Machine Learning for Columnar High Energy Physics Analysis

Elliott Kauffman - Princeton University
Alexander Held - University of Wisconsin-Madison
Oksana Shadura - University of Nebraska-Lincoln

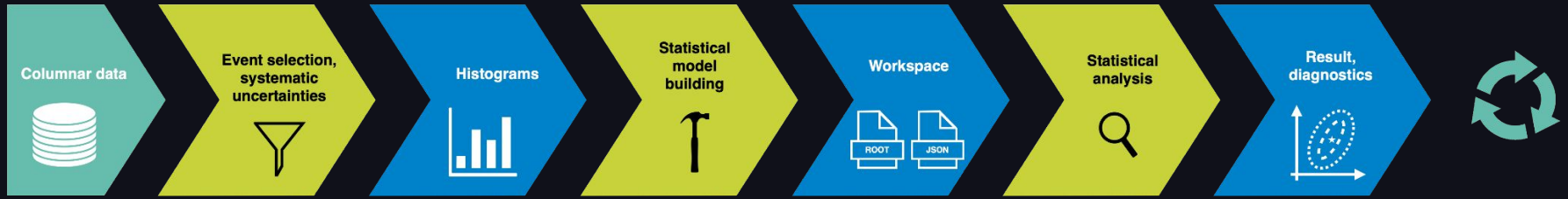
This work was supported by the U.S. National Science Foundation (NSF) Cooperative Agreement OAC-1836650 (IRIS-HEP)

Outline

1. Analysis Grand Challenge Overview
2. Strategy for Integrating Machine Learning
3. Training Pipeline
4. Inference Pipeline

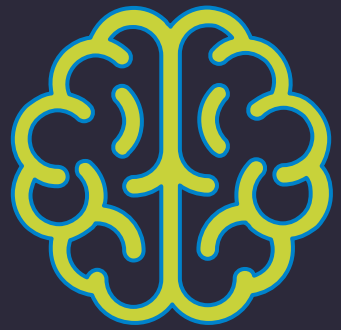
Analysis Grand Challenge

- ➔ Demonstrate **realistic analysis workflow** in anticipation of HL-LHC requirements
- ➔ Task: ttbar cross-section measurement using **2015 CMS Open Data**
  - ◆ Interactive and accessible user interface
  - ◆ **Open Data** - everyone can participate!
- ➔ Include all aspects of typical LHC analysis
  - ◆ Data access
  - ◆ Event selection
  - ◆ Histogramming
  - ◆ Statistical model building / fitting
  - ◆ Analysis preservation

