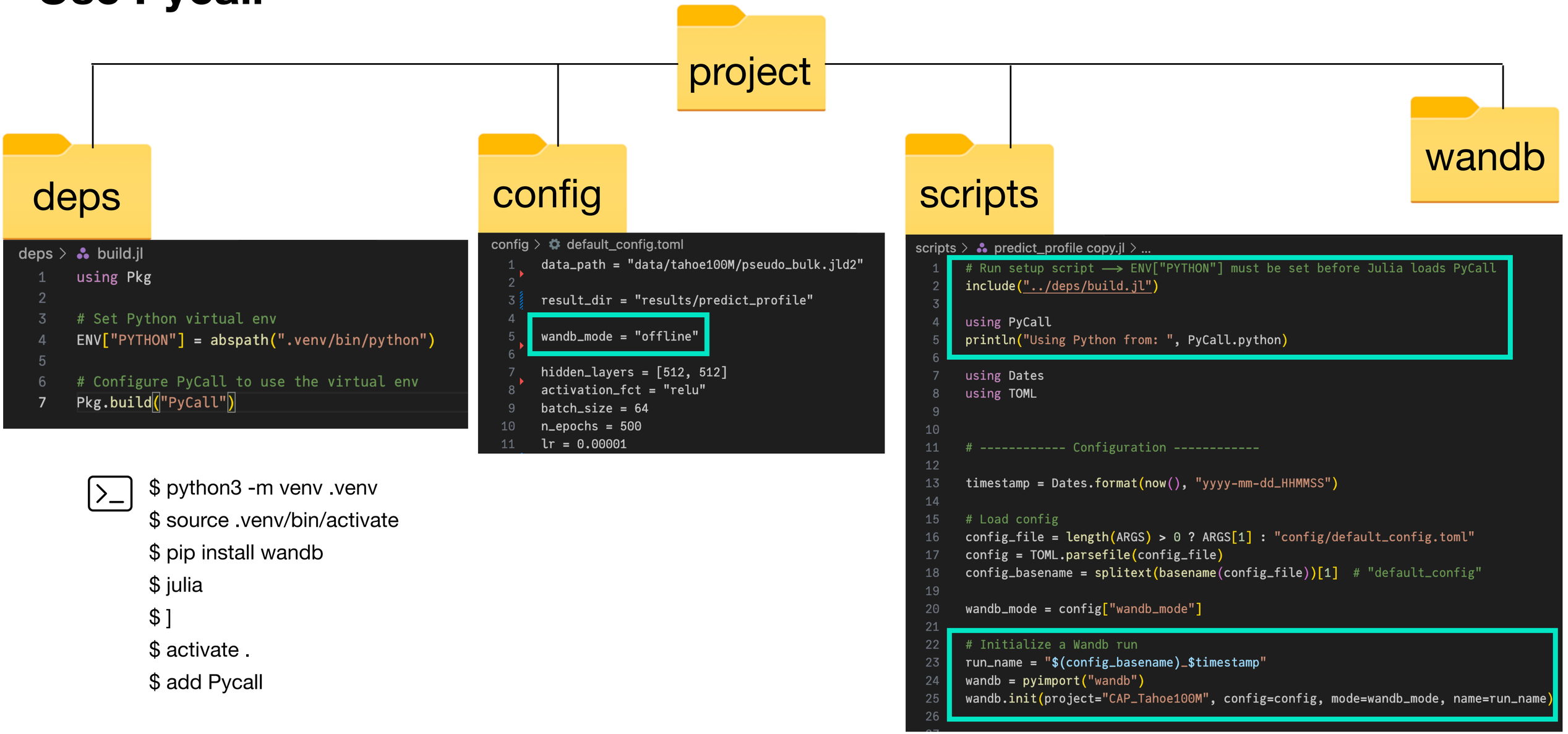
Optimize your model's parameters

Use WandB with Julia on Compute Canada

Léa Kaufmann - October 17th 2025

1. Bind Julia and Python

Use Pycall



deps > build.jl

```julia
using Pkg

# Set Python virtual env
ENV["PYTHON"] = abspath(".venv/bin/python")

# Configure PyCall to use the virtual env
Pkg.build("PyCall")
```

