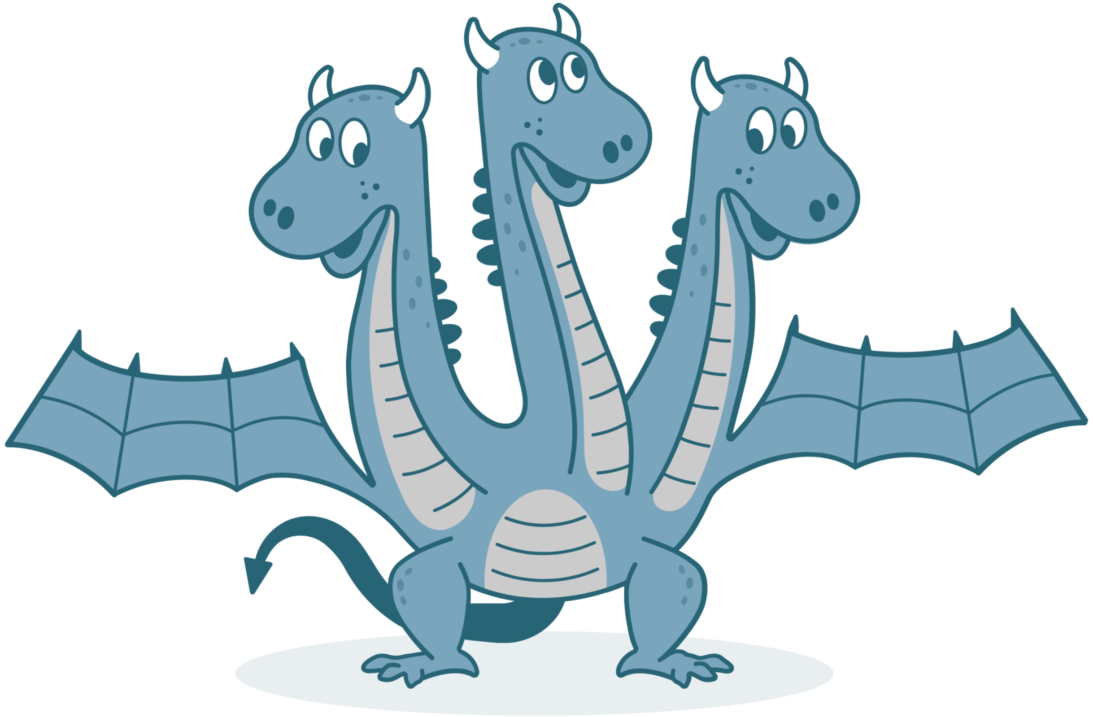
Experiment Configuration with Hydra



https://hydra.cc

Jonathan Carter

ML Experiment Configuration

Configuration

Model Architecture   Dataset(s)   Optimizer   …

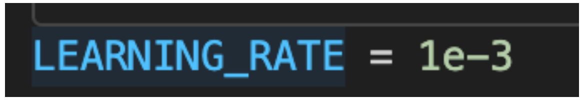
Application

How do we inject configuration into our application?

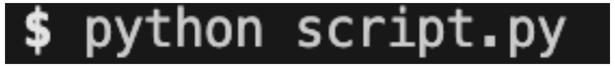
A very bad approach:

Configuration

Hard-coding



```python
LEARNING_RATE = 1e-3
```

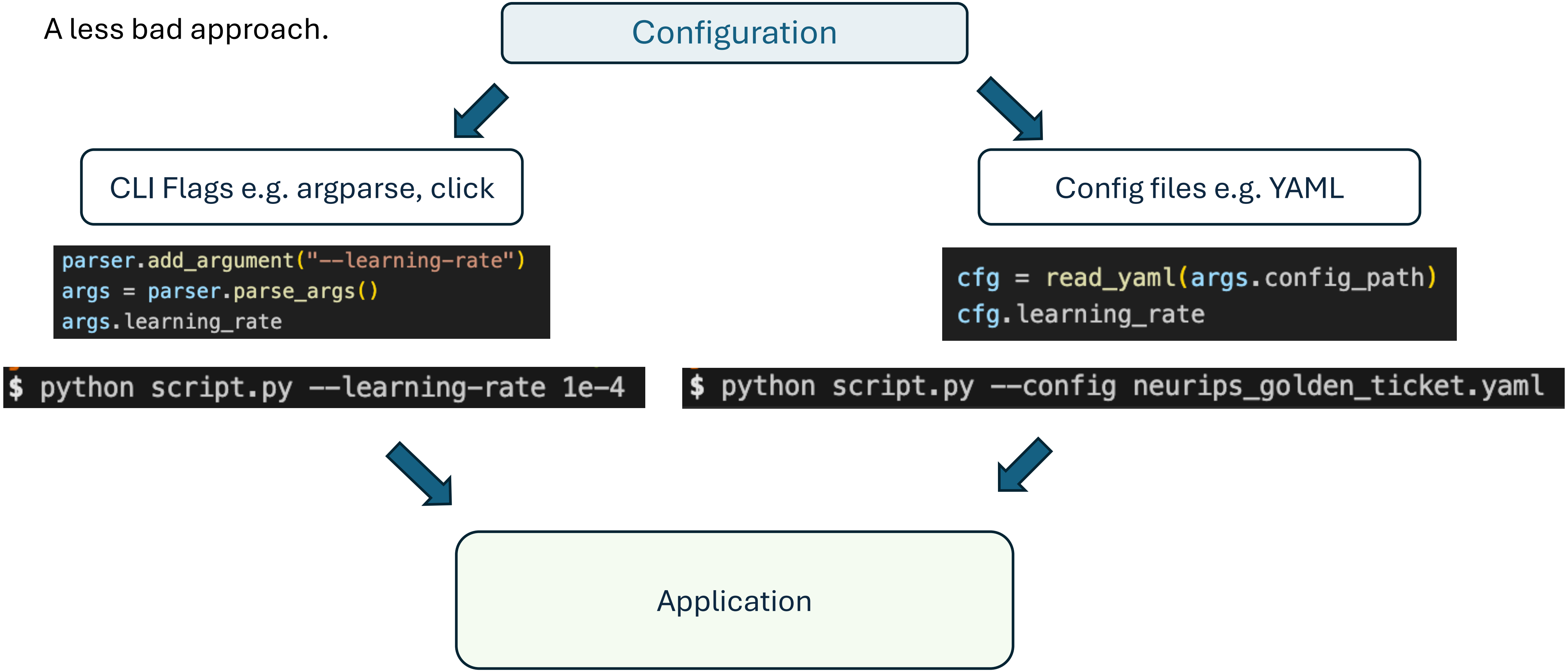


```
$ python script.py
```

Application

How do we inject configuration into our application?

A less bad approach.



Configuration

CLI Flags e.g. argparse, click

```
parser.add_argument("--learning-rate")
args = parser.parse_args()
args.learning_rate
```

```
$ python script.py --learning-rate 1e-4
```

Config files e.g. YAML

```
cfg = read_yaml(args.config_path)
cfg.learning_rate
```

```
$ python script.py --config neurips_golden_ticket.yaml
```