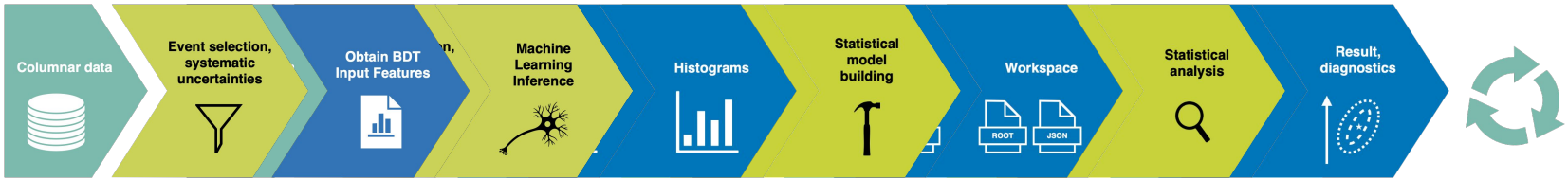


Machine Learning in HEP Analysis

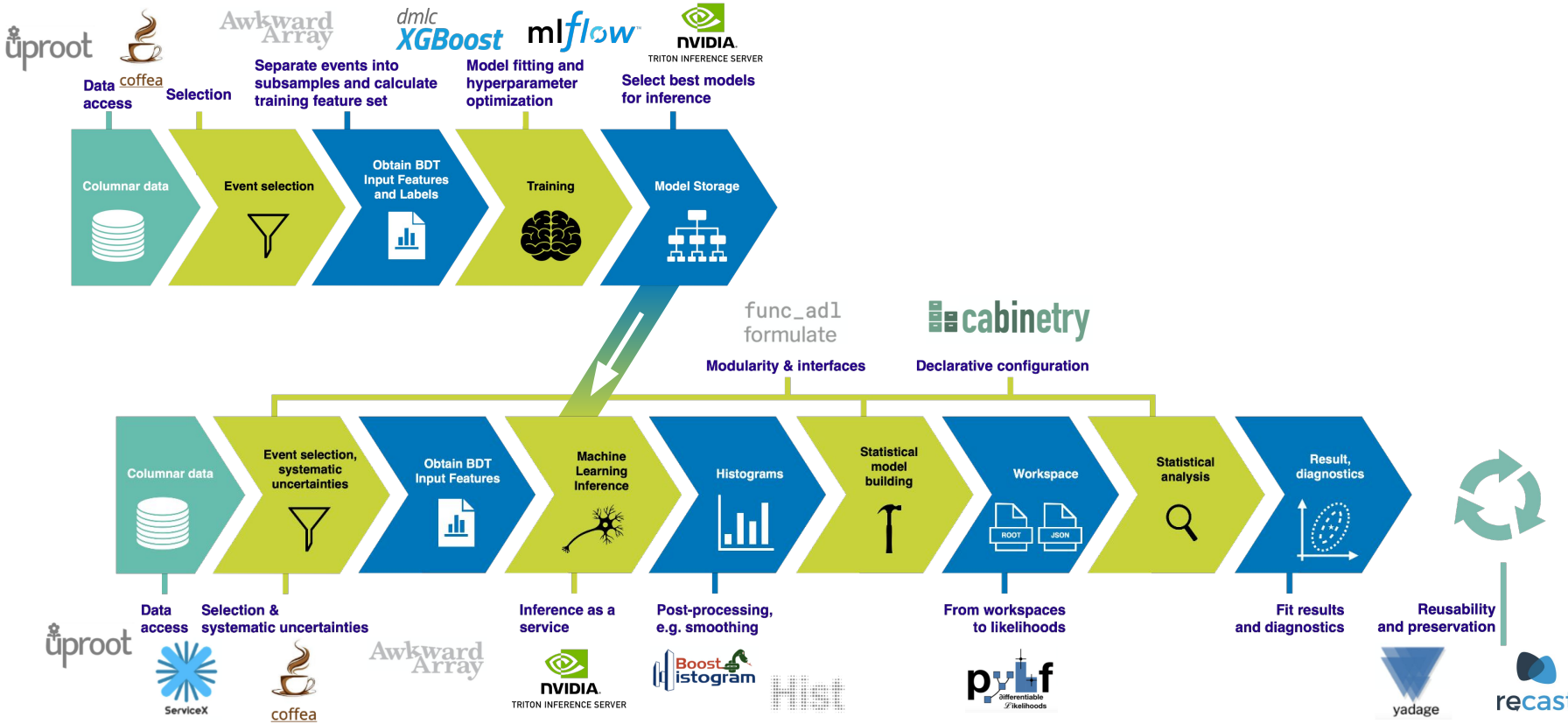
- What is needed for HL-LHC?
  - Improvement in **algorithm performance**
  - Better utilization of **computational resources**
- Machine learning (especially deep learning) can help with this!
- Add ML task to AGC pipeline to reflect developments in HEP analysis