config > default_config.toml

```toml
data_path = "data/tahoe100M/pseudo_bulk.jld2"

result_dir = "results/predict_profile"

wandb_mode = "offline"

hidden_layers = [512, 512]
activation_fct = "relu"
batch_size = 64
n_epochs = 500
lr = 0.00001
```

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

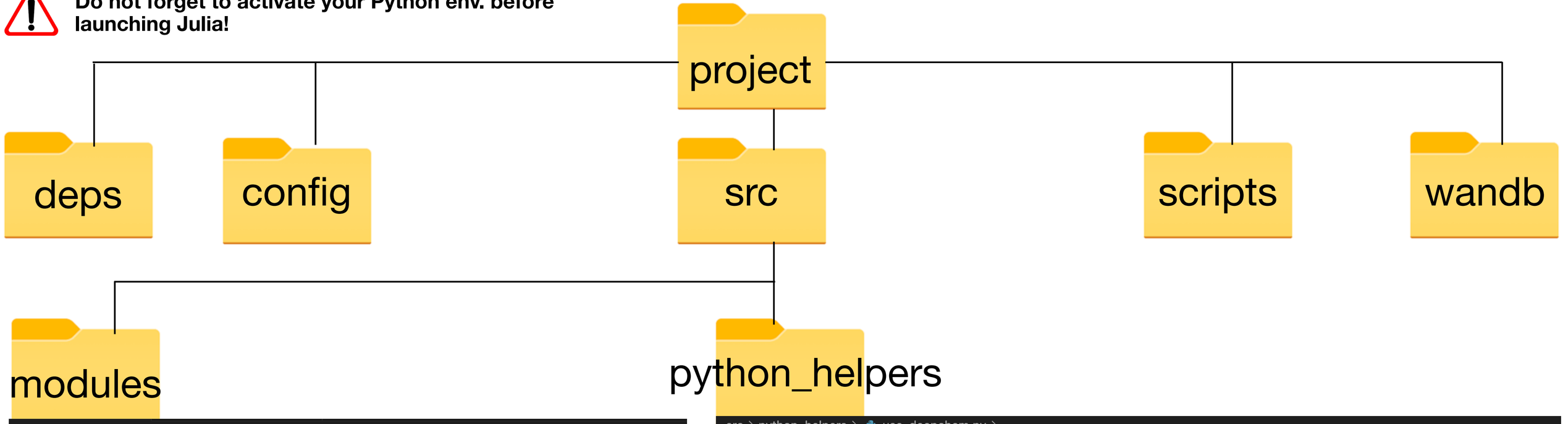
scripts > predict_profile copy.jl > ...

```julia
# Run setup script ⟶ ENV["PYTHON"] must be set before Julia loads PyCall
include("../deps/build.jl")

using PyCall
println("Using Python from: ", PyCall.python)

using Dates
using TOML


# ------------ Configuration ------------

timestamp = Dates.format(now(), "yyyy-mm-dd_HHMMSS")

# Load config
config_file = length(ARGS) > 0 ? ARGS[1] : "config/default_config.toml"
config = TOML.parsefile(config_file)
config_basename = splitext(basename(config_file))[1]  # "default_config"

wandb_mode = config["wandb_mode"]

# Initialize a Wandb run
run_name = "$(config_basename)_$timestamp"
wandb = pyimport("wandb")
wandb.init(project="CAP_Tahoe100M", config=config, mode=wandb_mode, name=run_name)
```

