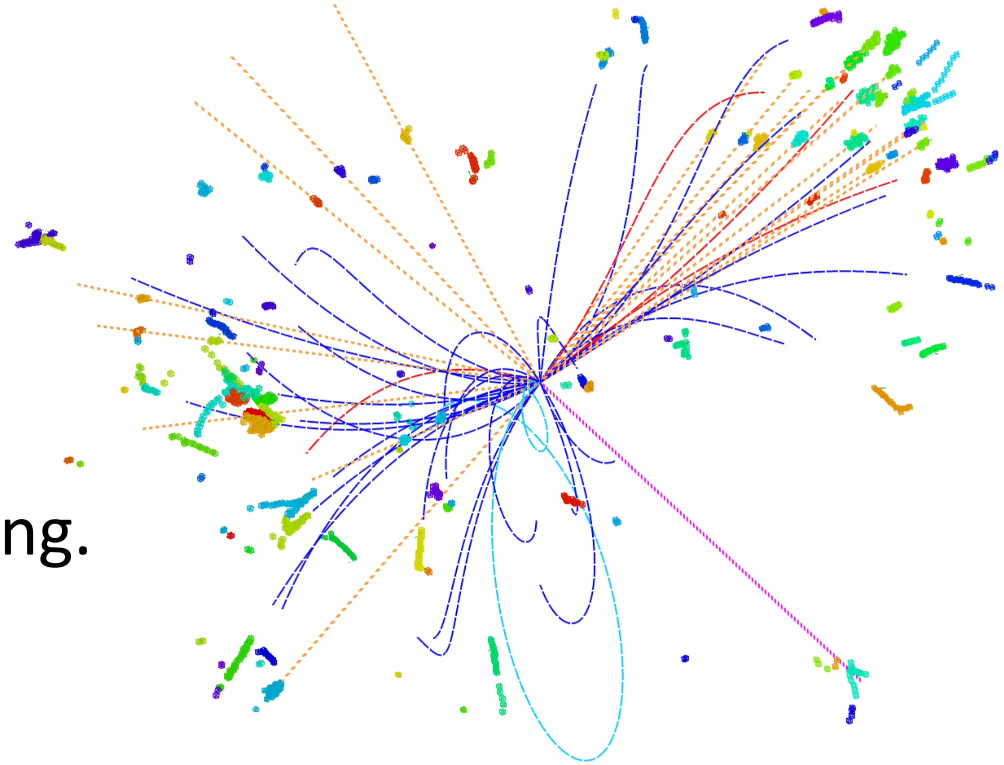
CEPC Jet Tutorial

Kaili Zhang

zhangkl@ihep.ac.cn

Updated in 2025.03.04

Jets 喷注



- Including varied components
- CEPC uses FastJet ee-kt/Durham to do jet clustering.
  - You need and only need to specify N_jets for Fastjet.
  - Generally, for all kt algos, 2 parameter: R and P can be adjusted.
  - ee-kt no R and P setting.
  - Inclusive clustering -> Everything should be in jet.

$$d_{ij} = \min(E_i^{2p}, E_j^{2p}) \frac{(1 - \cos\theta_{ij})}{(1 - \cos R)},$$

$$d_{iB} = E_i^{2p},$$

Jet @ CEPCSW

```
JetDefinition jet_def(ee_kt_algorithm);

ClusterSequence clust_seq(input_particles, jet_def);

vector<PseudoJet> jets = sorted_by_pt(clust_seq.exclusive_jets(nJets));
```