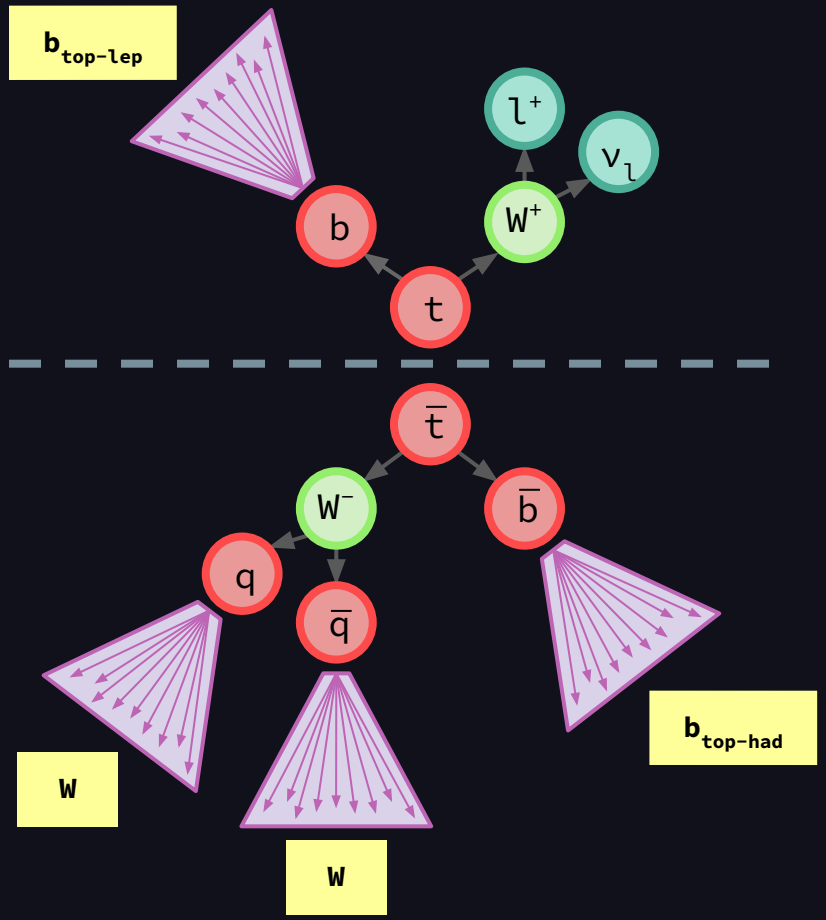


Analysis Grand Challenge + ML



Analysis Grand Challenge + ML



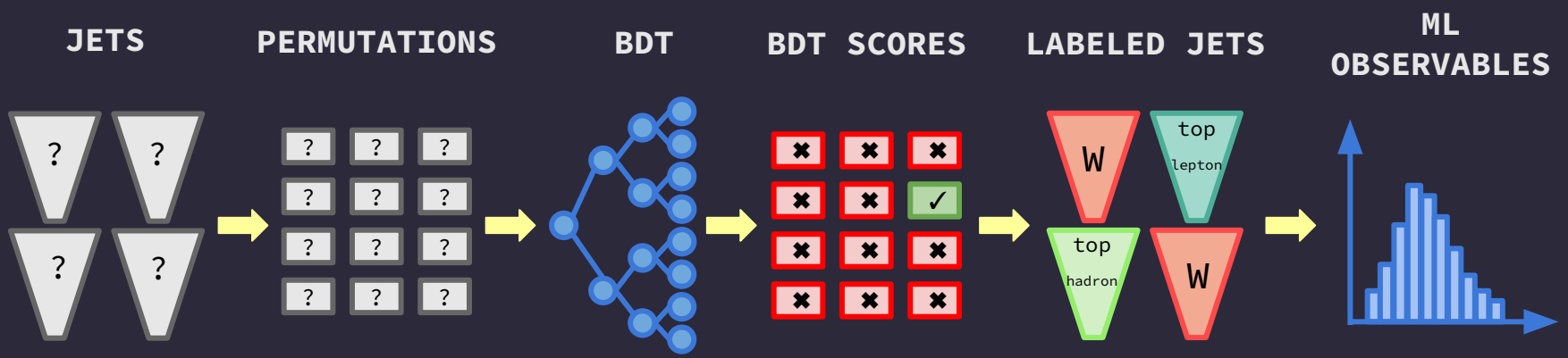


Machine Learning Task

- ➔ Assign jets to their parent partons
- ➔ Allows us to approximate observables such as
  - Mass of top quark (combined mass of $b_{top\text{-}had}$ and two W jets)
  - Angle between top_{lepton} jet and lepton ($\Delta\phi$)

Approach to ML Task

1. Consider **leading N jets** in each event
2. Find **all possible permutations** of parton assignments of these N jets (two W, one $b_{\text{top-had}}$, one $b_{\text{top-lep}}$)
3. **Calculate features** for each set of permutations and **feed into BDT**
4. Select permutation with **highest BDT score**
5. Use selection to **label jets**
6. Calculate **ML observables**