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 > ...

```julia
module Processing

using PyCall


function call_scaffold_split_py(all_smiles, test_fraction)
    unique_smiles = unique(all_smiles)
    deepchem = pyimport("use_deepchem")
    train_smiles, test_smiles = deepchem.scaffold_split(unique_smiles, test_fraction)
    test_indices = findall(smile -> smile in test_smiles, all_smiles)
    return test_indices
end

```

src > python_helpers > use_deepchem.py > ...

```python
import deepchem as dc
import numpy as np
from typing import List, Tuple


def scaffold_split(unique_smiles: List[str], test_fraction: float) -> Tuple[List[str], List[str]]:
    train_fraction = 1 - test_fraction
    # Creation of a deepchem dataset
    Xs = np.zeros(len(unique_smiles))
    Ys = np.ones(len(unique_smiles))
    dataset = dc.data.DiskDataset.from_numpy(X = Xs, y = Ys, w = np.zeros(len(unique_smiles)), ids = unique_smiles)
    scaffoldsplitter = dc.splits.ScaffoldSplitter()
    train, test = scaffoldsplitter.train_test_split(dataset, frac_train = train_fraction)
    return train.ids, test.ids
```

2. Use WandB

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")
```

Loop over epochs

```
prog = Progress(n_epochs; desc="Training", showspeed=true)
for epoch in 1:n_epochs

    for (x, y) in train_dataloader
    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_dataloader)
    test_loss  = mean(loss(model_gpu(gpu(x)), gpu(y)) ▷ cpu for (x, y) in test_dataloader)

    # 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
```

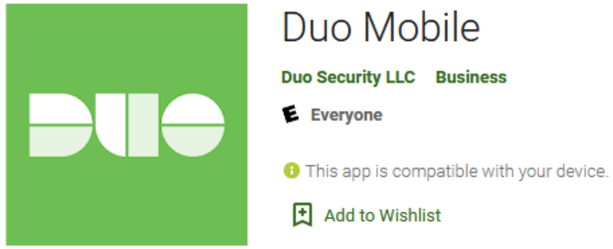
It is also possible to save a plot

