

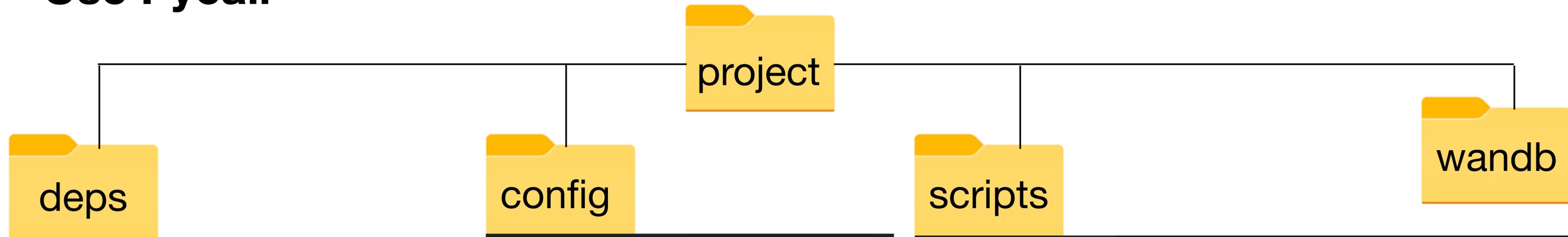
# **Optimize your model's parameters**

**Use WandB with Julia on Compute Canada**

**Léa Kaufmann - October 17<sup>th</sup> 2025**

# 1. Bind Julia and Python

## Use Pycall



```
deps > build.jl
1 using Pkg
2
3 # Set Python virtual env
4 ENV["PYTHON"] = abspath(".venv/bin/python")
5
6 # Configure PyCall to use the virtual env
7 Pkg.build(["PyCall"])
```

```
config > default_config.toml
1 data_path = "data/tahoe100M/pseudo_bulk.jld2"
2
3 result_dir = "results/predict_profile"
4 wandb_mode = "offline"
5
6
7 hidden_layers = [512, 512]
8 activation_fct = "relu"
9 batch_size = 64
10 n_epochs = 500
11 lr = 0.00001
```

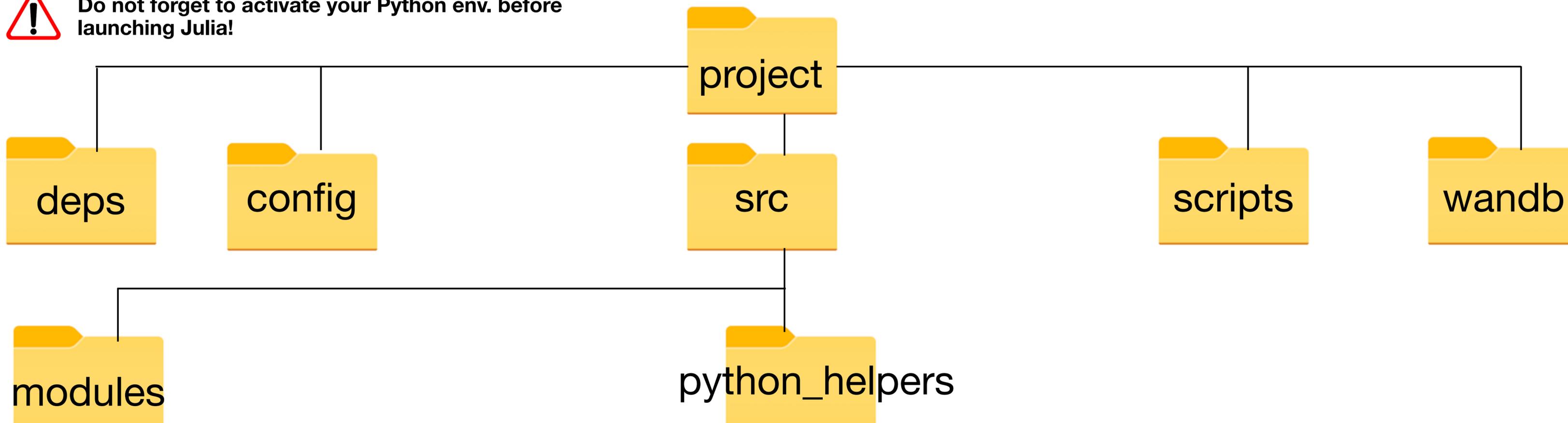
```
scripts > predict_profile copy.jl > ...
1 # Run setup script → ENV["PYTHON"] must be set before Julia loads PyCall
2 include("../deps/build.jl")
3
4 using PyCall
5 println("Using Python from: ", PyCall.python)
6
7 using Dates
8 using TOML
9
10
11 # ----- Configuration -----
12
13 timestamp = Dates.format(now(), "yyyy-mm-dd_HHMMSS")
14
15 # Load config
16 config_file = length(ARGS) > 0 ? ARGS[1] : "config/default_config.toml"
17 config = TOML.parsefile(config_file)
18 config_basename = splitext(basename(config_file))[1] # "default_config"
19
20 wandb_mode = config["wandb_mode"]
21
22 # Initialize a Wandb run
23 run_name = "$(config_basename)_$timestamp"
24 wandb = pyimport("wandb")
25 wandb.init(project="CAP_Tahoe100M", config=config, mode=wandb_mode, name=run_name)
26
```

```
>_ $ python3 -m venv .venv
$ source .venv/bin/activate
$ pip install wandb
$ julia
$ ]
$ activate .
$ add PyCall
```

