An Introduction of Hyperparameter Optimization with BioHPC

Peng Lian

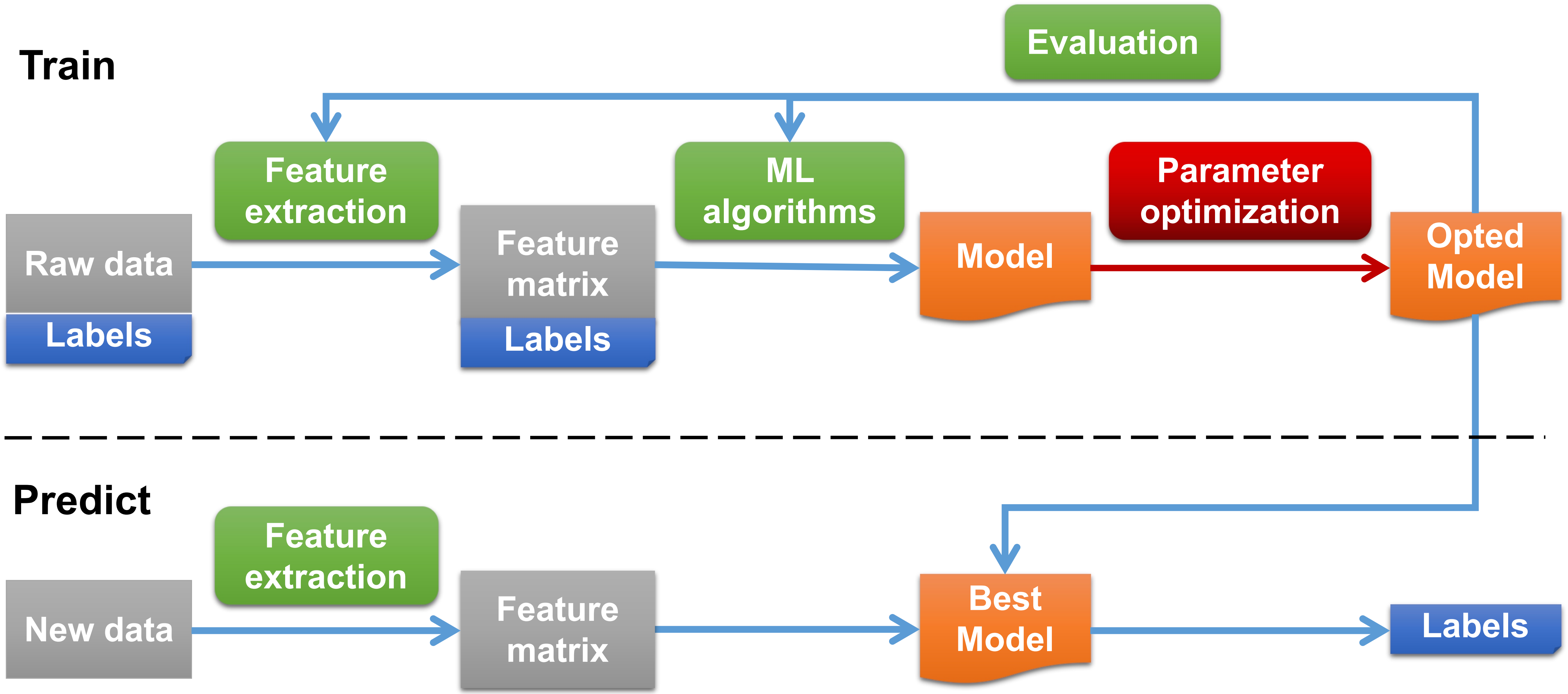
09/08/2021

Some Concepts to Think

Parameters
Hyperparameters

Analytical function
Black box function

A Flowchart of Machine Learning
(Supervised Model)