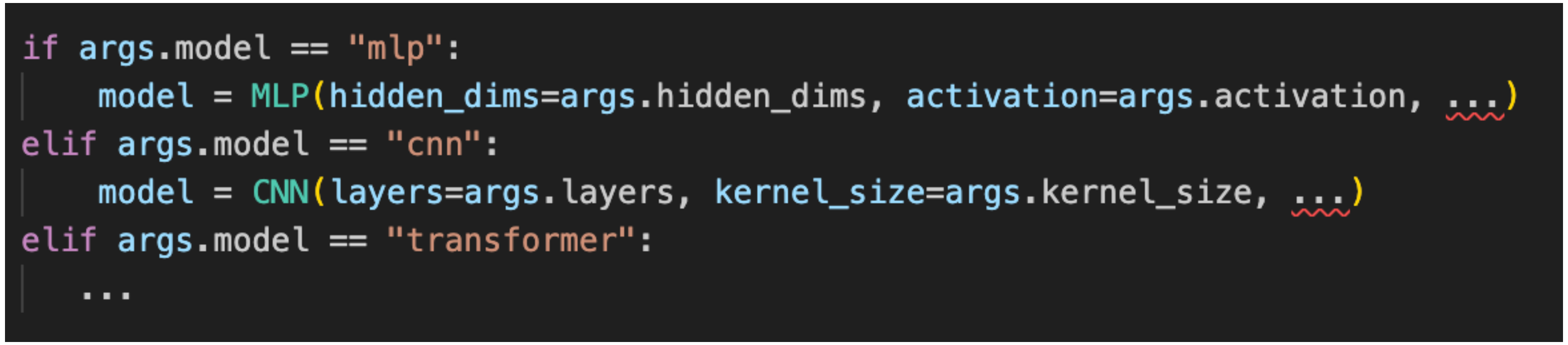
Application

What’s wrong with these approaches?

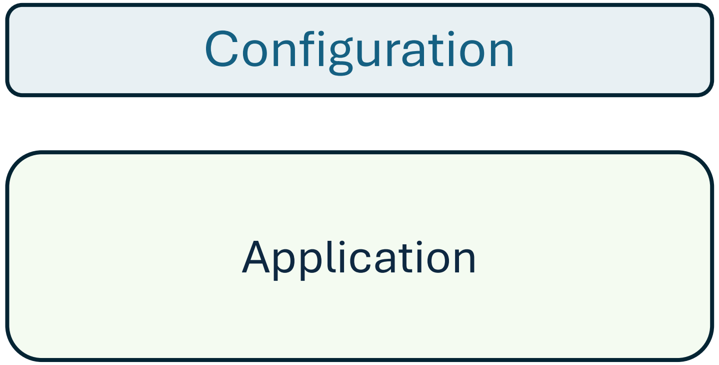
- CLI Flags don’t scale well with ML experiments.
- Config files often end up massively duplicated -> easy to make mistakes.
- Both result in large amounts of ‘boilerplate factory’ code. Difficult to maintain.

Boilerplate factory code

Code that turns configuration into objects within the application.



```
if args.model == "mlp":
    model = MLP(hidden_dims=args.hidden_dims, activation=args.activation, ...)
elif args.model == "cnn":
    model = CNN(layers=args.layers, kernel_size=args.kernel_size, ...)
elif args.model == "transformer":
    ...
```



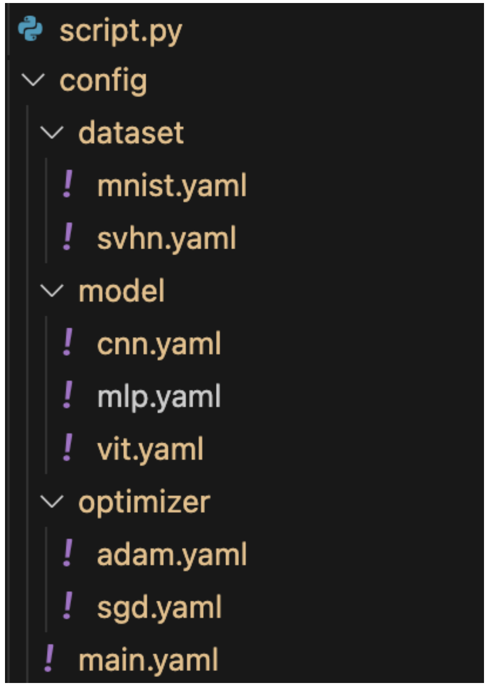
Configuration

Application

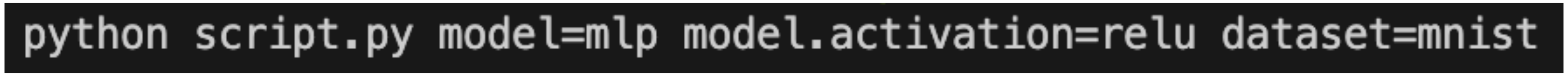
Hydra - Basics

Framework for elegantly configuring complex applications in Python (not just ML research).

Configuration composed from structured, *hierarchical* configuration files and command line overrides.