- Specify PFO container and njets.
- Not stored in final ntuple.
  - https://code.ihep.ac.cn/cepc/CEPCSW/-/tree/master/Analysis/GenMatch?ref_type=heads
  - Many variable stored.
  - Find jet_ntuple to extract information.

```
StatusCode GenMatch::initialize(){

  // Create a new TTree
  _file = TFile::Open(m_outputFile.value().c_str(), "RECREATE");
  _tree = new TTree("jets", "jets");
  _tree->Branch("jet1_px", &jet1_px, "jet1_px/D");
  _tree->Branch("jet1_py", &jet1_py, "jet1_py/D");
  _tree->Branch("jet1_pz", &jet1_pz, "jet1_pz/D");
  _tree->Branch("jet1_E", &jet1_E, "jet1_E/D");
  _tree->Branch("jet1_costheta", &jet1_costheta, "jet1_costheta/D");
  _tree->Branch("jet1_phi", &jet1_phi, "jet1_phi/D");
  _tree->Branch("jet1_pt", &jet1_pt, "jet1_pt/D");
  _tree->Branch("jet1_nconstituents", &jet1_nconstituents, "jet1_nconstituents/I");
  _tree->Branch("jet2_px", &jet2_px, "jet2_px/D");
  _tree->Branch("jet2_py", &jet2_py, "jet2_py/D");
  _tree->Branch("jet2_pz", &jet2_pz, "jet2_pz/D");
  _tree->Branch("jet2_E", &jet2_E, "jet2_E/D");
  _tree->Branch("jet2_costheta", &jet2_costheta, "jet2_costheta/D");
  _tree->Branch("jet2_phi", &jet2_phi, "jet2_phi/D");
  _tree->Branch("jet2_pt", &jet2_pt, "jet2_pt/D");
  _tree->Branch("jet2_nconstituents", &jet2_nconstituents, "jet2_nconstituents/I");
  _tree->Branch("constituents_E1tot", &constituents_E1tot, "constituents_E1tot/D");
  _tree->Branch("constituents_E2tot", &constituents_E2tot, "constituents_E2tot/D");
  _tree->Branch("mass", &mass, "mass/D");
  _tree->Branch("ymerge", ymerge, "ymerge[6]/D");
  _tree->Branch("nparticles", &nparticles, "nparticles/I");
  _tree->Branch("jet1_GENMatch_id", &jet1_GENMatch_id, "jet1_GENMatch_id/I");
  _tree->Branch("jet2_GENMatch_id", &jet2_GENMatch_id, "jet2_GENMatch_id/I");
  _tree->Branch("jet1_GENMatch_mindR", &jet1_GENMatch_mindR, "jet1_GENMatch_mindR/D");
  _tree->Branch("jet2_GENMatch_mindR", &jet2_GENMatch_mindR, "jet2_GENMatch_mindR/D");

  _tree->Branch("PFO_Energy_muon", &PFO_Energy_muon);
  _tree->Branch("PFO_Energy_muon_GENMatch_dR", &PFO_Energy_muon_GENMatch_dR);
  _tree->Branch("PFO_Energy_muon_GENMatch_ID", &PFO_Energy_muon_GENMatch_ID);
  _tree->Branch("PFO_Energy_muon_GENMatch_E", &PFO_Energy_muon_GENMatch_E);
  _tree->Branch("PFO_Energy_Charge", &PFO_Energy_Charge);
  _tree->Branch("PFO_Energy_Charge_Ecal", &PFO_Energy_Charge_Ecal);
  _tree->Branch("PFO_Energy_Charge_Hcal", &PFO_Energy_Charge_Hcal);
  _tree->Branch("PFO_Energy_Charge_GENMatch_dR", &PFO_Energy_Charge_GENMatch_dR);
  _tree->Branch("PFO_Energy_Charge_GENMatch_ID", &PFO_Energy_Charge_GENMatch_ID);
  _tree->Branch("PFO_Energy_Charge_GENMatch_E", &PFO_Energy_Charge_GENMatch_E);
  _tree->Branch("PFO_Hits_Charge_E", &PFO_Hits_Charge_E);
  _tree->Branch("PFO_Hits_Charge_R", &PFO_Hits_Charge_R);
  _tree->Branch("PFO_Hits_Charge_theta", &PFO_Hits_Charge_theta);
  _tree->Branch("PFO_Hits_Charge_phi", &PFO_Hits_Charge_phi);

  _tree->Branch("PFO_Energy_Neutral", &PFO_Energy_Neutral);
  _tree->Branch("PFO_Energy_Neutral_singleCluster", &PFO_Energy_Neutral_singleCluster);
  _tree->Branch("PFO_Energy_Neutral_singleCluster_R", &PFO_Energy_Neutral_singleCluster_R);
  _tree->Branch("PFO_Hits_Neutral_E", &PFO_Hits_Neutral_E);
  _tree->Branch("PFO_Hits_Neutral_R", &PFO_Hits_Neutral_R);
  _tree->Branch("PFO_Hits_Neutral_theta", &PFO_Hits_Neutral_theta);
  _tree->Branch("PFO_Hits_Neutral_phi", &PFO_Hits_Neutral_phi);
```