Training

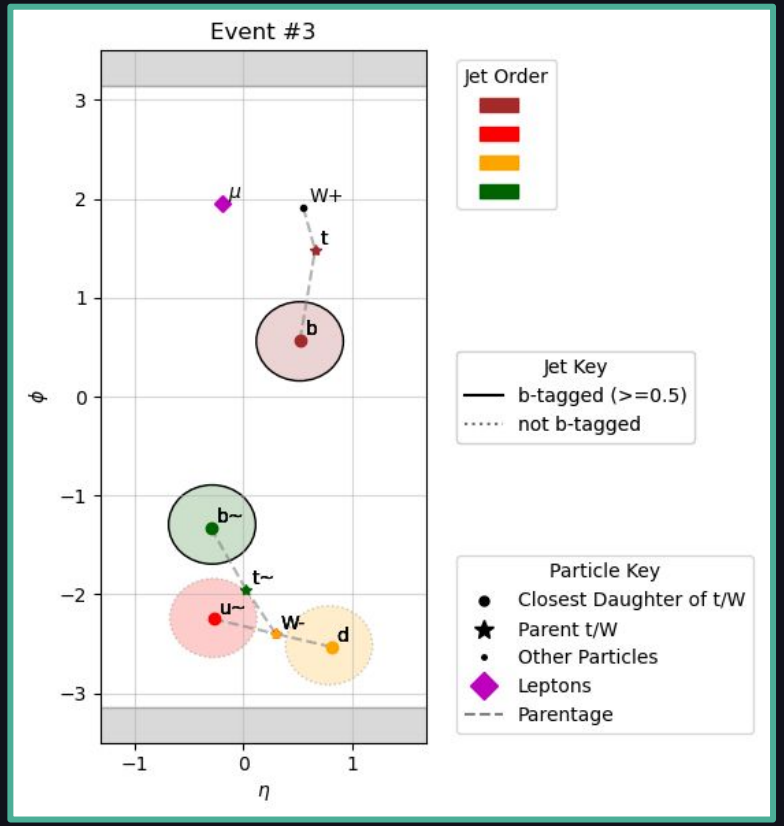
Columnar data
Event selection
Obtain Training Features
Training
Model Storage

Training: Event Selection

- Only look at ttbar MC events
- Match jets to particles using ΔR matching
- Find parent particle (either W or top quark)
- Keep events which are 100% reconstructable

```
nearest_genpart = jets.nearest(genparts, threshold=0.4)
nearest_parent = nearest_genpart.distinctParent  # parent of matched particle
parent_pdgid = nearest_parent.pdgId  # pdgId of parent particle
```

Example Event



Training: Obtaining Training Features

Python HEP libraries utilized:

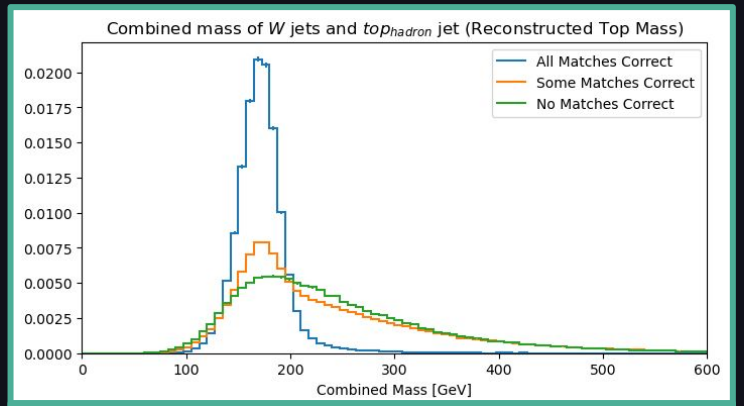
- uproot
- coffea
- awkward

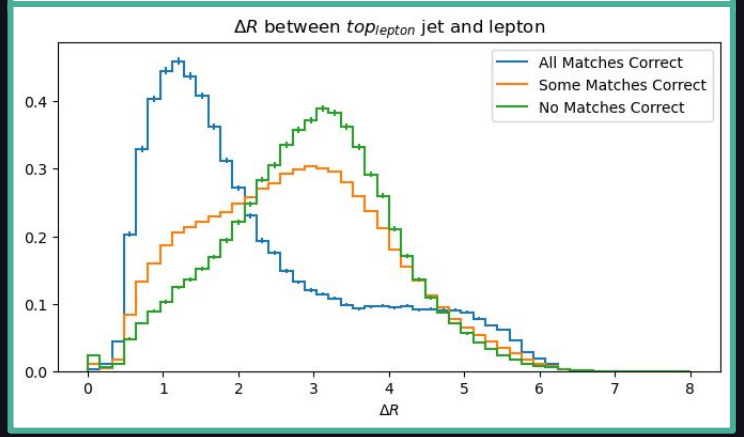
coffea processor with
coffea.nanoevents.NanoAODSchema

use awkward to calculate features

coffea.processor.column_accumulator objects

EXAMPLE FEATURES





Training: Hyperparameter Optimization

Set up mlflow on dask workers

```
client = utils.get_client()
client.run(initialize_mlflow)
```

