Philipp Hager, IRLab Amsterdam
Soos talk - October 7th, 2025

Taming Hydra

Have you ever found yourself here?

```
# other params:
flags.DEFINE_integer( 'max_train_iteration', 0,
                      'Limit on the iterations of training (0: no limit).')
flags.DEFINE_integer( 'steps_per_checkpoint', 20,
                      'How many training steps to do per checkpoint.')
flags.DEFINE_string(  'steps_no_checkpoint', '20',
                      'How many training steps to do before any checkpoint.')
flags.DEFINE_boolean( 'predict', False,
                      'set for predicting test data using learned model in "ckpt_dir" directory')

flags.DEFINE_boolean( 'train_and_predict', True,
                      'set for training on train data and then predicting test data using learned model in "ckpt_dir" directory')
flags.DEFINE_string(  'train_and_predict_output', 'train_and_predict_output.txt',
                      'comma separated output file for hyper parameter tuning purposes.')
# flags.DEFINE_string(  'perplexity_output', 'perplexity_output.txt',
#                       'comma separated output file for hyper parameter tuning purposes.')
# flags.DEFINE_string(  'perplexity_prob_fn', 'softmax',
#                       'function used for converting logits to probs for perplexity. either "softmax", "min_max" or "sigmoid".')

flags.DEFINE_string(  'slurm_job_id', '0',
                      'job id (and task id in case of array) from slurm')
flags.DEFINE_boolean( 'test_per_eval', True, 'debugging flag. do not set!')
# use sgd instead of adam
flags.DEFINE_string(  'eval_method', 'loss', 'either "metric" or "loss"')
flags.DEFINE_integer( 'nonimproving_steps', 50, 'stop training after this many checkpoints have been seen without validation improvement.')


DEFINED_FLAGS = set()
for k, _ in FLAGS.__flags.items():
  if not k in NON_DEFINED_FLAGS:
    DEFINED_FLAGS.add(k)
```