Jet Study example

/cefs/higgs/zhangkl/CEPCSW/Analysis/JetOrigin/src

- Reco/Truth Jet match
- PFO reco PID
- PFO reco/truth matching via dE, dR.
- PFO reco/truth matching via simHit
- Track D0/Z0, Cluster position
- Prepare JOI inputs
- ……

Flavor Tagging

- Traditional: LCFIPlus BDT based.
- P-CNN https://scipost.org/10.21468/SciPostPhys.7.1.014
- Particle Flow Network
  - https://arxiv.org/abs/1810.05165
  - CEPC@Xiaotian : https://arxiv.org/abs/2410.04465v2
- LundNet https://doi.org/10.1007/jhep03(2021)052
- ParticleNet
  - Arxiv:1902.08570 https://github.com/hqucms/ParticleNet

ParticleTransformer

- https://arxiv.org/abs/2202.03772
- https://github.com/jet-universe/particle_transformer
- Platforms: https://github.com/hqucms/weaver-core
- Application on CEPC: 2309.13231 , PRL 132, 221802 (2024)
- Tutorial on CEPC: https://github.com/ZHUYFgit/CEPC-Jet-Origin-Identification
- Inputs from CEPCsoft: /cefs/higgs/zhangkl/AI/datasets
- Inputs from LHC, JetClass: /cefs/higgs/zhangkl/AI/jetclass
- Require higgsgpu group. Request on https://ccsinfo.ihep.ac.cn/
- Follow the tutorial, build the env if you are interested.

ParticleTransformer @ CEPC

https://github.com/ZHUYFgit/CEPC-Jet-Origin-Identification

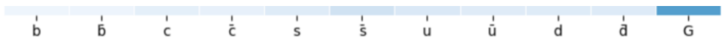
- Variable list in JetClass_full.yaml
  - Under development to CEPCSW
  - Unit as one jet: 4 momentum, M11 id information……
- Train in Weaver:
  - Submit jobs on IHEP: train_JetClass.sh
  - Output: Pred.root: Label and score for each jets.
  - Application: pt model or onnx format
- Alternative: Pytorch + Transformer. @Zuofei

Inputs for JOI

/cefs/higgs/zhangkl/CEPCSW/Analysis/JetOrigin/src

Input ntuple

- Jet->Event;
- PFO->Component;
- Length: ~50
- Label: M11



- Current training use truth PID information.

  Reco ID is ready to replace.

Type	Var	Comment
PFO point distance	$\Delta\phi(pfo, Jet)$	Delta Phi, pfo to jet
	$\Delta\eta(pfo, Jet)$	Delta Eta, pfo to jet
PFO Vector variable	(px, py, pz, E)	4 momentum of PFO
PFO feature variable	$\mathrm{P}_t^{PFO}, \log\frac{P_t^{PFO}}{P_t^{jet}}$	Pfo pt and relative pt
	$E_t^{PFO}, \log\frac{E_t^{PFO}}{E_t^{jet}}$	Pfo E and relative E
	$\Delta R(pfo, Jet)$	Delta R, pfo to jet
	D0, Z0, D0err, Z0err	(if with track) impact parameters
	PID	Truth PID type

While new inputs like vertex can help,
Introducing new variables can reduce the performance.
Weaver has strict range conventions. See guide.

Slurm Script

Higgsgpu group: Sbatch x.sh
http://afsapply.ihep.ac.cn/cchelp/en/local-cluster/jobs/slurm/

```
#!/bin/bash
###### Part 1 ######
#SBATCH --partition=gpu
#SBATCH --qos=normal
#SBATCH --account=higgsgpu
#SBATCH --job-name=0303
#SBATCH --ntasks=8
#SBATCH --output=logs/0303.log
#SBATCH --mem-per-cpu=32768
#SBATCH --gres=gpu:v100:4

###### Part 2 ######
```

- Weaver config: if you are not sure, keep it.
- Source you anaconda
- --num-workers 5
- --batch-size 512 –start-lr 1e-3
- Samples-per-epoch(-val): consistent with your inputs.
- --optimizer ranger