Map jobs to workers (each job trains models for each hyperparameter selection)

```
futures = client.map(fit_model,
                     samples_even,
                     features=features_even,
                     labels=labels_even,
                     evaluation_matrix=evaluation_matrix,
                     n_folds=N_FOLD,
                     mlflowclient=mlflowclient,
                     use_mlflow=USE_MLFLOW,
                     log_models=MODEL_LOGGING)

res = client.gather(futures)
```

Number of times to split dataset for cross-validation

For each trial:

Track input hyperparameters

```
mlflowclient.log_param(run_id, param, np.average(value))
```

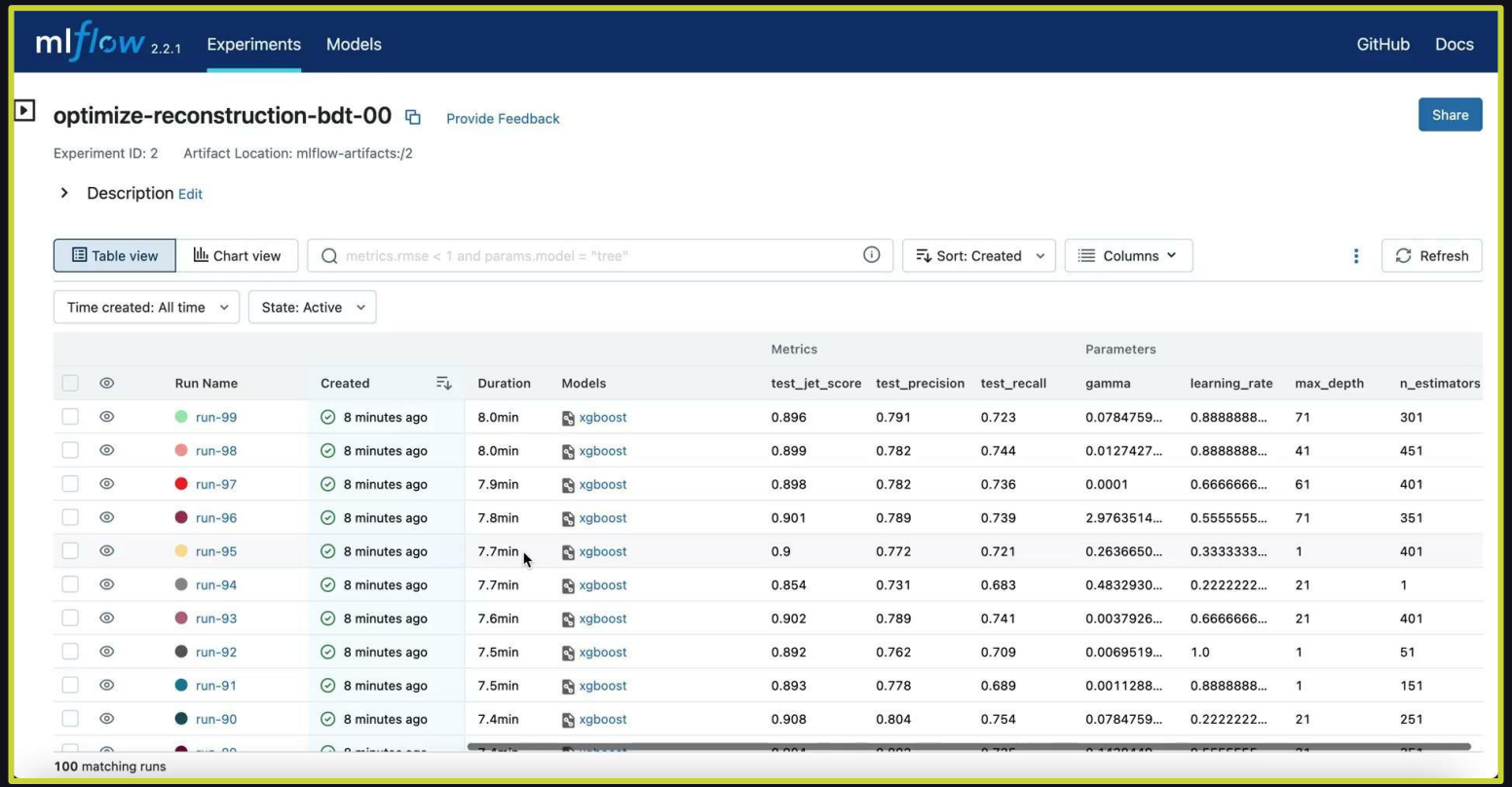
Track training scores

```
mlflowclient.log_metric(run_id, metric, np.average(value))
```

