis.

## OFFICIAL REFERENCES AND CREDITS

- ▶ MesoNET account and profile; SSH keys and project association; Juliet documentation.
- ▶ Slurm Quick Start; sinfo; squeue; scancel; sacct; backfill.
- ▶ Torchvision ResNet; CIFAR-10 dataset.
- ▶ PyTorch TensorBoard; W&B tracking; Optuna; VS Code Remote SSH.
- ▶ Theme: Lars Spreng, CC BY 4.0. Earlier pedagogical material: DL3ISO project-organization slides and Constant Bourdrez's Slurm support. Asset sources recorded in `SOURCES_VISUELS.md`.