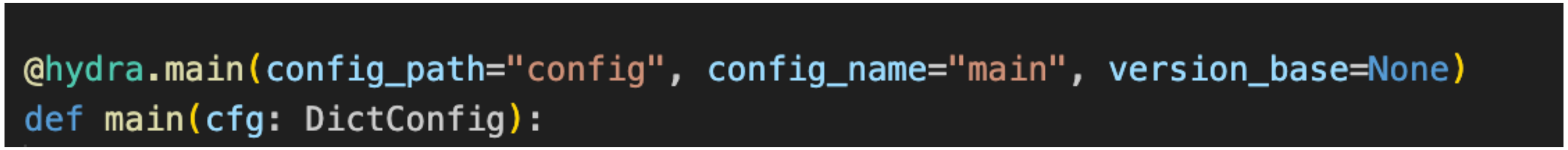


```
script.py
config
  dataset
    mnist.yaml
    svhn.yaml
  model
    cnn.yaml
    mlp.yaml
    vit.yaml
  optimizer
    adam.yaml
    sgd.yaml
  main.yaml
```



```
python script.py model=mlp model.activation=relu dataset=mnist
```

Configuration



```python
@hydra.main(config_path="config", config_name="main", version_base=None)
def main(cfg: DictConfig):
```

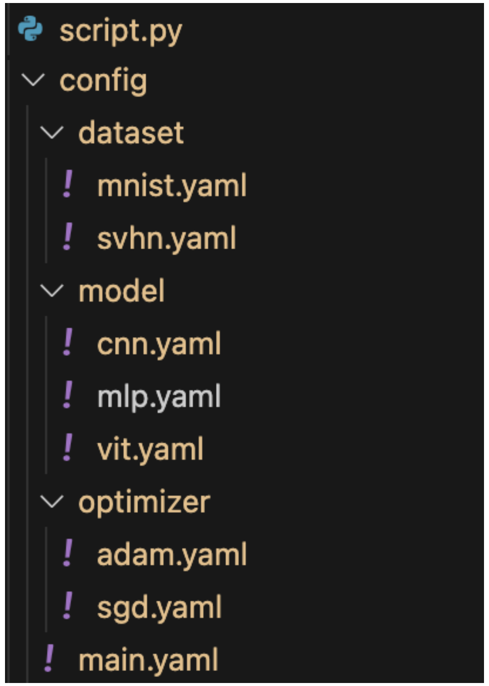
Application

https://hydra.cc

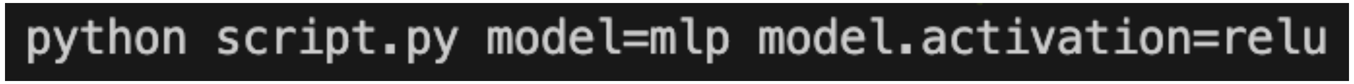
Jonathan Carter

Hydra – Object Instantiation

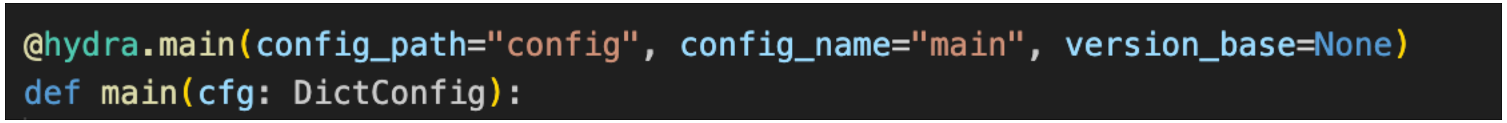
Can eliminate boilerplate by directly instantiating objects from config.



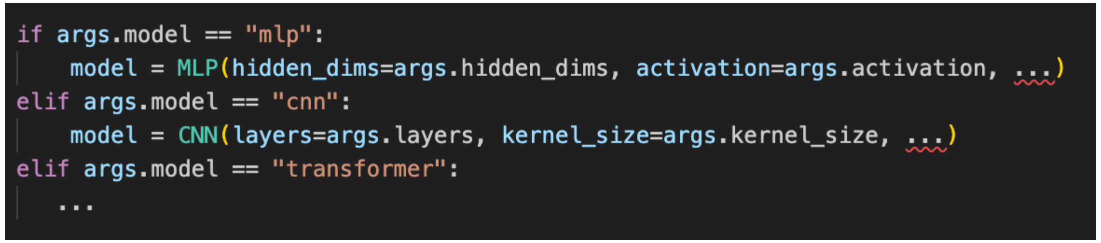
```
script.py
config
  dataset
    mnist.yaml
    svhn.yaml
  model
    cnn.yaml
    mlp.yaml
    vit.yaml
  optimizer
    adam.yaml
    sgd.yaml
  main.yaml
```



```
python script.py model=mlp model.activation=relu
```