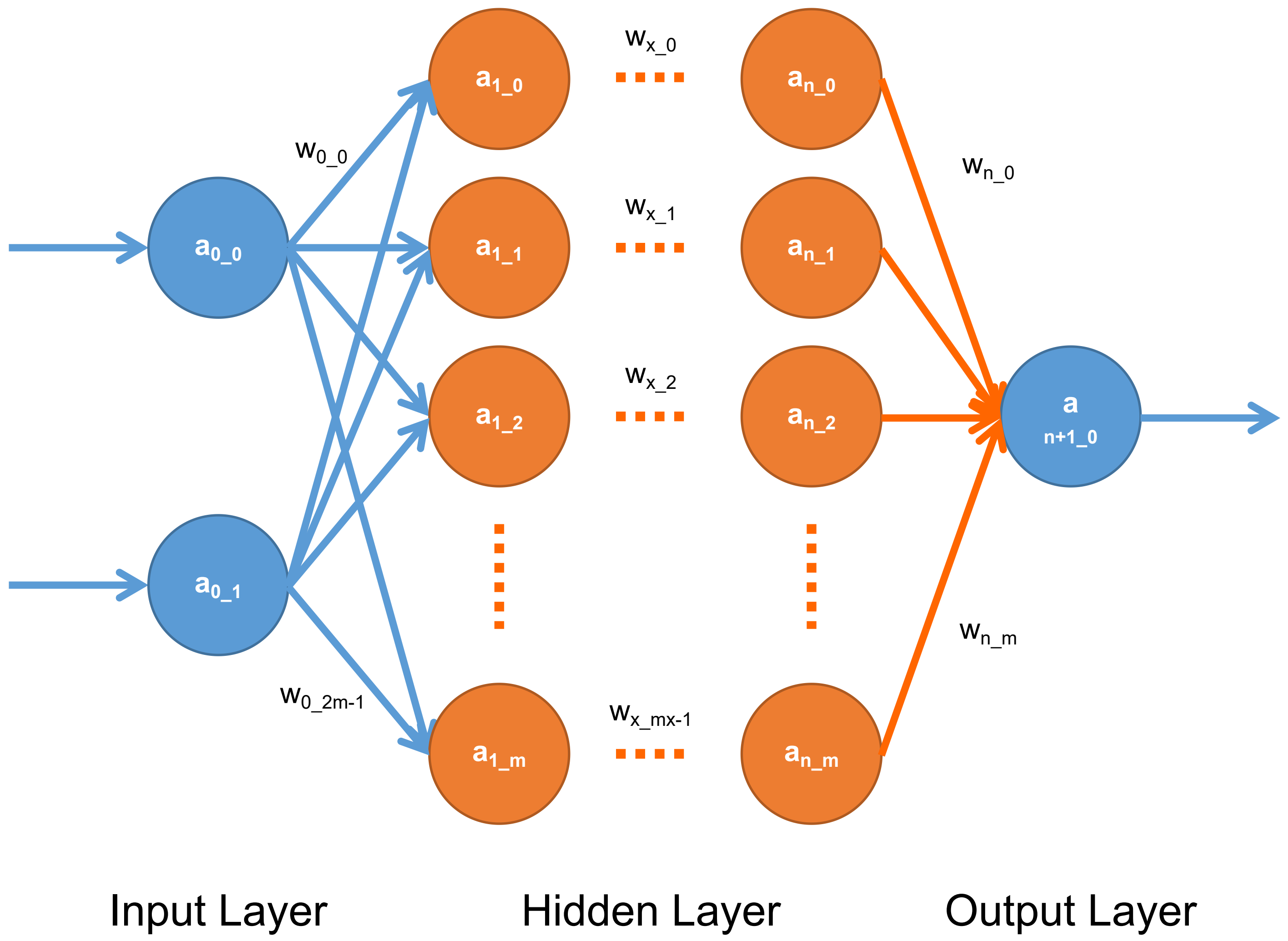


What is a hyperparameter in ML?

A **hyperparameter** is a parameter whose value is set **before** the learning process begins.

E.g. # of perceptrons in a layer, # of hidden layers.

By contrast, the values of **other parameters** are derived via training.