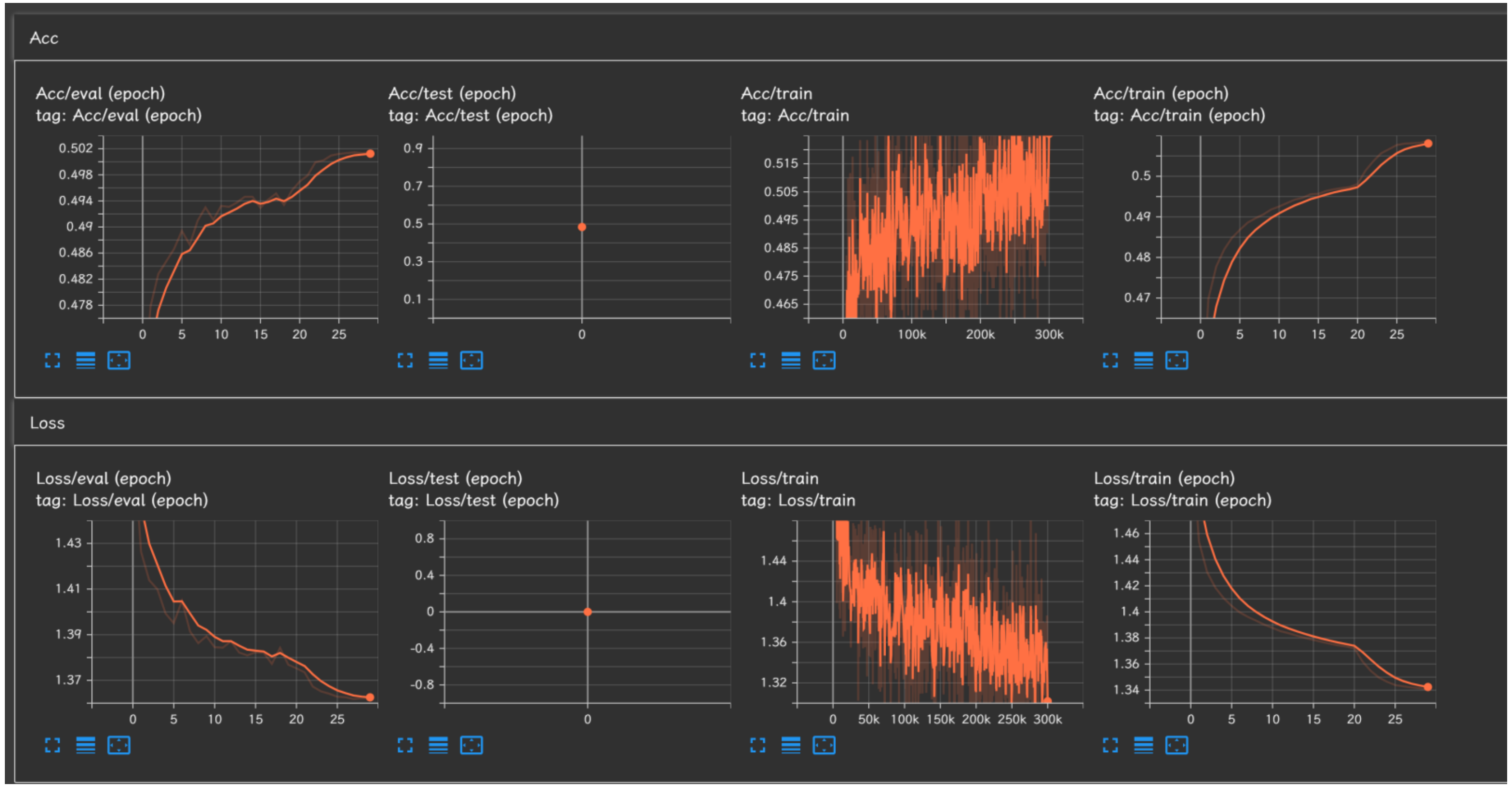
Monitoring

Monitoring training process with tensorboard

```
conda activate weaver
cd runs
tensorboard --logdir . --bind_all
# wait for two or three miniutes

# if the port is used by the other program, you need to specify a port manualy
tensorboard --logdir . --bind_all --port 8888