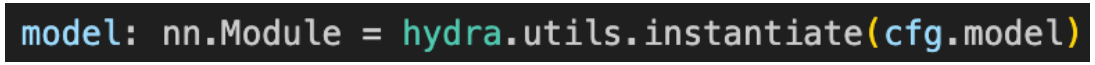


```python
@hydra.main(config_path="config", config_name="main", version_base=None)
def main(cfg: DictConfig):
```



```python
if args.model == "mlp":
    model = MLP(hidden_dims=args.hidden_dims, activation=args.activation, ...)
elif args.model == "cnn":
    model = CNN(layers=args.layers, kernel_size=args.kernel_size, ...)
elif args.model == "transformer":
    ...
```

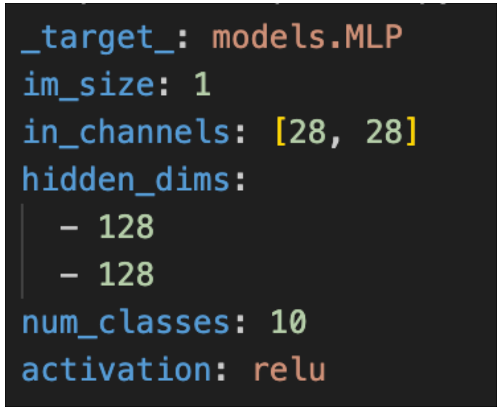
✗



```python
model: nn.Module = hydra.utils.instantiate(cfg.model)
```

✓

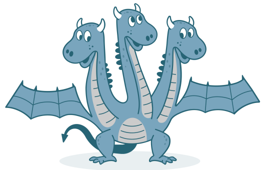
mlp.yaml



```yaml
_target_: models.MLP
im_size: 1
in_channels: [28, 28]
hidden_dims:
  - 128
  - 128
num_classes: 10
activation: relu
```

Can modify **everything** from the command line:

- From high-level options e.g. the model class.
- Right down to low-level options e.g. feedforward dim within a layer.

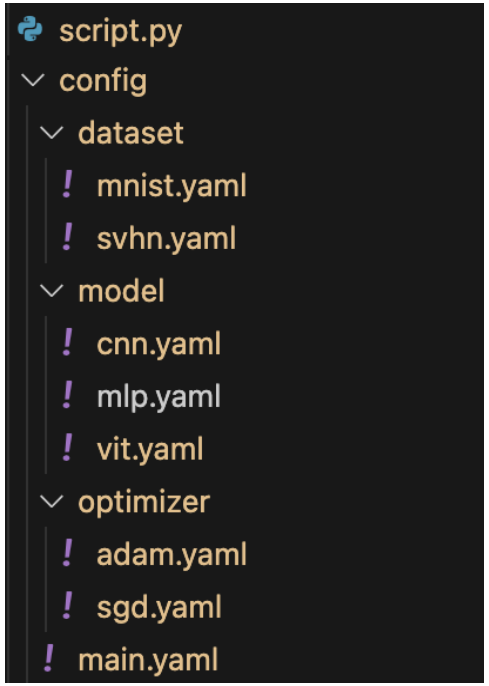


https://hydra.cc

Jonathan Carter

Hydra – Multirun

Trivially sweep over configuration options with comma-separated arguments.



```
script.py
config
  dataset
    mnist.yaml
    svhn.yaml
  model
    cnn.yaml
    mlp.yaml
    vit.yaml
  optimizer
    adam.yaml
    sgd.yaml
  main.yaml
```