Or even here?

```
./baseline/yahoo_set1.model -l conf/sweep/1e-10.json -k 0 -g dcg -b random --eta 1.0
./baseline/yahoo_set1.model -l conf/sweep/1e-10.json -k 0 -g dcg -b random --eta 2.0
./baseline/yahoo_set1.model -l conf/sweep/1e-10.json -k 0 -g dcg -b perfect
./baseline/yahoo_set1.model -l conf/sweep/1e-10.json -k 10 -g hinge -b position --eta 1.0
./baseline/yahoo_set1.model -l conf/sweep/1e-10.json -k 10 -g hinge -b position --eta 2.0
./baseline/yahoo_set1.model -l conf/sweep/1e-10.json -k 10 -g hinge -b random --eta 1.0
./baseline/yahoo_set1.model -l conf/sweep/1e-10.json -k 10 -g hinge -b random --eta 2.0
./baseline/yahoo_set1.model -l conf/sweep/1e-10.json -k 10 -g hinge -b perfect
./baseline/yahoo_set1.model -l conf/sweep/1e-10.json -k 10 -g dcg -b position --eta 1.0
./baseline/yahoo_set1.model -l conf/sweep/1e-10.json -k 10 -g dcg -b position --eta 2.0
./baseline/yahoo_set1.model -l conf/sweep/1e-10.json -k 10 -g dcg -b random --eta 1.0
./baseline/yahoo_set1.model -l conf/sweep/1e-10.json -k 10 -g dcg -b random --eta 2.0
./baseline/yahoo_set1.model -l conf/sweep/1e-10.json -k 10 -g dcg -b perfect
./baseline/yahoo_set1.model -l conf/sweep/1e-10.json -k 0 -g hinge -b position --eta 1.0 --deployments 200000 400000 600000 800000
./baseline/yahoo_set1.model -l conf/sweep/1e-10.json -k 0 -g hinge -b position --eta 2.0 --deployments 200000 400000 600000 800000
./baseline/yahoo_set1.model -l conf/sweep/1e-10.json -k 0 -g hinge -b random --eta 1.0 --deployments 200000 400000 600000 800000
./baseline/yahoo_set1.model -l conf/sweep/1e-10.json -k 0 -g hinge -b random --eta 2.0 --deployments 200000 400000 600000 800000
./baseline/yahoo_set1.model -l conf/sweep/1e-10.json -k 0 -g hinge -b perfect --deployments 200000 400000 600000 800000
./baseline/yahoo_set1.model -l conf/sweep/1e-10.json -k 0 -g dcg -b position --eta 1.0 --deployments 200000 400000 600000 800000
./baseline/yahoo_set1.model -l conf/sweep/1e-10.json -k 0 -g dcg -b position --eta 2.0 --deployments 200000 400000 600000 800000
./baseline/yahoo_set1.model -l conf/sweep/1e-10.json -k 0 -g dcg -b random --eta 1.0 --deployments 200000 400000 600000 800000
./baseline/yahoo_set1.model -l conf/sweep/1e-10.json -k 0 -g dcg -b random --eta 2.0 --deployments 200000 400000 600000 800000
./baseline/yahoo_set1.model -l conf/sweep/1e-10.json -k 0 -g dcg -b perfect --deployments 200000 400000 600000 800000
./baseline/yahoo_set1.model -l conf/sweep/1e-10.json -k 10 -g hinge -b position --eta 1.0 --deployments 200000 400000 600000 800000
./baseline/yahoo_set1.model -l conf/sweep/1e-10.json -k 10 -g hinge -b position --eta 2.0 --deployments 200000 400000 600000 800000
./baseline/yahoo_set1.model -l conf/sweep/1e-10.json -k 10 -g hinge -b random --eta 1.0 --deployments 200000 400000 600000 800000
./baseline/yahoo_set1.model -l conf/sweep/1e-10.json -k 10 -g hinge -b random --eta 2.0 --deployments 200000 400000 600000 800000
./baseline/yahoo_set1.model -l conf/sweep/1e-10.json -k 10 -g hinge -b perfect --deployments 200000 400000 600000 800000
./baseline/yahoo_set1.model -l conf/sweep/1e-10.json -k 10 -g dcg -b position --eta 1.0 --deployments 200000 400000 600000 800000
./baseline/yahoo_set1.model -l conf/sweep/1e-10.json -k 10 -g dcg -b position --eta 2.0 --deployments 200000 400000 600000 800000
./baseline/yahoo_set1.model -l conf/sweep/1e-10.json -k 10 -g dcg -b random --eta 1.0 --deployments 200000 400000 600000 800000
./baseline/yahoo_set1.model -l conf/sweep/1e-10.json -k 10 -g dcg -b random --eta 2.0 --deployments 200000 400000 600000 800000
./baseline/yahoo_set1.model -l conf/sweep/1e-10.json -k 10 -g dcg -b perfect --deployments 200000 400000 600000 800000
```

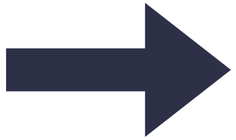
Or often write…

```
if main_args.sim == "mms" or main_args.sim == "MultiModalSearchSim":
    sim_class = MultiModalSearchSim
elif main_args.sim == "mmr" or main_args.sim == "MultiModalRecSim":
    sim_class = MultiModalRecSim
elif main_args.sim == "sss" or main_args.sim == "SemiSyntheticSearchSim":
    sim_class = SemiSyntheticSearchSim
else:
    raise NotImplementedError("This simulator has not been implemented yet.")

if main_args.policy == "random":
    policy_class = RandomPolicy
elif main_args.policy == "oracle":
    policy_class = OraclePolicy
elif main_args.policy == "logged":
    policy_class = LoggedPolicy
elif main_args.policy == "plackettluce":
    policy_class = PlackettLuceLoggedPolicy
elif main_args.policy == "plackettluceoracle":
    policy_class = PlackettLuceNoisyOracle
else:
    raise NotImplementedError("This logging policy has not been implemented yet.")
```

What is config management?

```
python train.py \
    --optimizer adamw \
    --learning_rate 0.0003 \
    --weight_decay 0.9 \
    --model model_a \
    --layers 5 \
    --hidden_dim 100 \
    --dropout 0.2 \
    --activation gelu \
...
```



config.yaml

```yaml
optimizer: adamw
learning_rate: 0.0003
weight_decay: 0.9
```