```
# 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

Double authentication



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

```
$ 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 “squeue -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

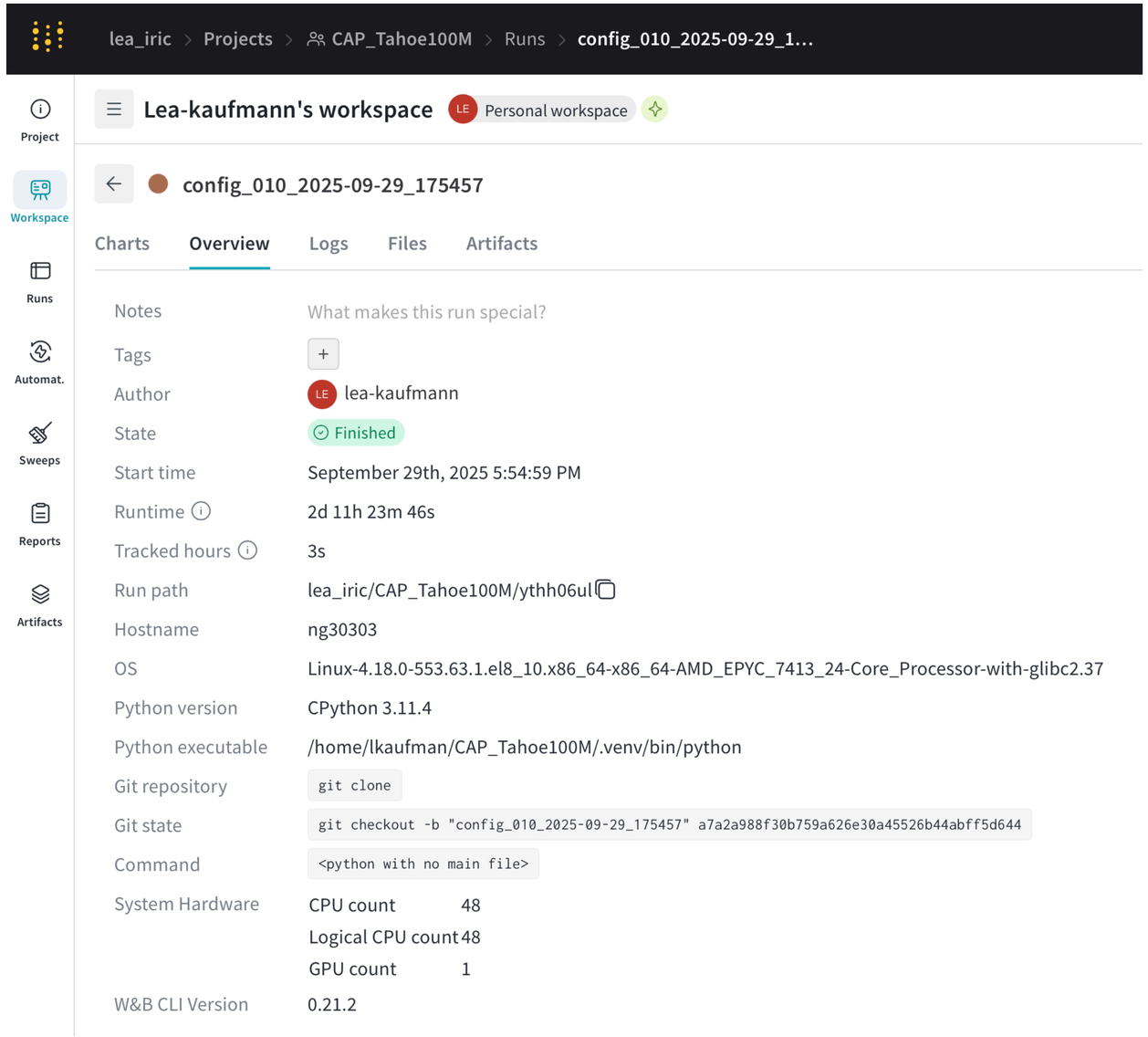
Launch several jobs at the same time + WandB

```
$  username@narval.calculcanada.ca
$ sbatch scripts/grid_search.slurm
$ wandb sync wandb/
```

grid_search.slurm

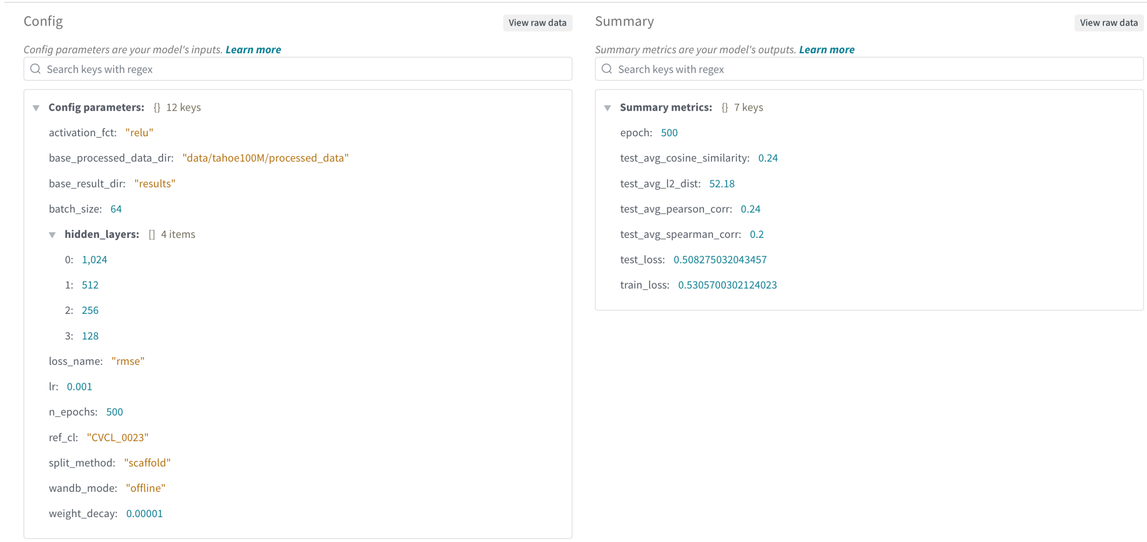
```
#SBATCH --error=logs/grid_%A_%a.err
#SBATCH --time=03:00:00
#SBATCH --cpus-per-task=12
#SBATCH --gpus=a100_2g.10gb:1
#SBATCH --mem=64G
#SBATCH --array=901-1800