Log resulting model

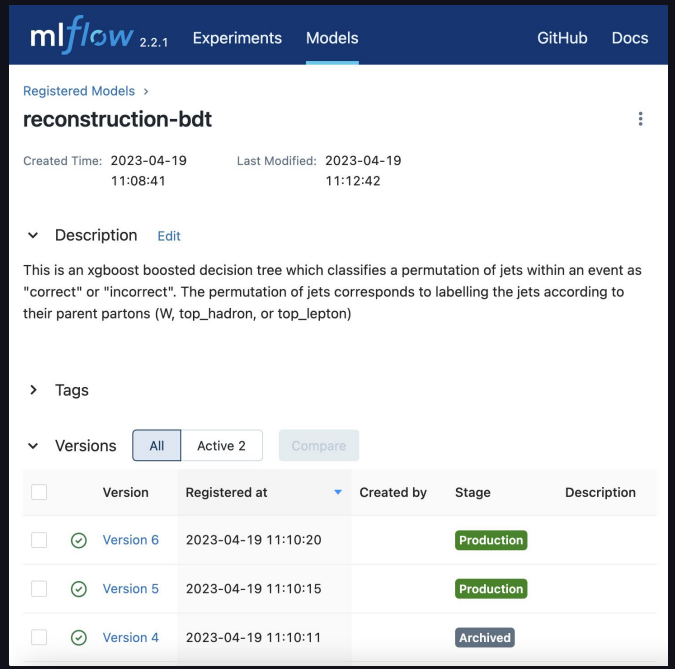
```
with mlflow.start_run(run_id=run_id, nested=True) as run:
    mlflow.xgboost.log_model(result["model"], "model", signature=signature)
```

Training: Hyperparameter Optimization



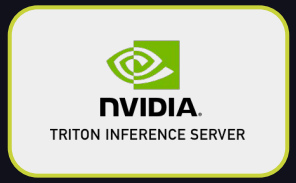
Training: Model Storage

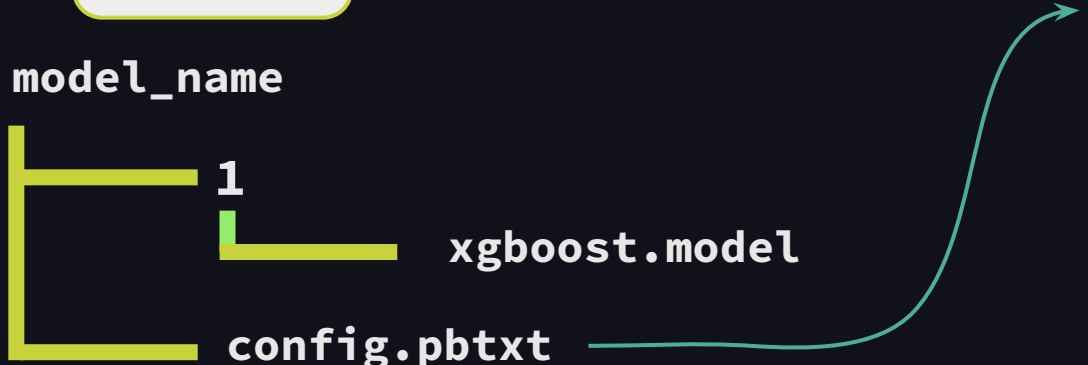
```
best_model_path = f'runs:/{best_run_id}/model'
best_model = mlflow.xgboost.load_model(best_model_path)  # load model locally
result = mlflow.register_model(best_model_path, "reconstruction-bdt")  # register best model in mlflow model repository
```