Algorithms in Hyperparameter Optimization

Hyperparameter optimization is a **black box optimization problem**, where the **analytical expression** for the objective function ***f(x)*** or its **derivatives** are not available. Evaluation of ***f*** is restricted to sampling at a point ***x*** and getting a possible nosiy response.

- Random search
- Grid search
- Bayesian optimization
- Hyperband
- ... ...

Good for **cheap** black box function ***f***

Good for **expensive** black box function ***f***

Bayesian Optimization Intro

$$x^* = \underset{x \in X}{\operatorname{argmin}} f(x)$$

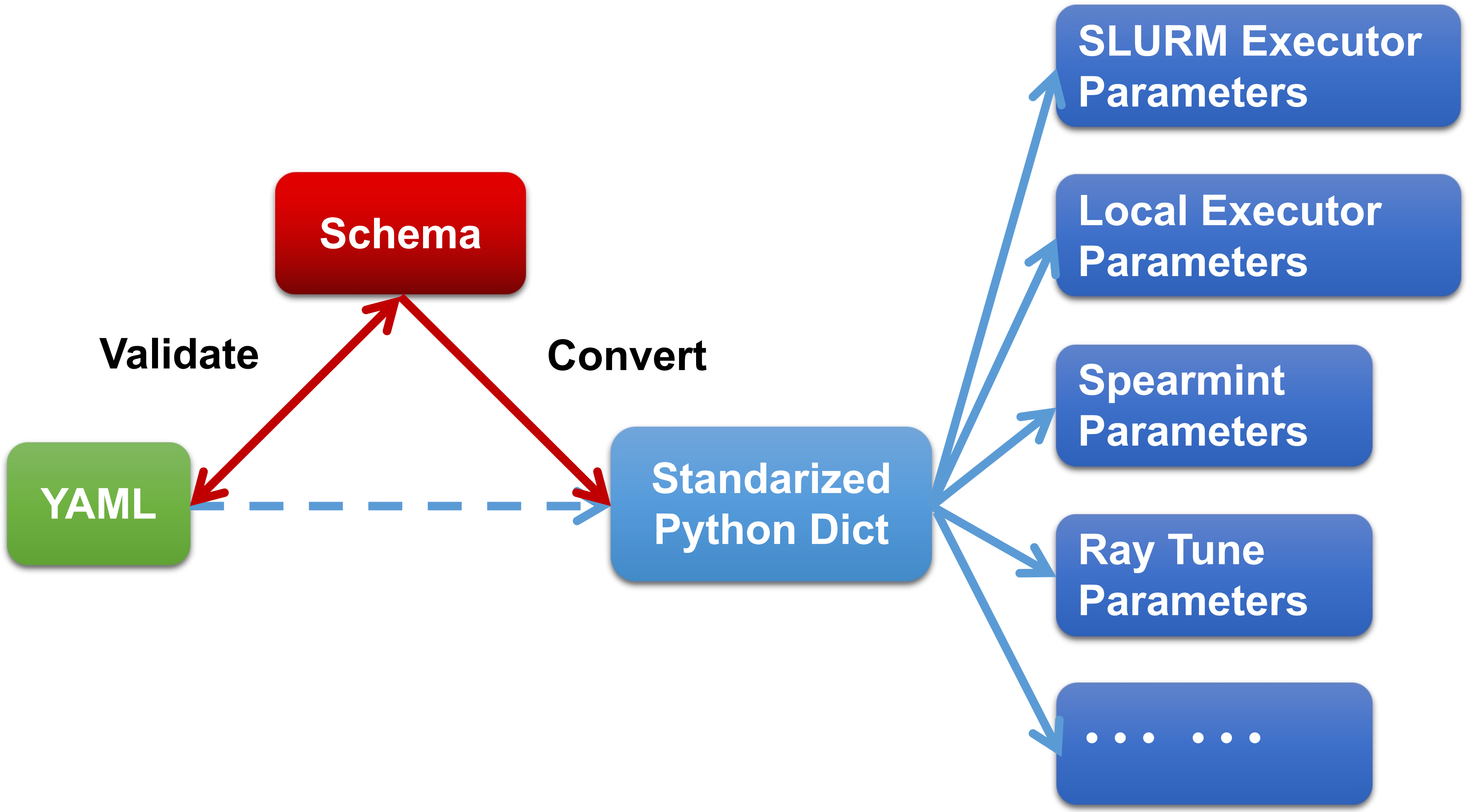
f: Objective function, your to-be-optimized model. E.g. the MLP in page 3
x: A group of hyperparameters. E.g (m, n) in the MLP example
X: All the possible combinations of the hyperparameters