# 1. Bind Julia and Python

It can be useful to use PyCall for other Python packages than WandB, or for your own Python functions

 Do not forget to activate your Python env. before launching Julia!



```
src > modules > Processing.jl > ...
1  module Processing
2
3  using PyCall
4
5
6  function call_scaffold_split_py(all_smiles, test_fraction)
7      unique_smiles = unique(all_smiles)
8      deepchem = pyimport("use_deepchem")
9      train_smiles, test_smiles = deepchem.scaffold_split(unique_smiles, test_fraction)
10     test_indices = findall(smile → smile in test_smiles, all_smiles)
11     return test_indices
12 end
13
```

```
src > python_helpers > use_deepchem.py > ...
1  import deepchem as dc
2  import numpy as np
3  from typing import List, Tuple
4
5
6  def scaffold_split(unique_smiles: List[str], test_fraction: float) → Tuple[List[str], List[str]]:
7      train_fraction = 1 - test_fraction
8      # Creation of a deepchem dataset
9      Xs = np.zeros(len(unique_smiles))
10     Ys = np.ones(len(unique_smiles))
11     dataset = dc.data.DiskDataset.from_numpy(X = Xs, y = Ys, w = np.zeros(len(unique_smiles)), ids = unique_smiles)
12     scaffoldsplitter = dc.splits.ScaffoldSplitter()
13     train, test = scaffoldsplitter.train_test_split(dataset, frac_train = train_fraction)
14     return train.ids, test.ids
15
```

# 2. Use WandB

## Loop over epochs

### Train function

```
function custom_train!(X_train,  
                       Y_train,  
                       X_test,  
                       Y_test,  
                       model_gpu,  
                       batch_size,  
                       n_epochs,  
                       opt_state;  
                       wandb=wandb, ←  
                       loss_name="mse")
```