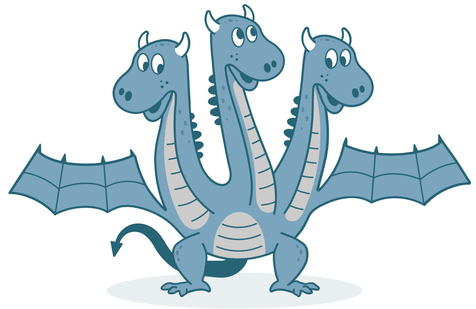
model/model_a.yaml

```yaml
layers: 5
hidden_dim: 100
activation: gelu
dropout: 0.2
```

What is Hydra?

Open-source framework from Meta building on the YAML framework OmegaConf

- Composable, hierarchical config files
- Dependency injection
- Sweep parameters in parallel SLURM jobs
- Dynamic working directories
- Logging (not today)



We'll use the Hydra icon to link related tutorials / resources in this talk.

Configuration Management

Basic setup

```
pip install hydra-core
```

main.py

```python
import hydra
from omegaconf import DictConfig, OmegaConf

@hydra.main(
    version_base="1.3",
    config_path="config/",
    config_name="config"
)
def main(config: DictConfig):
    print(OmegaConf.to_yaml(config))

if __name__ == "__main__":
    main()
```

Project

```
.
├── config/
│   └── config.yaml
└── main.py
```

Primitive types

config/config.yaml

```yaml
experiment: “my_awesome_experiment” # Strings
learning_rate: 0.001 # Numbers
early_stopping: True # Booleans
layers: # Lists
  - 256
  - 128
  - 64
splits: # Dictionaries
  train: 60
  val: 20
  test: 20
random_state: ??? # Missing value
```

Interpolation

config/config.yaml

```yaml
# Interpolate variables:
out_dir: "results/${experiment}/${now:%Y-%m-%d}/"

# Access environment variables:
api_key: ${oc.env:API_KEY}

# Resorts to default value if not set:
data_dir: ${oc.env:DATA_DIR,"my/default/"}
```

Use cases: Reference variables across files, dynamic path generation,
change configs based on laptop/cluster, avoid storing secrets, optional values, etc.

Accessing variables

main.py

```
print(config.learning_rate) # Attribute-style access
# 0.001

print(config["layers"]) # Dictionary-style access
# [256, 128, 64]

print(config.out_dir) # Assemble variables at runtime
# results/my_awesome_experiment/2025-10-07/

print(config.api_key)
# Error: Environment variable 'API_KEY' not found
```

Config groups

config.yaml

```yaml
optimizer: adam
learning_rate: 0.0003
weight_decay: 0.9
```

model/model_a.yaml

```yaml
name: model_a
hidden_dim: 100
activation: gelu
dropout: 0.2
```

model/model_b.yaml

```yaml
name: model_b
hidden_dim: 512
activation: elu
```

output

```yaml
optimizer: adam
learning_rate: 0.0003
weight_decay: 0.9
model:
  name: model_a
  hidden_dim: 100
  activation: gelu
  dropout: 0.2
```

```
python main.py +model=model_a
```

Use cases: Switch between datasets, models, simulators, ...

Defaults & packages

config.yaml

```yaml
defaults:
  - _self_
  - model: model_a
  - model@retriever: model_a
  - model@reranker: model_b
```

```yaml
model:
  name: Model A
retriever:
  name: Model A
reranker:
  name: Model B
```

Project

```
.
├── config/
│   ├── model/
│   │   ├── model_a.yaml
│   │   └── model_b.yaml
│   └── config.yaml
└── main.py
```

Runs & overrides

config.yaml

```yaml
defaults:
  - _self_
  - model: model_a

learning_rate: 0.003
```