# 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



Run details

Config

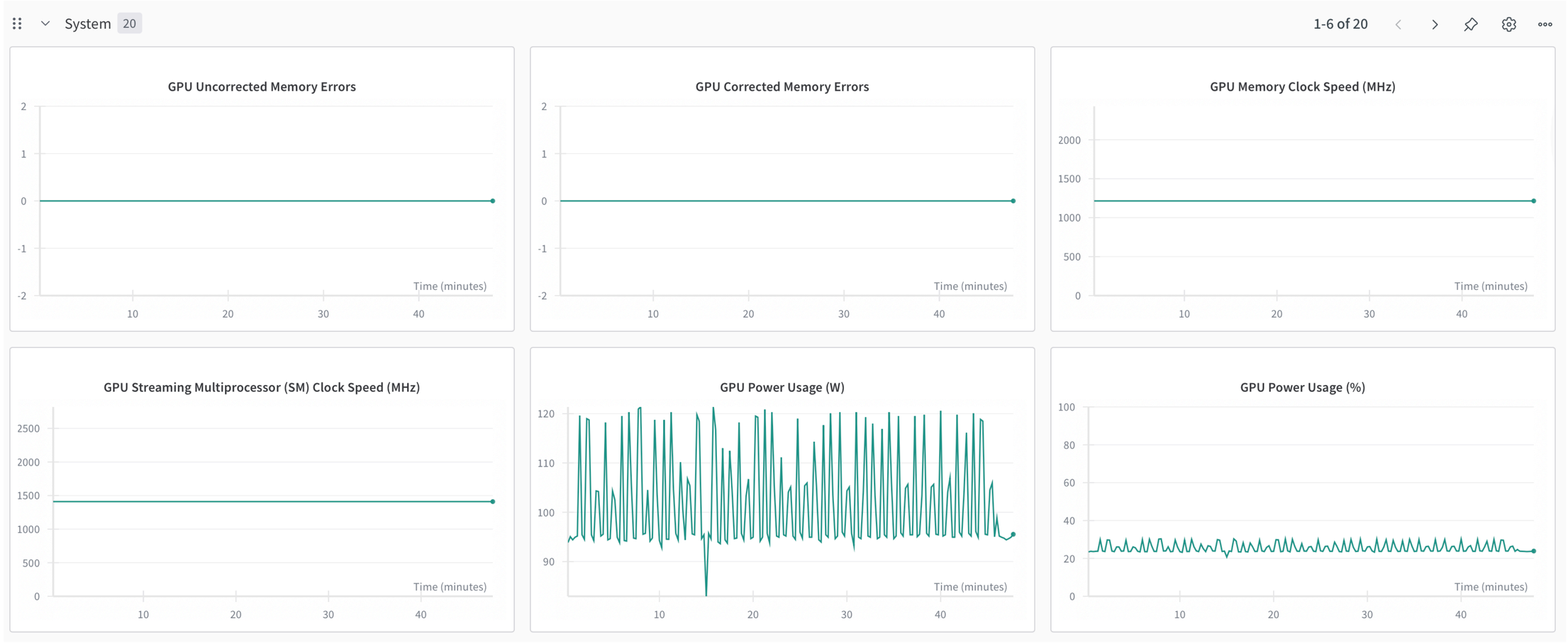