Input: f, X, S, GP

```
$D = \text{InitSamples}(f, X)$
for i to T:
    $p(y|x,D) = \text{FitModel}(GP, D)$
    $x_i = \underset{x \in X}{\operatorname{argmax}} S(x, p(y|x, D))$
    $y_i = \text{f}(x_i)$
    $D = D \bigcup (x_i, y_i)$
```

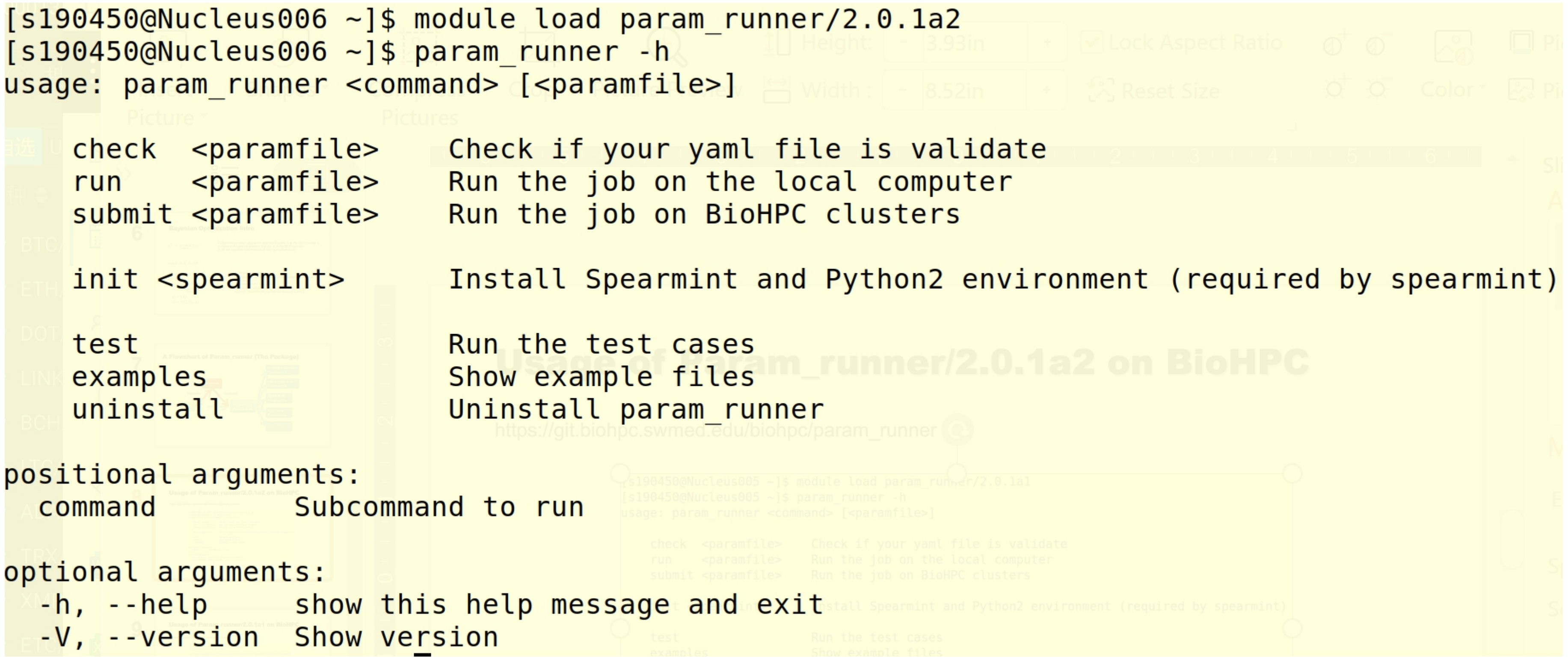
S: Acquisition function
GP: Gaussian process prior. E.g. a Gaussian distribution function.
y: Response of *f. E.g. y=f(x)*
D: Historical set of x and y. E.g. $D = \{(x_1, y_1), ..., (x_i, y_i)\}$
$p(y|x,D)$: Proability of y on the condition of x and D.