Run default

```
python main.py
```

Override a config value

```
python main.py learning_rate=0.01
```

Append a new config value

```
python main.py +data=msmarco
```

Dependency Injection

Instantiating Python objects from configs

config.yaml

```yaml
defaults:
  - _self_
  - model: model_a
```

config/model/model_a.yaml

```yaml
_target_: src.models.ModelA
name: Model A
layers: 3
...
```

main.py

```python
from hydra.utils import instantiate

def main(config: DictConfig):
    model = instantiate(config.model)
    model(…)
```

Use cases: Init datasets, models, etc. with different arguments

Multiruns & Sweeps

Multirun

Sweep over all combinations of model & random state:

```
python main.py -m model=model_a,model_b random_state=1,2,3
```

```
[2025-10-04 11:34:45,436][HYDRA] Launching 6 jobs locally
[2025-10-04 11:34:45,436][HYDRA]        #0 : model=model_a random_state=1
[2025-10-04 11:34:45,479][HYDRA]        #1 : model=model_a random_state=2
[2025-10-04 11:34:45,524][HYDRA]        #2 : model=model_a random_state=3
[2025-10-04 11:34:45,567][HYDRA]        #3 : model=model_b random_state=1
[2025-10-04 11:34:45,609][HYDRA]        #4 : model=model_b random_state=2
[2025-10-04 11:34:45,655][HYDRA]        #5 : model=model_b random_state=3
```

Working directory management

By default, Hydra creates a new directory per run based on date/time but this can be changed.

Thus, a file saved with `df.to_csv("test.csv")` is not saved in your project but in the run directory.

Use `hydra.utils.get_original_cwd()` to access files in your project root or use absolute paths.

Lastly, Hydra stores the assembled config per job in the job's working directory under **.hydra/config.yaml.**

```
.
├── config/
├── multirun/
│   └── 2025-10-04/
│       └── 15-11-24/  # Sweep
│           ├── 0/     # Job
│           │   └── test.csv
│           ├── 1/     # Job
│           │   └── test.csv
│           └── ...
└── main.py
```

Sweeps

Hydra supports custom parameters to define parameter search ranges.

config/config.yaml

```
python main.py -m \
    layers=range(3,6) \ # Integer/ Float ranges
    model=glob(*) \ # List all configs in directory
    lr=interval(0.00001,0.001) \ # Constrain param tuners
```

Sweepers

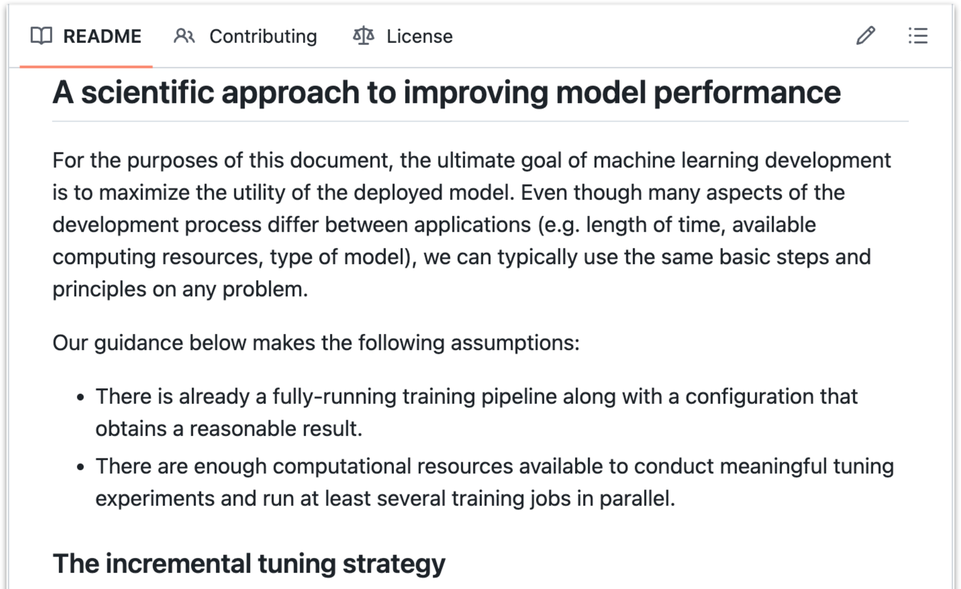
The built-in sweeper performs grid search.