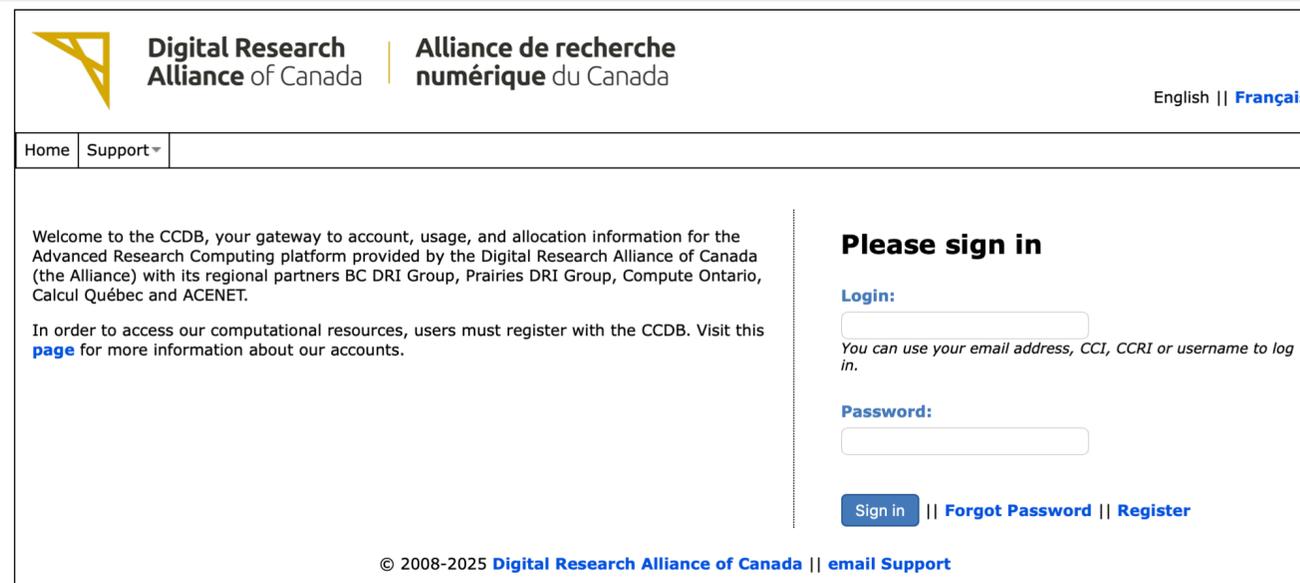
```
prog = Progress(n_epochs; desc="Training", showspeed=true)  
for epoch in 1:n_epochs  
  
    for (x, y) in train_data_loader  
        grads = Flux.gradient(model_gpu) do m  
            ŷ_gpu = m(gpu(x))  
            loss(ŷ_gpu, gpu(y))  
        end  
        Flux.update!(opt_state, model_gpu, grads[1])  
    end  
  
    # Compute losses  
    train_loss = mean(loss(model_gpu(gpu(x)), gpu(y)) ▷ cpu for (x, y) in train_data_loader)  
    test_loss = mean(loss(model_gpu(gpu(x)), gpu(y)) ▷ cpu for (x, y) in test_data_loader)  
  
    # Log at every epoch  
    log_data = Dict(  
        "epoch" => epoch,  
        "train_loss" => train_loss,  
        "test_loss" => test_loss  
    )  
  
    # Evaluation on the test set only every 10 epochs  
    if epoch % 10 == 0  
        ŷ_test, test_avg_pearson_corr, test_avg_spearman_corr, test_avg_l2_dist, test_avg_cosine_similarity =  
            evaluate_model(X_test, Y_test, batch_size, model_gpu)  
  
        merge!(log_data, Dict(  
            "test_avg_pearson_corr" => test_avg_pearson_corr,  
            "test_avg_spearman_corr" => test_avg_spearman_corr,  
            "test_avg_l2_dist" => test_avg_l2_dist,  
            "test_avg_cosine_similarity" => test_avg_cosine_similarity  
        ))  
    end  
  
    wandb.log(log_data)  
  
    next!(prog)  
end
```

It is also possible to save a plot

```
# Plot hexbin truth vs. predictions  
train_hexbin_file = run_dir * "/train_truth_vs_predictions.pdf"  
plot_truth_vs_predictions("Train", Y_train, ŷ_train, plot_info, train_hexbin_file)  
wandb.save(train_hexbin_file)
```

# 3. Compute Canada

## Create an account and log in



Digital Research Alliance of Canada | Alliance de recherche numérique du Canada

English || Français

Home | Support ▾

Welcome to the CCDB, your gateway to account, usage, and allocation information for the Advanced Research Computing platform provided by the Digital Research Alliance of Canada (the Alliance) with its regional partners BC DRI Group, Prairies DRI Group, Compute Ontario, Calcul Québec and ACENET.

In order to access our computational resources, users must register with the CCDB. Visit this [page](#) for more information about our accounts.

**Please sign in**

Login:

*You can use your email address, CCI, CCRI or username to log in.*

Password:

[Sign in](#) || [Forgot Password](#) || [Register](#)

© 2008-2025 Digital Research Alliance of Canada || [email Support](#)

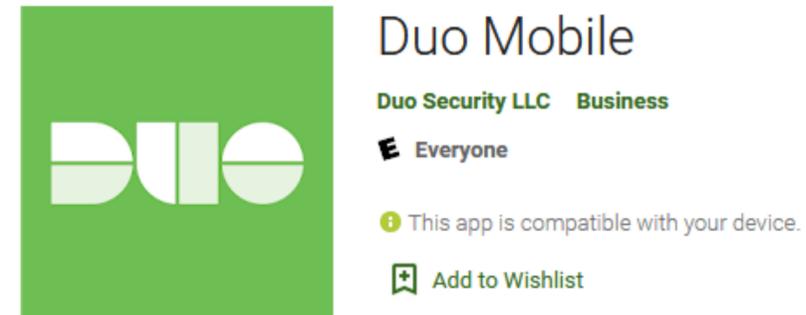
[https://docs.alliancecan.ca/wiki/Technical\\_documentation](https://docs.alliancecan.ca/wiki/Technical_documentation)