```



Output

- Best model are stored in

  net_best_epoch_state.pt

- Option: use pytorch to read

- Ntuple: Pred*.root.
  - Test sample with score

- Logs to debug:

- /best: epoch performance

- /entries: epoch sample entry label

- /metric train/test/validation

Tagging Score

- For single jet, use the max score for M11 to get best jet likelihood;
- For flavor tagging, with certain scenario:
- Define tagging eff, mis tagging rate, purity, ROC curve.

$$\text{B vs All} = \frac{prob(b)}{prob(b) + prob(c) + prob(udsg)}$$

$$\text{B vs L} = \frac{prob(b)}{prob(b) + prob(udsg)}$$

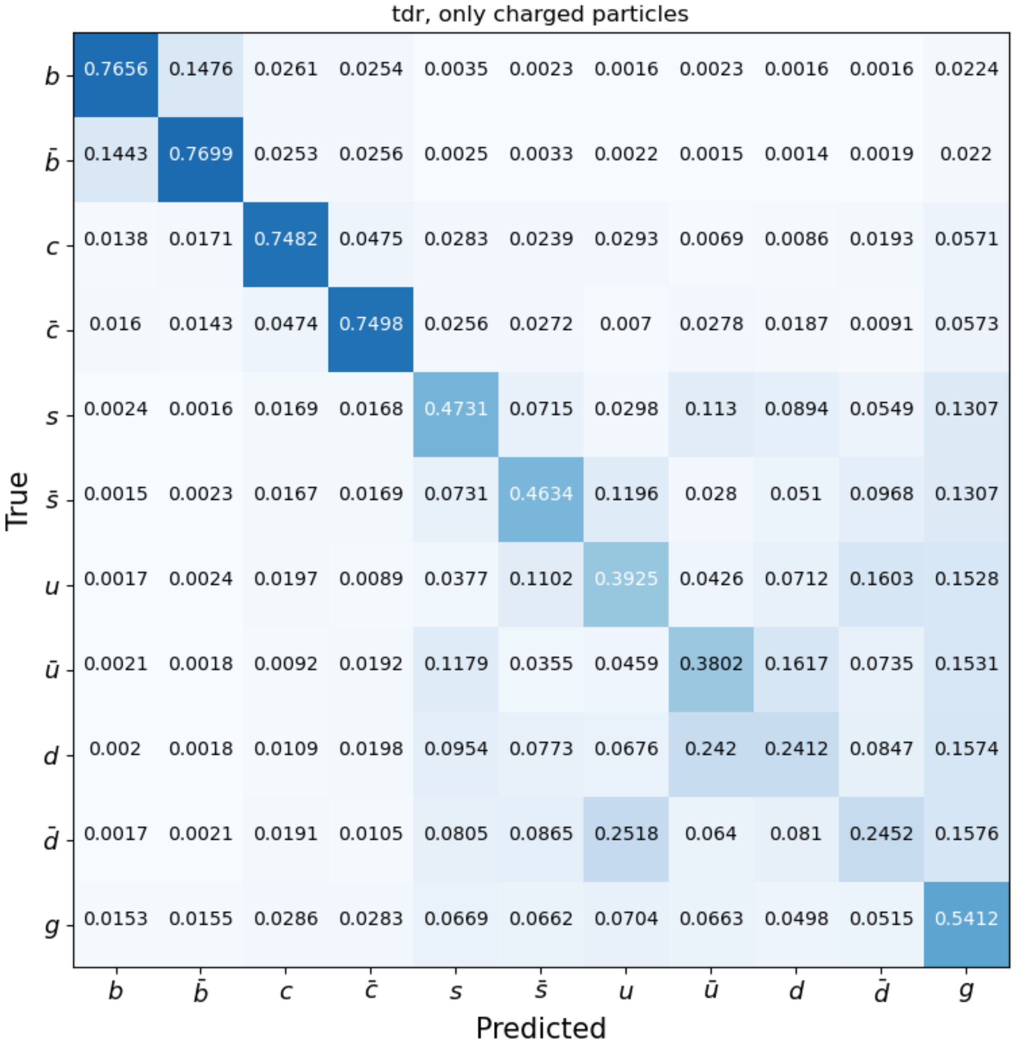
$$\text{B vs C} = \frac{prob(b)}{prob(b) + prob(c)}$$