4. WandB interface

Logged metrics

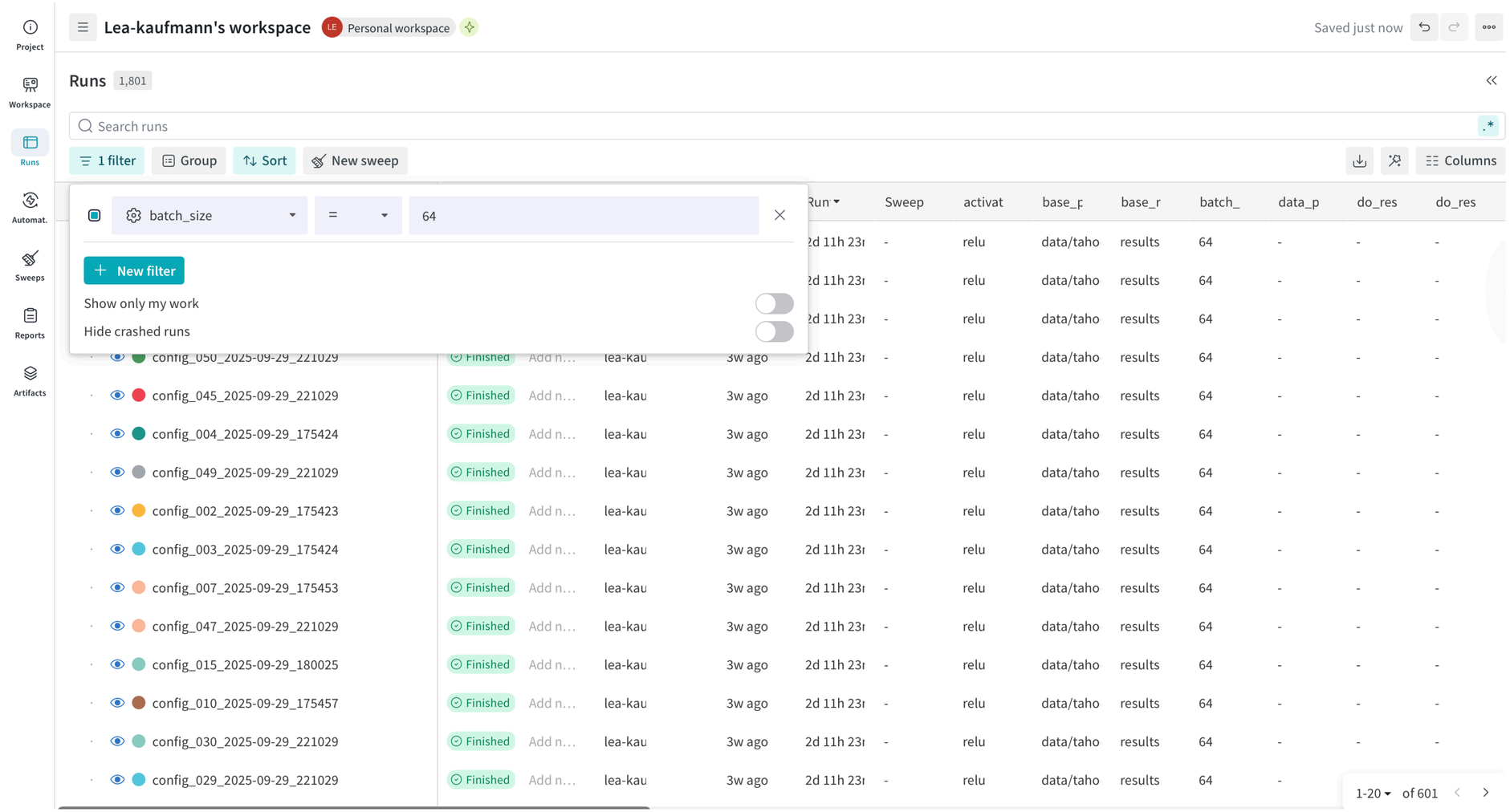
4. WandB interface

System