### Systems and services

#### Systems

- [2025 infrastructure renewal](#)
- **General-purpose clusters**, for CPU or GPU jobs:
  - [Fir](#), [Nibi](#), [Narval](#), [Rorqual](#)
- [Trillium](#), a cluster designed for large parallel jobs
- GPU clusters which are part of the Pan-Canadian AI Compute Environment (PAICE)
  - [Killarney](#), [tamIA](#), [Vulcan](#)
- Legacy clusters that are at the end of their life or retired:
  - [Béluga](#), [Cedar](#), [Graham](#), [Niagara](#)

## Double authentication



```
>_ $ username@narval.calculcanada.ca
# Generate a SSH key on the cluster
$ ssh-keygen -t ed25519 -C "user_email_address"
# Display and copy the public key
$ cat ~/.ssh/id_ed25519.pub
```

Add the key on GitHub: <https://github.com/settings/keys>

# 3. Compute Canada

## Setup your project

The calcul clusters have no Internet access!

Everything must be installed on the login node.

Home

```
[lkaufman@narval3 ~]$ ls  
CAP_Tahoe100M  nearline  project  projects  scratch  
[lkaufman@narval3 ~]$
```

Cloned project

```
>_ $ username@narval.calculcanada.ca
```

```
$ git clone CAP_Tahoe100M
```

```
$ module load cuda/12.9
```

```
$ module load Julia
```

Python  
package  
(optional)

```
# See RDKit available versions
```

```
$ module spider rdkit
```

```
$ module load rdkit/2024.09.6
```

```
$ cd CAP_Tahoe100M
```

```
$ python3 -m venv .venv
```

```
$ source .venv/bin/activate
```

```
$ pip install -r requirements.txt
```

```
$ pip install wandb
```

```
$ julia
```

```
$ ]
```

```
$ activate .
```

```
$ instantiate
```

```
# Return to the Julia prompt
```

```
$ include("deps/build.jl") # Requires Internet
```

Adapt with your own

Do it only the first time you  
run your project

Always do it

# 3. Compute Canada

## Run your code in interactive mode + WandB

```
>_ $ username@narval.calculcanada.ca
```

```
$ salloc --time=03:00:00 --cpus-per-task=12 --gpus=a100_2g.10gb:1 --mem=64G
```

```
# Get the list of loaded modules
```

```
$ module list
```

```
$ module load cuda/12.9
```

```
$ module load Julia
```

```
$ cd CAP_Tahoe100M
```

```
$ source .venv/bin/activate
```

```
$ export WANDB_MODE=offline
```

```
# Provide your WandB API key
```

```
$ wandb login a302ba43d4e7fbb4b98e5579f17e011831d1d3b4
```

```
$ nohup julia --project=. scripts/predict_profile.jl &
```

```
# See your jobs
```

```
$ sq # equivalent to "queue -u username"
```



**WARNING:** Remove the line `include("deps/build.jl")` from the script

```
scripts > predict_profile copy.jl > ...
1 # Run setup script → ENV["PYTHON"] must be set before Julia loads PyCall
2 include("../deps/build.jl")
3
4 using PyCall
5 println("Using Python from: ", PyCall.python)
6
7 using Dates
8 using TOML
9
```

Adapt with your own

Always do it

**Put your logs on WandB platform once your run is done (from login node)**

```
$ wandb sync wandb/
```

# 3. Compute Canada

## Launch several jobs at the same time + WandB



```
$ username@narval.calculcanada.ca
```

```
$ sbatch scripts/grid_search.slurm
```

```
$ wandb sync wandb/
```

grid\_search.slurm

```
#SBATCH --error=logs/grid_%A_%a.err
#SBATCH --time=03:00:00
#SBATCH --cpus-per-task=12
#SBATCH --gpus=a100_2g.10gb:1
#SBATCH --mem=64G
#SBATCH --array=901-1800

# If SLURM_SUBMIT_DIR is not set (manual run), fallback to current directory
if [ -z $SLURM_SUBMIT_DIR ]; then
    export SLURM_SUBMIT_DIR=$(pwd)
fi

cd $SLURM_SUBMIT_DIR

module load cuda/12.9
module load julia

# Activate Python virtual environment
source .venv/bin/activate

export WANDB_MODE=offline
wandb login a302ba43d4e7fbb4b98e5579f17e011831d1d3b4

# Use SLURM_ARRAY_TASK_ID, or default to 1 (useful for local testing)
TASK_ID=${SLURM_ARRAY_TASK_ID:-1}
CONFIG_ID=$(printf "%03d" $TASK_ID)
CONFIG_FILE=config/grid_configs/config_${CONFIG_ID}.toml

echo "Running with config: $CONFIG_FILE"

# Run the Julia script
julia --project=. scripts/predict_profile.jl "$CONFIG_FILE"
```