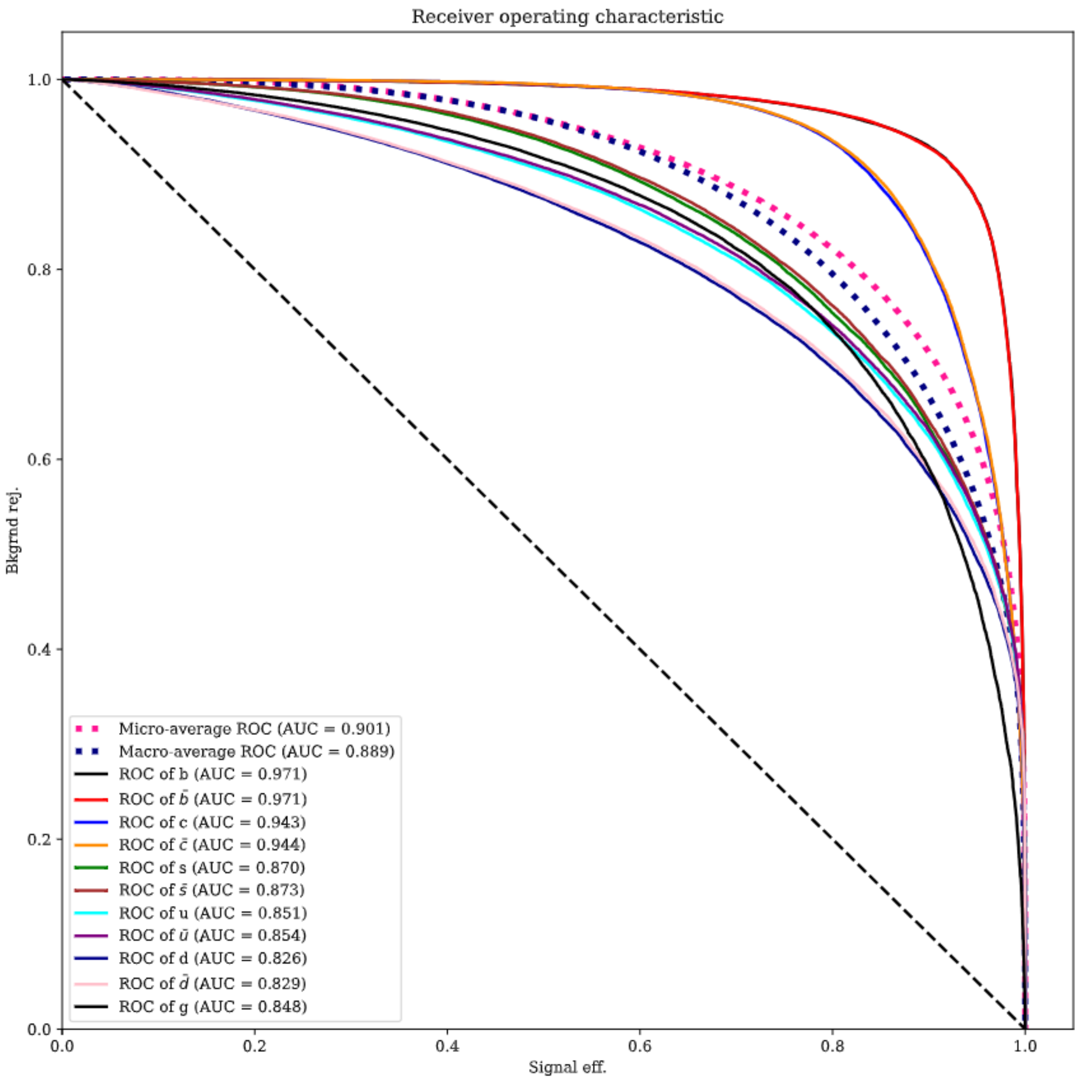
$$\text{B vs All weighted} = \frac{prob(b)}{k_c \cdot prob(c) + (1 - k_c) \cdot prob(udsg)}$$