A Flowchart of Param_runner (The Package)



Usage of Param_runner/2.0.1a2 on BioHPC

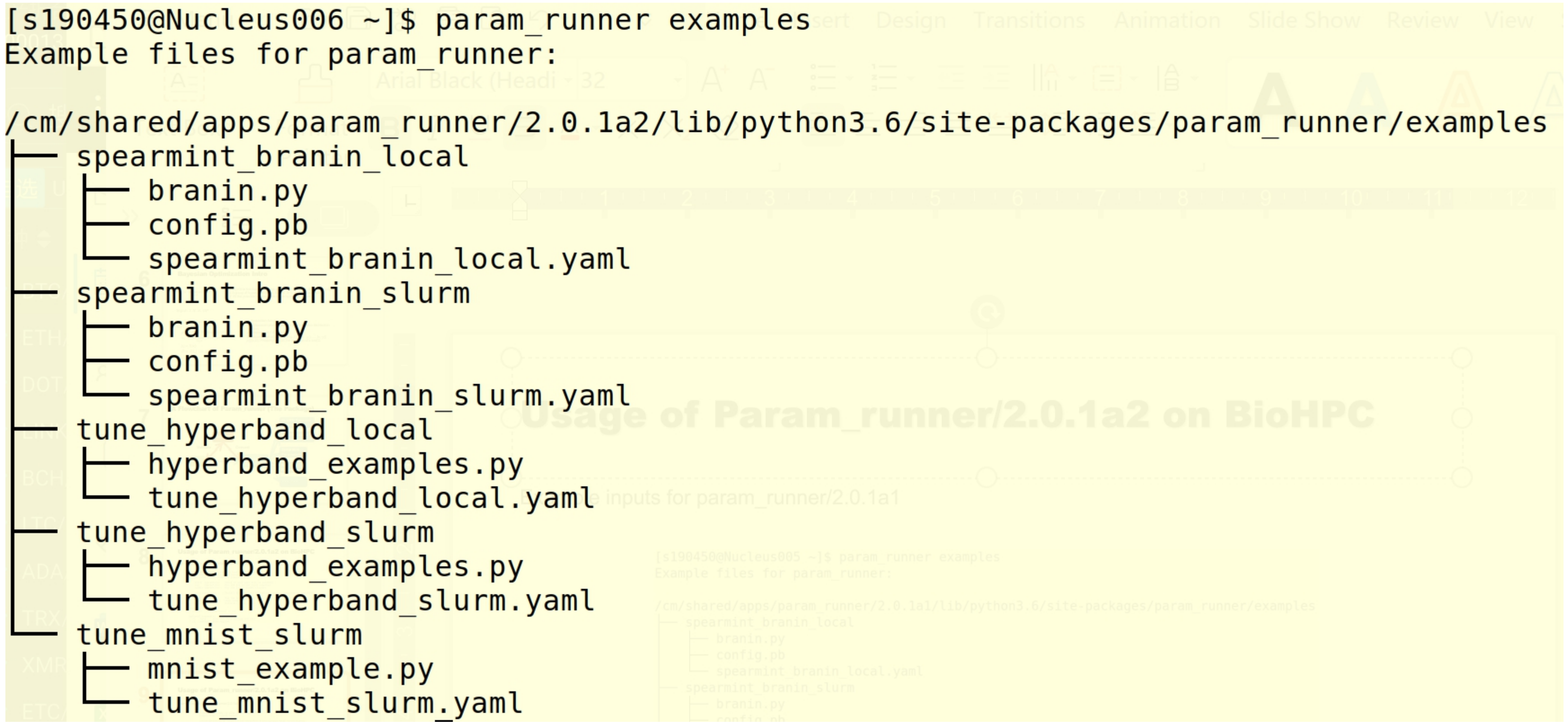
https://git.biohpc.swmed.edu/biohpc/param_runner



```
[s190450@Nucleus006 ~]$ module load param_runner/2.0.1a2
[s190450@Nucleus006 ~]$ param_runner -h
usage: param_runner <command> [<paramfile>]

    check  <paramfile>    Check if your yaml file is validate
    run    <paramfile>    Run the job on the local computer
    submit <paramfile>    Run the job on BioHPC clusters

    init <spearmint>      Install Spearmint and Python2 environment (required by spearmint)

    test                  Run the test cases
    examples              Show example files
    uninstall             Uninstall param_runner

positional arguments:
  command         Subcommand to run

optional arguments:
  -h, --help      show this help message and exit
  -V, --version   Show version
```