NVIDIA
TRITON INFERENCE SERVER

Training: Model Storage






```
name: "reconstruction_bdt_xgb"
backend: "fil"
max_batch_size: 50000000
input [
 {
    name: "input__0"
    data_type: TYPE_FP32
    dims: [ 20 ]
  }
]
output [
 {
    name: "output__0"
    data_type: TYPE_FP32
    dims: [ 2 ]
  }
]
instance_group [{ kind: KIND_GPU }]
parameters [
  {
    key: "model_type"
    value: { string_value: "xgboost" }
  },
  {
    key: "predict_proba"
    value: { string_value: "true" }
  },
]
version_policy: { all { }}
dynamic_batching { }
```

Inference

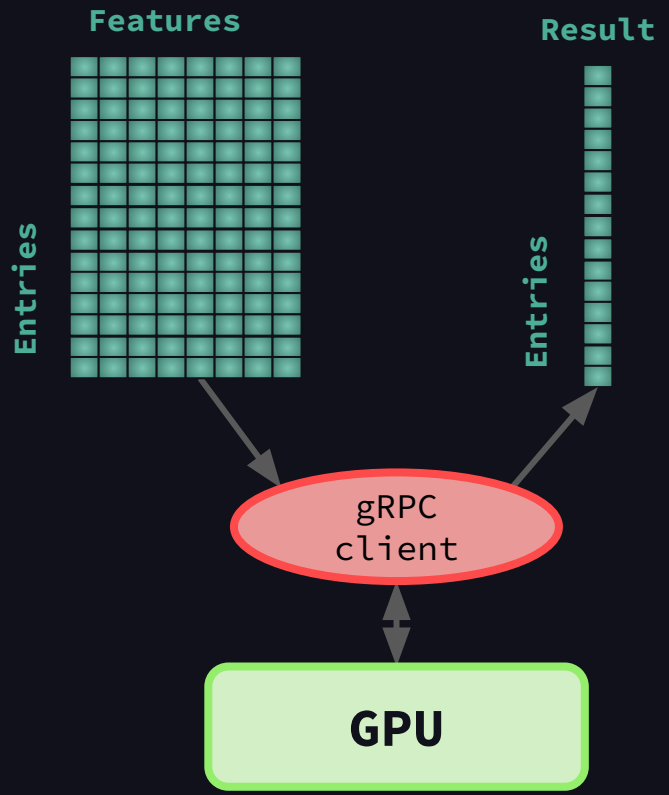
Columnar data
Event selection, systematic uncertainties
Obtain BDT Input Features
Machine Learning Inference
Histograms
Statistical model building
Workspace
ROOT
JSON
Statistical analysis
Result, diagnostics

Inference: Machine Learning Inference

NVIDIA TRITON INFERENCE SERVER

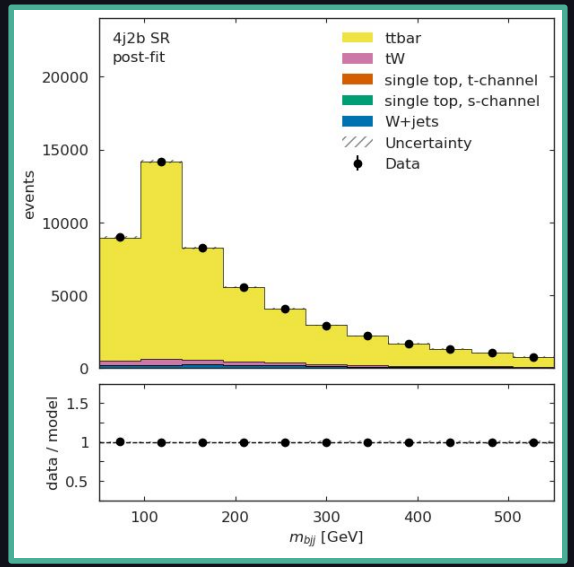
1. Set up Triton gRPC client
2. Perform inference request:

```
output = grpcclient.InferRequestedOutput(output_name)

inpt = [grpcclient.InferInput(input_name, features.shape, dtype)]
inpt[0].set_data_from_numpy(features)

results[even_perm]=triton_client.infer(
    model_name=self.model_name,
    model_version=self.model_vers_odd,
    inputs=inpt,
    outputs=[output]
).as_numpy(output_name)[:, 1]
```