Matrix/Curve

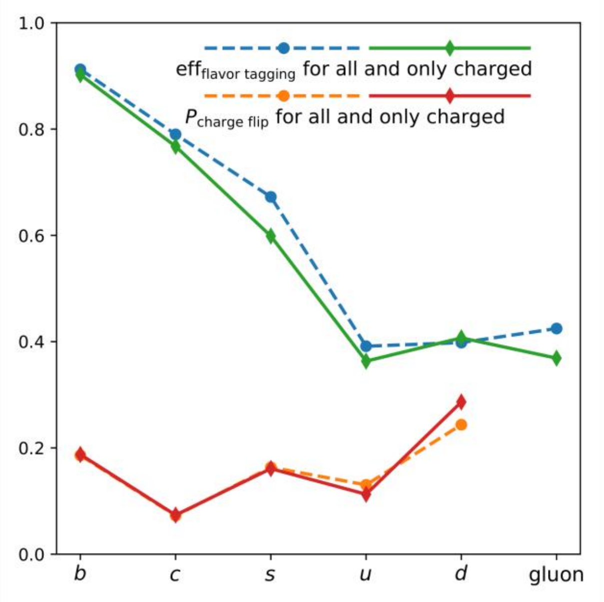
`sklearn.metrics.roc_curve()`

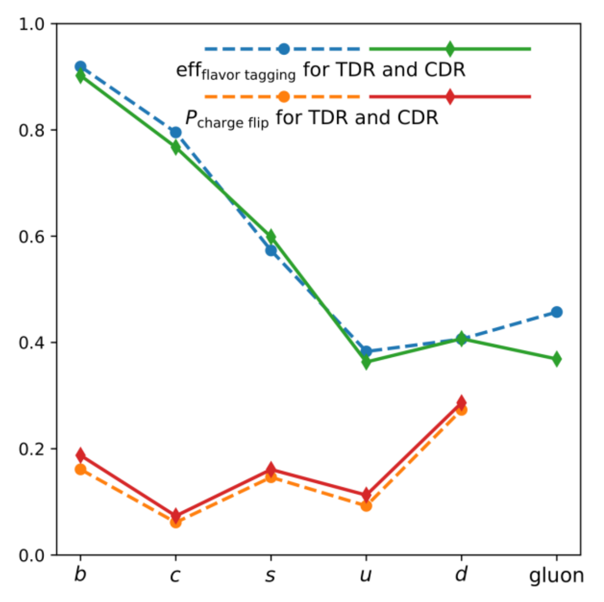
Current result are ready for TDR.

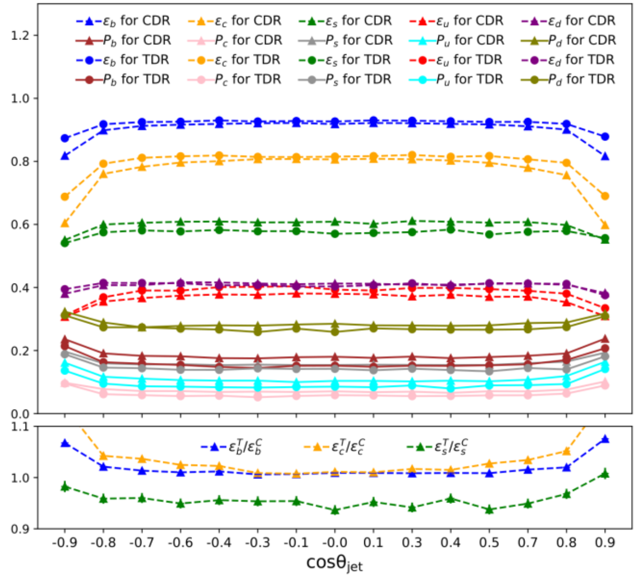




Tagging eff, Charge flip rate







Using test ntuples(assuming yields are uniform)
CDR flavor tagging eff use test set
max(score_b+score_bbar, score_c+score_cbar……)

Application: Weaver Predict

Prepare the inputs with same format as training.
Suggest with evt number in “observers” to match.

- Do not need GPU. Directly run weaver
- weaver --predict --data-test "nnHgg:${test_dir}/*.root" --data-config ${input_var} --network-config ${modelopts} --model-prefix ${trained_model} \
  ${GPU} ${batchopts} --log ${log_file} --predict-output ${output_file}
- To onnx: weaver --data-config ${input_var} --network-config ${modelopts} --model-prefix ${trained_model} --export-onnx ${output_onnx}
- CEPCSW implement: to do

Q&A