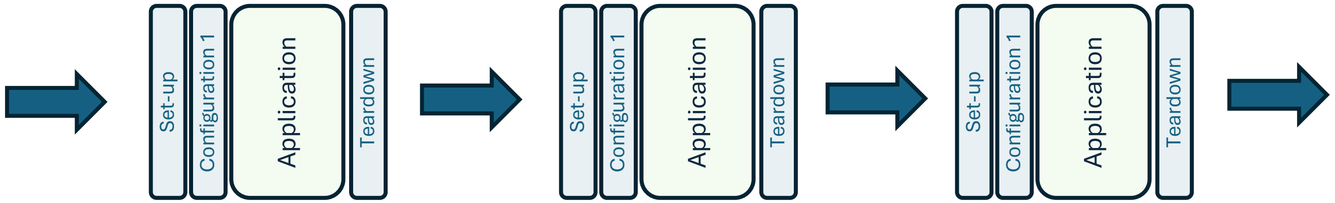
Sweep over all combination of models and datasets:

```
python script.py --multirun model=mlp,cnn,vit dataset=mnist,svhn
```

Sweep over optimizer learning rates:

```
python script.py optimizer=adam optimizer.lr=1e-4,5e-4,1e-3
```

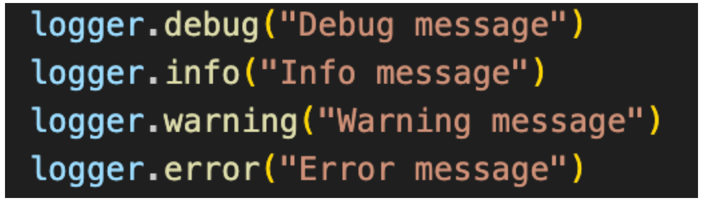
Jobs run serially by default:



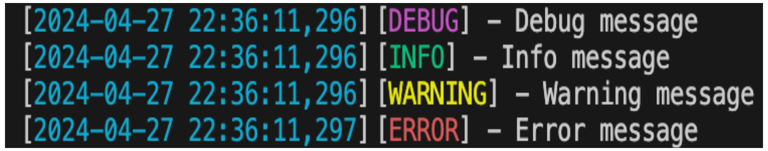
https://hydra.cc

Hydra – Further reading

1. Logging: Simplifies configuration of the Python standard logging library.

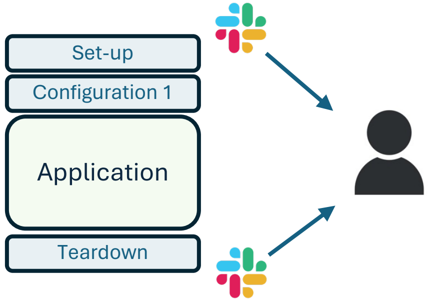


```
logger.debug("Debug message")
logger.info("Info message")
logger.warning("Warning message")
logger.error("Error message")
```

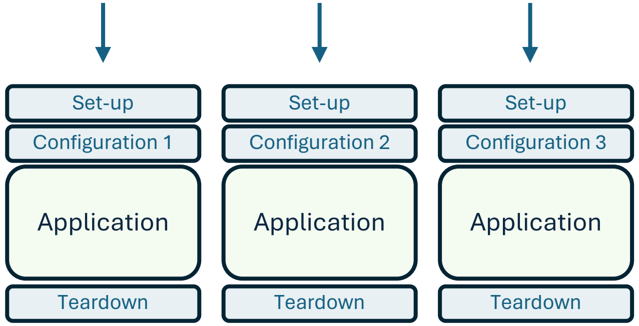


```
[2024-04-27 22:36:11,296] [DEBUG] - Debug message
[2024-04-27 22:36:11,296] [INFO] - Info message
[2024-04-27 22:36:11,296] [WARNING] - Warning message
[2024-04-27 22:36:11,297] [ERROR] - Error message
```

2. Custom Callbacks: e.g. have Hydra send you a Slack message when jobs start/end.



3. Launchers: Execute *all* your configuration options in parallel on any backend e.g. a Slurm cluster.



https://hydra.cc

1. https://hydra.cc/docs/configure_hydra/logging/
2. https://hydra.cc/docs/experimental/callbacks/
3. https://hydra.cc/docs/plugins/submitit_launcher/

Jonathan Carter