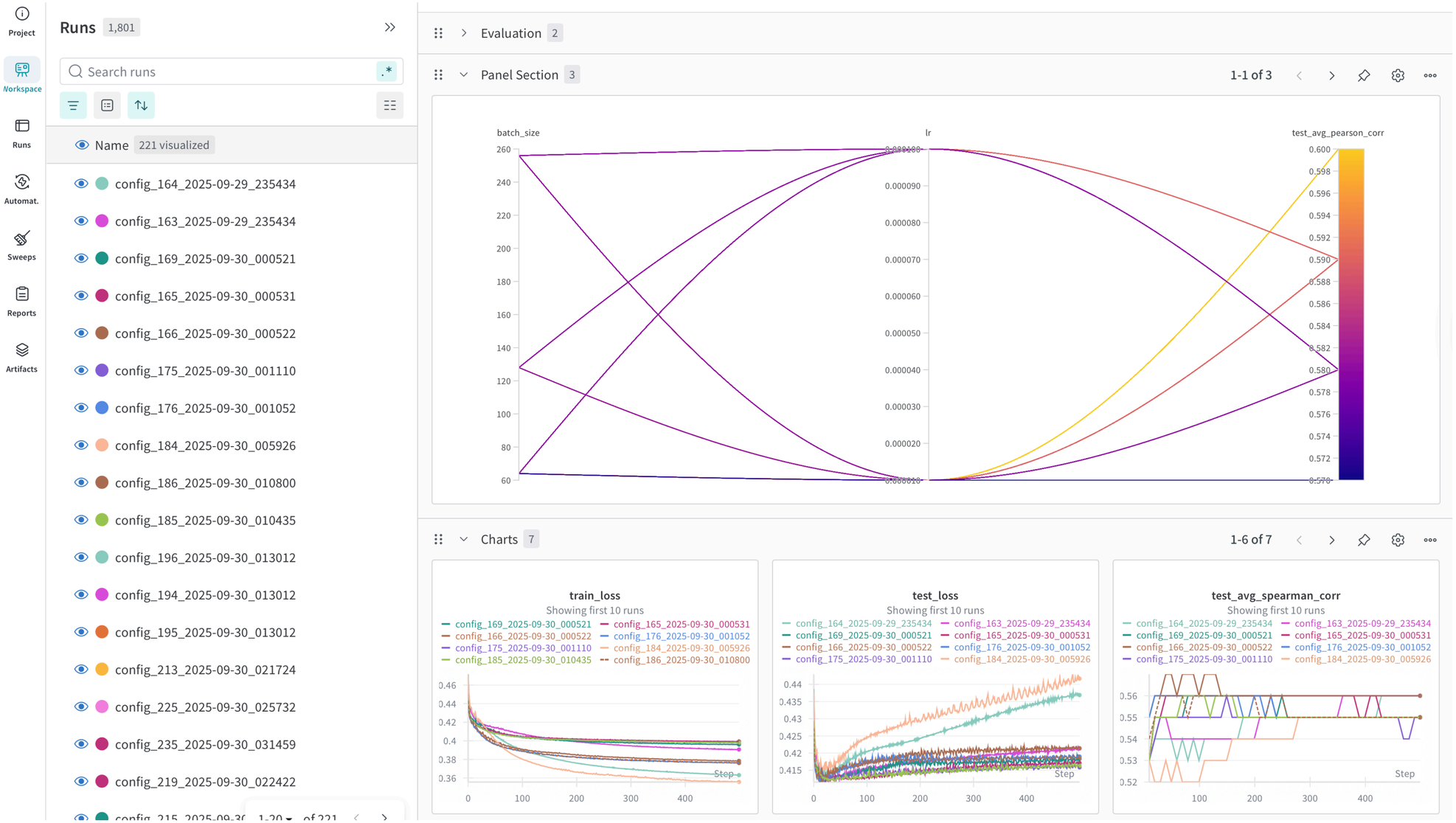


4. WandB interface

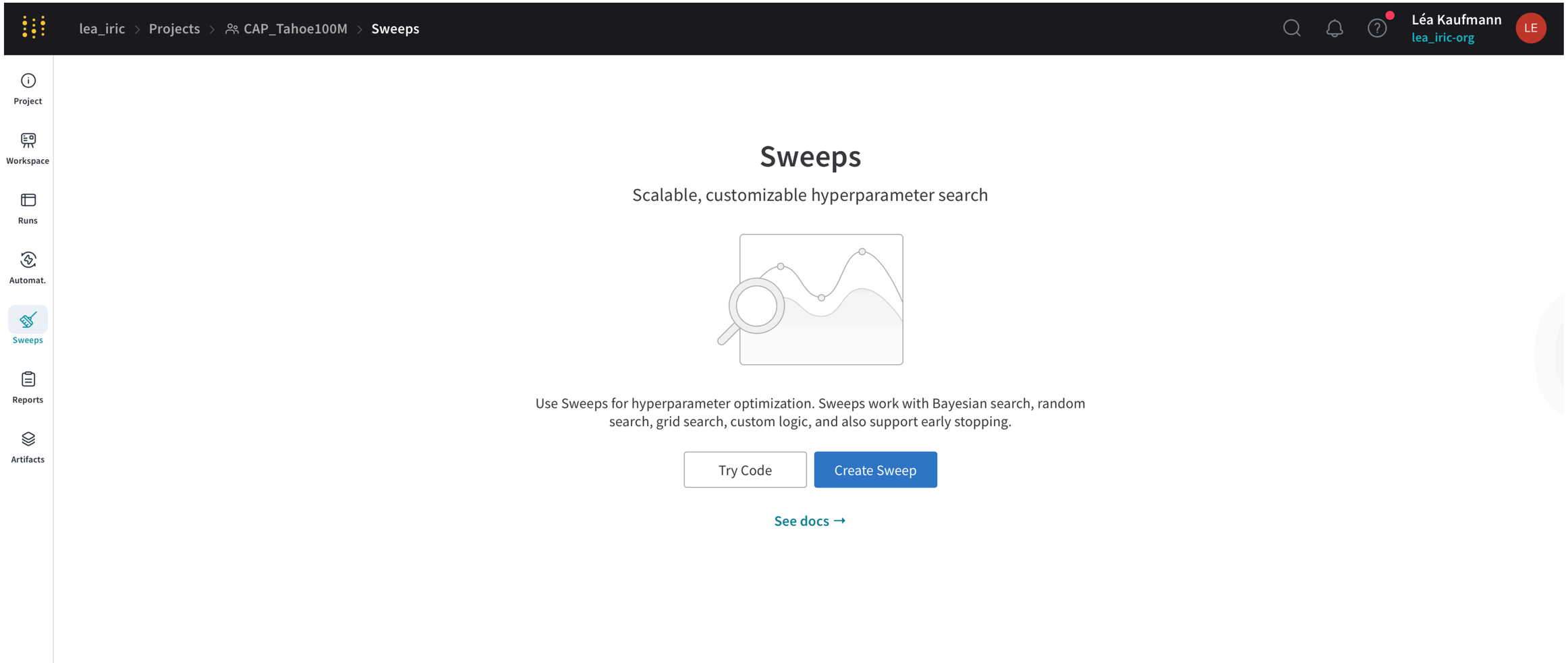
Run filtering



4. WandB interface



4. WandB interface



4. WandB interface

Parameter importance with correlation