Usage of Param_runner/2.0.1a2 on BioHPC

Example inputs for param_runner/2.0.1a2



```
[s190450@Nucleus006 ~]$ param_runner examples
Example files for param_runner:

/cm/shared/apps/param_runner/2.0.1a2/lib/python3.6/site-packages/param_runner/examples
├── spearmint_branin_local
│   ├── branin.py
│   ├── config.pb
│   └── spearmint_branin_local.yaml
├── spearmint_branin_slurm
│   ├── branin.py
│   ├── config.pb
│   └── spearmint_branin_slurm.yaml
├── tune_hyperband_local
│   ├── hyperband_examples.py
│   └── tune_hyperband_local.yaml
├── tune_hyperband_slurm
│   ├── hyperband_examples.py
│   └── tune_hyperband_slurm.yaml
└── tune_mnist_slurm
    ├── mnist_example.py
    └── tune_mnist_slurm.yaml
```

Usage of Param_runner/2.0.1a2 on BioHPC

```
# "cpus_per_task" is required for every kind of jobs.
# To run the job on BioHPC cluster, "partition" and "time_limit" are also required.

# Cluster partition to use
partition: GPU

# Total number of nodes to use
nodes: 2

# Number of CPUs required by each task
cpus_per_task: 4

# Number of GPUs required by each task
gpus_per_task: 1

# Time limit
time_limit: 3-00:00:00

# Tune settings
optimizer: ray_tune
rt_function_file: hyperband_examples.py
```