# 4. WandB interface

The screenshot shows the WandB interface for a project named 'Lea-kaufmann's workspace'. The breadcrumb navigation at the top indicates the path: lea\_irc > Projects > CAP\_Tahoe100M > Runs > config\_010\_2025-09-29\_1... The workspace is identified as 'Personal workspace'. The specific run is 'config\_010\_2025-09-29\_175457', which is in a 'Finished' state. The interface includes a sidebar with navigation options: Project, Workspace, Runs, Automat., Sweeps, Reports, and Artifacts. The main content area displays various metadata for the run, including author, start time, runtime, and system hardware.

Notes	What makes this run special?
Tags	<a href="#">+</a>
Author	lea-kaufmann
State	<span>Finished</span>
Start time	September 29th, 2025 5:54:59 PM
Runtime <sup>i</sup>	2d 11h 23m 46s
Tracked hours <sup>i</sup>	3s
Run path	lea_irc/CAP_Tahoe100M/ythh06ul
Hostname	ng30303
OS	Linux-4.18.0-553.63.1.el8_10.x86_64-x86_64-AMD_EPYC_7413_24-Core_Processor-with-glibc2.37
Python version	CPython 3.11.4
Python executable	/home/lkaufman/CAP_Tahoe100M/.venv/bin/python
Git repository	<a href="#">git clone</a>
Git state	<code>git checkout -b "config_010_2025-09-29_175457" a7a2a988f30b759a626e30a45526b44abff5d644</code>
Command	<code>&lt;python with no main file&gt;</code>
System Hardware	CPU count: 48 Logical CPU count: 48 GPU count: 1
W&B CLI Version	0.21.2

## Run details

## Config

The screenshot displays the 'Config' page for the run. It is divided into two main sections: 'Config parameters' and 'Summary metrics'. Both sections include a search bar for keys using regex. The 'Config parameters' section lists 12 keys, including activation\_fct, base\_processed\_data\_dir, base\_result\_dir, batch\_size, hidden\_layers, loss\_name, lr, n\_epochs, ref\_cl, split\_method, wandb\_mode, and weight\_decay. The 'Summary metrics' section lists 7 keys, including epoch, test\_avg\_cosine\_similarity, test\_avg\_l2\_dist, test\_avg\_pearson\_corr, test\_avg\_spearman\_corr, test\_loss, and train\_loss.

Config parameters: {} 12 keys	Summary metrics: {} 7 keys
activation_fct: "relu"	epoch: 500
base_processed_data_dir: "data/tahoe100M/processed_data"	test_avg_cosine_similarity: 0.24
base_result_dir: "results"	test_avg_l2_dist: 52.18
batch_size: 64	test_avg_pearson_corr: 0.24
hidden_layers: [] 4 items	test_avg_spearman_corr: 0.2
0: 1,024	test_loss: 0.508275032043457
1: 512	train_loss: 0.5305700302124023
2: 256	
3: 128	
loss_name: "rmse"	
lr: 0.001	
n_epochs: 500	
ref_cl: "CVCL_0023"	
split_method: "scaffold"	
wandb_mode: "offline"	
weight_decay: 0.00001	