Hydra supports a range of black-box & Bayesian optimizers:

- Ax
- Nevergrad
- Optuna

Typically require parameter ranges and tuning budget.

I prefer incremental tuning strategies over black box



Launchers

Parallel jobs on SLURM using Submitit

```
pip install hydra-submitit-launcher
```

Hydra can apply files on the root config level

config/launcher/slurm.yaml

```yaml
# @package _global_
defaults:
  - override /hydra/launcher: submitit_slurm

hydra:
  launcher:
    mem_gb: 32 # RAM per job
    cpus_per_task: 16 # CPUs per job
    array_parallelism: 8 # Max parallel jobs
    timeout_min: 60 # Timeout
    partition: gpu
    gres: gpu:nvidia_rtx_a6000:1 # GPUs per job
```

Parallel jobs on SLURM using Submitit

```
python main.py -m model=gctr,rctr,dctr +launcher=slurm
```

```
[2025-10-03 15:11:25,368][HYDRA] Submitit 'slurm' sweep output dir : multirun/2025-10-03/15-11-24
[2025-10-03 15:11:25,369][HYDRA]  #0 : model=gctr +launcher=slurm
[2025-10-03 15:11:25,372][HYDRA]  #1 : model=rctr +launcher=slurm
[2025-10-03 15:11:25,376][HYDRA]  #2 : model=dctr +launcher=slurm
```

```
  JOBID PARTITION     NAME     USER ST       TIME  NODES NODELIST(REASON)
         268091_1        gpu     main   phager  R       0:01      1 ilps-cn118
         268091_2        gpu     main   phager  R       0:01      1 ilps-cn118
         268091_0        gpu     main   phager  R       0:02      1 ilps-cn118
```

Some useful Design Patterns

Reuse parameters across files

config/data/msmarco.yaml

```yaml
_target_: src.data.MSMarco
vocab_size: 30_522
```

config/model/mono-bert.yaml

```yaml
_target_: src.models.MonoBERT
vocab_size: ${data.vocab_size}
```

Hyperparameters based on data & model

config/config.yaml

```yaml
defaults:
  - _self_
  - data: msmarco
  - model: splade
  - hyperparameter: “${data}_${model}”
```

Auto-select parameters in next sweep:

```
python main.py data=msmarco,trec_covid
```

Project

```
.
├── config/
│   ├── config.yaml
│   ├── model/
│   │   └── splade
│   ├── data/
│   │   ├── msmarco.yaml
│   │   └── trec_covid.yaml
│   └── hyperparameter/
│       ├── msmarco_splade.yaml
│       └── trec_covid_splade.yaml
└── main.py
```

Different configs between environments / people

config/config.yaml

```yaml
defaults:
  - _self_
  - optional: env
```

config/env.yaml

```yaml
data_dir: /my/local/path
logger:
    wandb_project: my_project
    wandb_entity: my_entity
launcher: slurm
```

Add file to .gitignore before

```
Project
.
├── config/
│   ├── config.yaml
│   └── env.yaml
└── main.py
```

The Sharp Bits

Debugging

When starting out with Hydra, error messages can be confusing, especially those caused by **lazy loading** and **dependency injection**.

Make sure to enable full stacktraces:

```
export HYDRA_FULL_ERROR=1
```

And **print the assembled config** at the beginning of your code.

```
print(OmegaConf.to_yaml(config))
```

Being too modular

By decoupling code and params, Hydra can make your code more opaque:

```
train_data = instantiate(config.train_data)
val_data = instantiate(config.val_data)
model = instantiate(config.model)
trainer = instantiate(config.trainer)

trainer.train(model, train_data, val_data)
```

Instead of making everything modular and configurable,
try to focus on what actually needs to be varied across your experiments.

Hydra structured configs allows to write your configs as data classes
(and enables advanced type checking & validation).

Summary

- Use config managers to escape endless if-else chains and long shell scritps. And avoid writing your own config managers.
- Hydra allows modular and hierarchical parameter configuration and dependency injection.
- Hydra enables parameter sweeps and SLURM management.
- The functionality of Hydra can be overwhelming at times. Start small, don’t try to learn everything all at once.