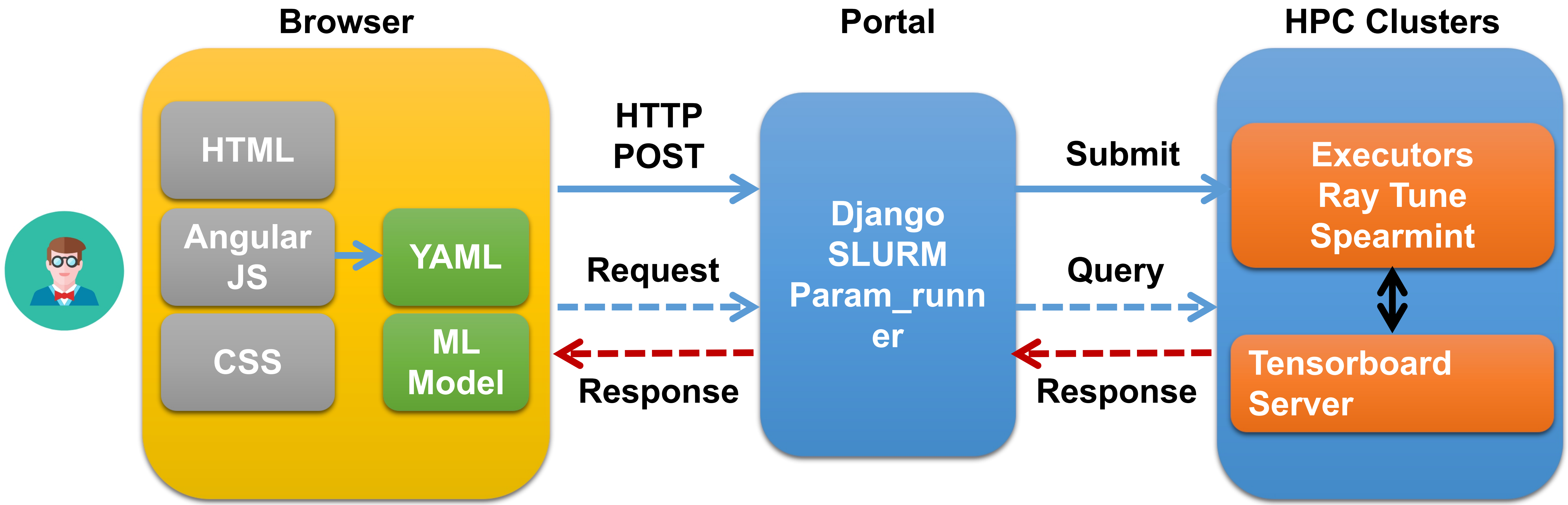
Usage of Param_runner/2.0.1a2 on BioHPC

```
#!/usr/bin/env python

import argparse
import json
import os
import random

import numpy as np

import ray
from ray.tune import Trainable, run, Experiment, sample_from
from ray.tune.schedulers import HyperBandScheduler


class MyTrainableClass(Trainable):
    """Example agent whose learning curve is a random sigmoid.

    The dummy hyperparameters "width" and "height" determine the
slope and
    maximum reward value reached.
    """

    def _setup(self, config):
        self.timestep = 0

    def _train(self):
        self.timestep += 1
        v = np.tanh(float(self.timestep) / self.config.get("width",
1))
        v *= self.config.get("height", 1)

        # Here we use `episode_reward_mean`, but you can also report
other
        # objectives such as loss or accuracy.
        return {"episode_reward_mean": v}

    def _save(self, checkpoint_dir):
        path = os.path.join(checkpoint_dir, "checkpoint")
        with open(path, "w") as f:
            f.write(json.dumps({"timestep": self.timestep}))
        return path

    def _restore(self, checkpoint_path):
        with open(checkpoint_path) as f:
            self.timestep = json.loads(f.read())["timestep"]
```

```
if __name__ == "__main__":
    parser = argparse.ArgumentParser()
    parser.add_argument(
        "--smoke-test", action="store_true", help="Finish quickly
for testing")
    args, _ = parser.parse_known_args()

    # Hyperband early stopping, configured with
`episode_reward_mean` as the
    # objective and `training_iteration` as the time unit,
    # which is automatically filled by Tune.
    hyperband = HyperBandScheduler(
        time_attr="training_iteration",
        metric="episode_reward_mean",
        mode="max",
        max_t=100)

    exp = Experiment(
        name="hyperband_test",
        run=MyTrainableClass,
        num_samples=10,
        stop={"training_iteration": 1 if args.smoke_test else 99999},
        config={
            "width": sample_from(lambda spec: 10 + int(90 *
random.random())),
            "height": sample_from(lambda spec: int(100 *
random.random()))
        })


    # The reserved lines. The value will be assigned by SLURM
    ray.init(redis_address=os.environ["RAY_HEAD_IP"]) # Assigned by
SLURM

    # The local_dir controls where to save the intermediate files.
The default place, "~/ray_results", will
    # cause file flush issuses to ray on our cluster.
    exp.spec['local_dir'] = os.environ["Localdir"]
    run(exp, scheduler=hyperband, resources_per_trial={'gpu':
os.environ["NUM_GPUs"]})
```

https://github.com/ray-project/ray/tree/master/python/ray/tune/examples

A Flowchart of Param_runner (Web Interface)



Param_runner/2.0.1a2 Web Interface

BioHPC Portal ----> Cloud Services ----> Parameter Runner

Currently, available on both the production and test server of BioHPC Portal.

Questions?