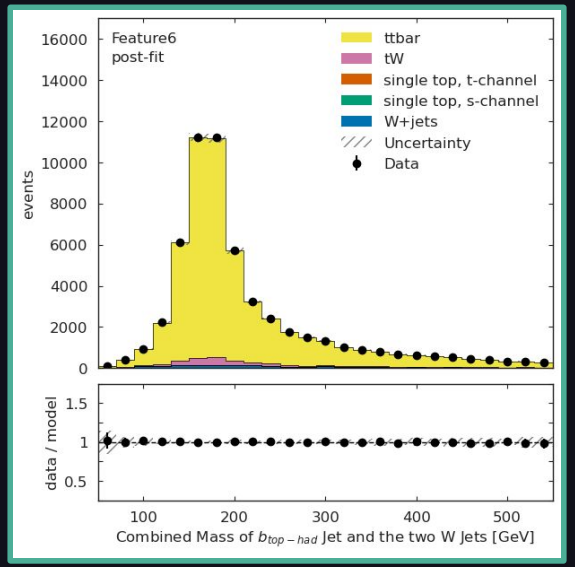


Inference: Result, diagnostics

Statistical model fit from non-ML approach



Apply fit results to model with ML observables



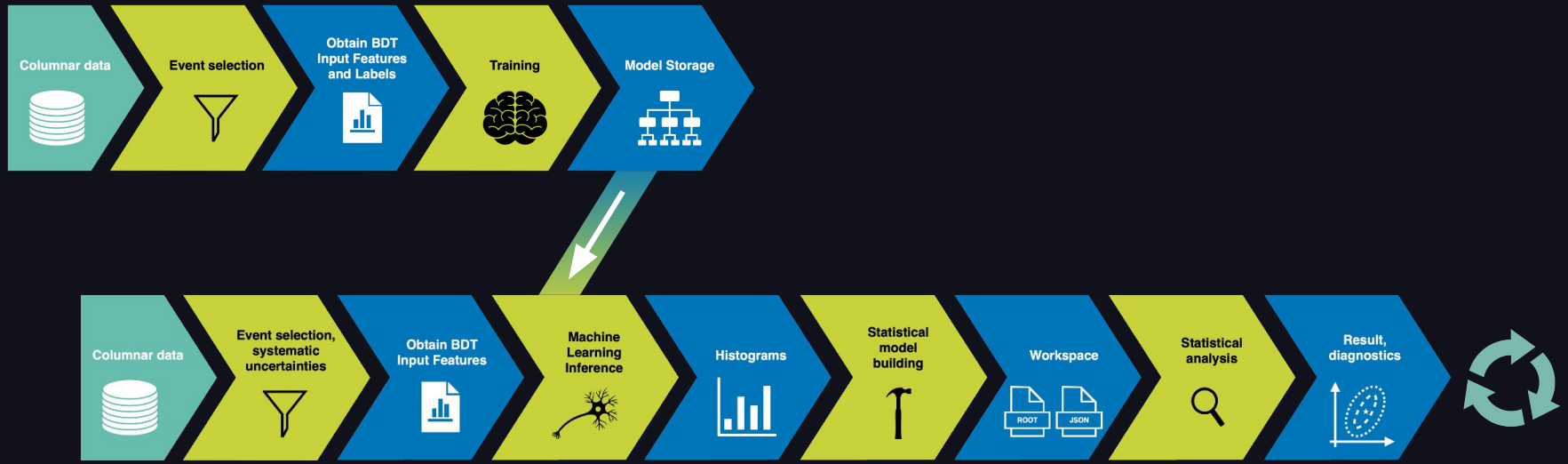
Python HEP libraries utilized:

- cabinetry
- hist

```
model_prediction = cabinetry.model_utils.prediction(model_ml)
fit_results_mod = cabinetry.model_utils.match_fit_results(model_ml, fit_results)
model_prediction_postfit = cabinetry.model_utils.prediction(model_ml, fit_results=fit_results_mod)
```

Future Goals

- ➔ Explore more complex ML models
  - ◆ Now that we have the infrastructure in place, we can extend to deep learning approaches
  - ◆ Triton inference server will help a lot here!
- ➔ Benchmarking



AGC Documentation

Try it yourself!

- Coffea-casa @ UNL
- Elastic Analysis Facility @ Fermilab
- Local Setup

Learn more about AGC!