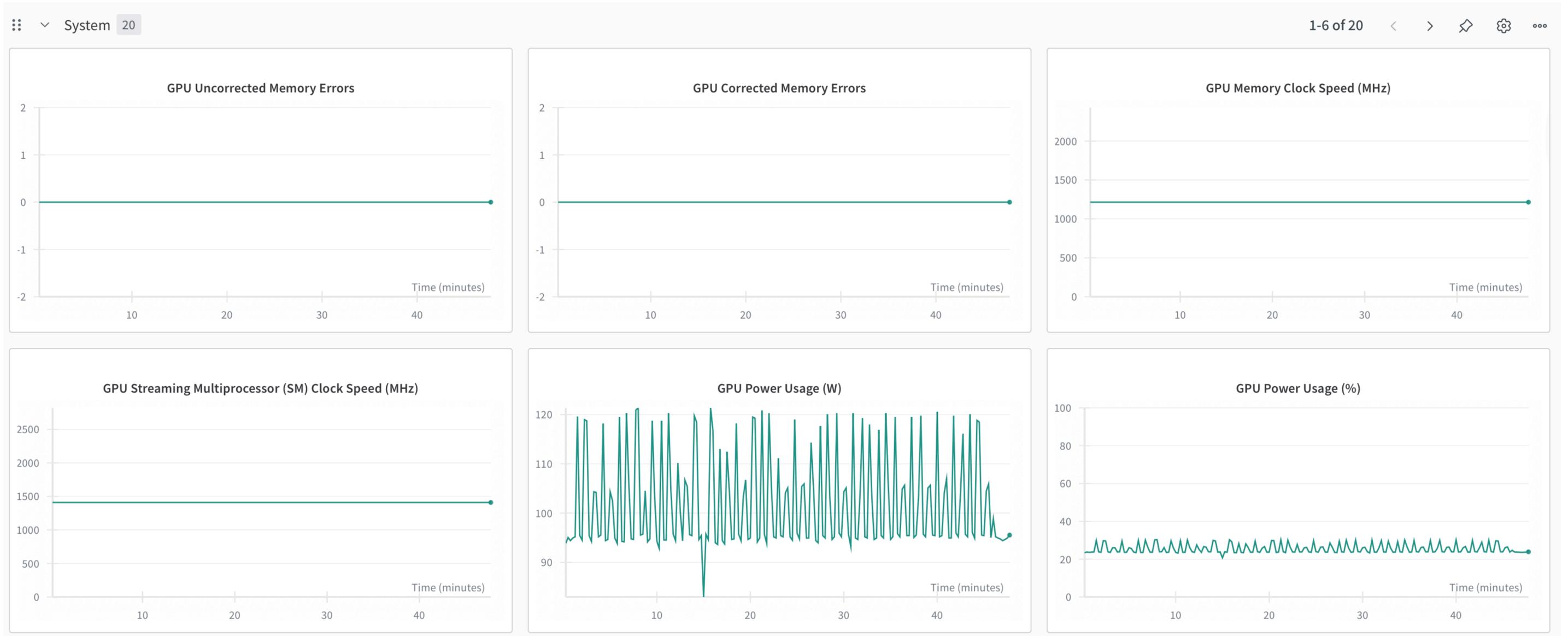
# 4. WandB interface

## Logged metrics



# 4. WandB interface

## System





# 4. WandB interface

The screenshot displays the WandB interface with the following components:

- Left Sidebar:** Navigation icons for Project, Workspace, Runs, Automat., Sweeps, Reports, and Artifacts.
- Runs List:** A list of 221 runs, with 221 visualized. The list includes configuration names and dates, such as `config_164_2025-09-29_235434`.
- Evaluation Panel:** A chart showing `batch_size` (left y-axis, 60-260) and `lr` (middle y-axis, 0.000010-0.000100) over time. A color bar on the right indicates `test_avg_pearson_corr` values from 0.570 to 0.600.
- Panel Section:** A chart showing `test_avg_pearson_corr` over time, with a color bar on the right indicating values from 0.570 to 0.600.
- Charts Section:** Three charts showing the first 10 runs for `train_loss`, `test_loss`, and `test_avg_spearman_corr`. The `train_loss` chart shows values decreasing from 0.46 to 0.36. The `test_loss` chart shows values increasing from 0.415 to 0.44. The `test_avg_spearman_corr` chart shows values fluctuating between 0.52 and 0.56.

# 4. WandB interface

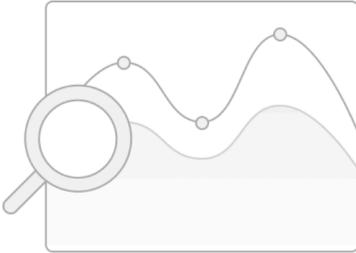
lea\_irc > Projects > CAP\_Tahoe100M > Sweeps

lea Kaufmann  
lea\_irc-org

Project  
Workspace  
Runs  
Automat.  
**Sweeps**  
Reports  
Artifacts

## Sweeps

Scalable, customizable hyperparameter search



Use Sweeps for hyperparameter optimization. Sweeps work with Bayesian search, random search, grid search, custom logic, and also support early stopping.

[Try Code](#) [Create Sweep](#)

[See docs →](#)

# 4. WandB interface

## Parameter importance with correlation

