Phase 2 L1 Muon Performance

EVAN KOENIG

Phase 2 Performance Codebase

Main working repo: https://github.com/ekoenig4/P2L1TMuonVal

CMSSW 12_5_X

Goal: to provide a framework for easy and scalable scripts

Once added to CMSSW all the utilities can be imported into a python script with

```
from L1Trigger.Phase2L1GMTNtuples.tools import *
```

The framework is optimized for running with uproot and awkward arrays

- These packages allow for faster processing of TTrees than PyROOT

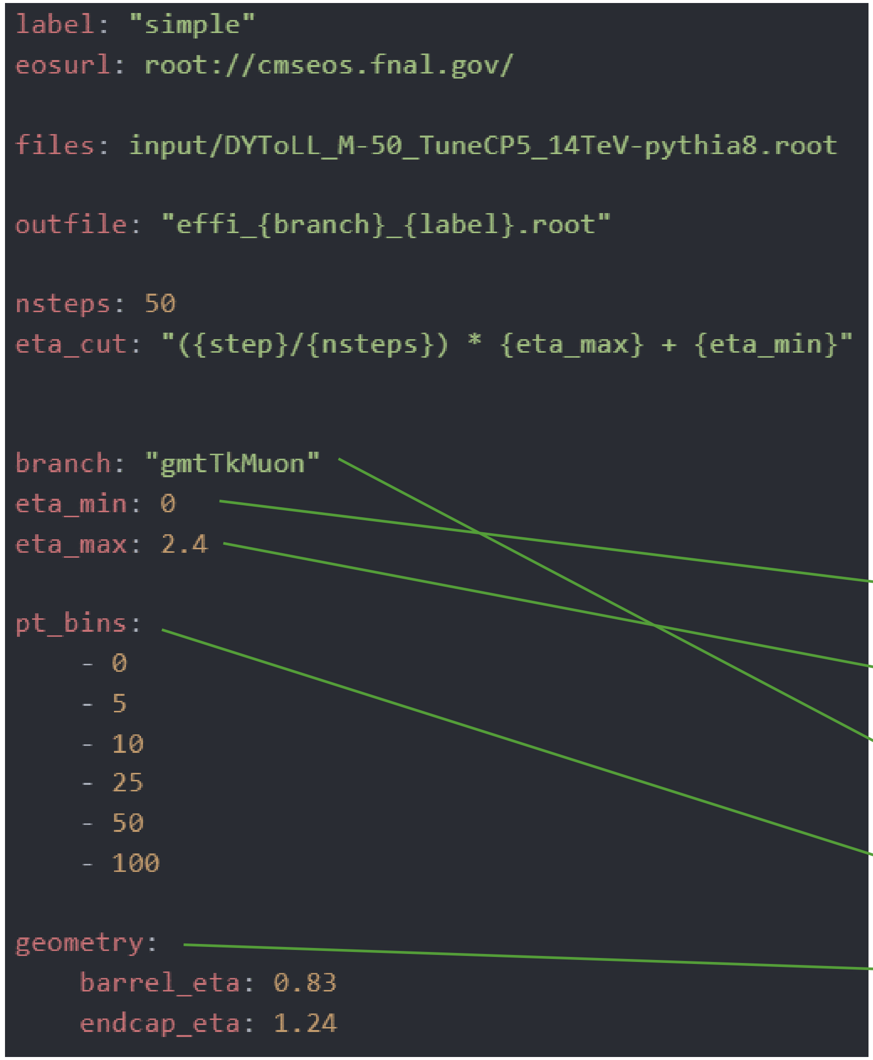
Existing scripts written using this framework can be found here

- effiMuonSimple.py: Calculates gen matching efficiencies for L1 Tracker and Standalone muons
- diMuonSimple.py: Calculates the di-muon mass distributions for both gen and L1 Tracker/Standalone muons

The utility of this framework will continue to expand as many more scripts get added to the repo

Making a script approachable and scalable

configs/simple_config.yaml



```yaml
label: "simple"
eosurl: root://cmseos.fnal.gov/

files: input/DYToLL_M-50_TuneCP5_14TeV-pythia8.root

outfile: "effi_{branch}_{label}.root"

nsteps: 50
eta_cut: "({step}/{nsteps}) * {eta_max} + {eta_min}"


branch: "gmtTkMuon"
eta_min: 0
eta_max: 2.4

pt_bins:
    - 0
    - 5
    - 10
    - 25
    - 50
    - 100

geometry:
    barrel_eta: 0.83
    endcap_eta: 1.24
```

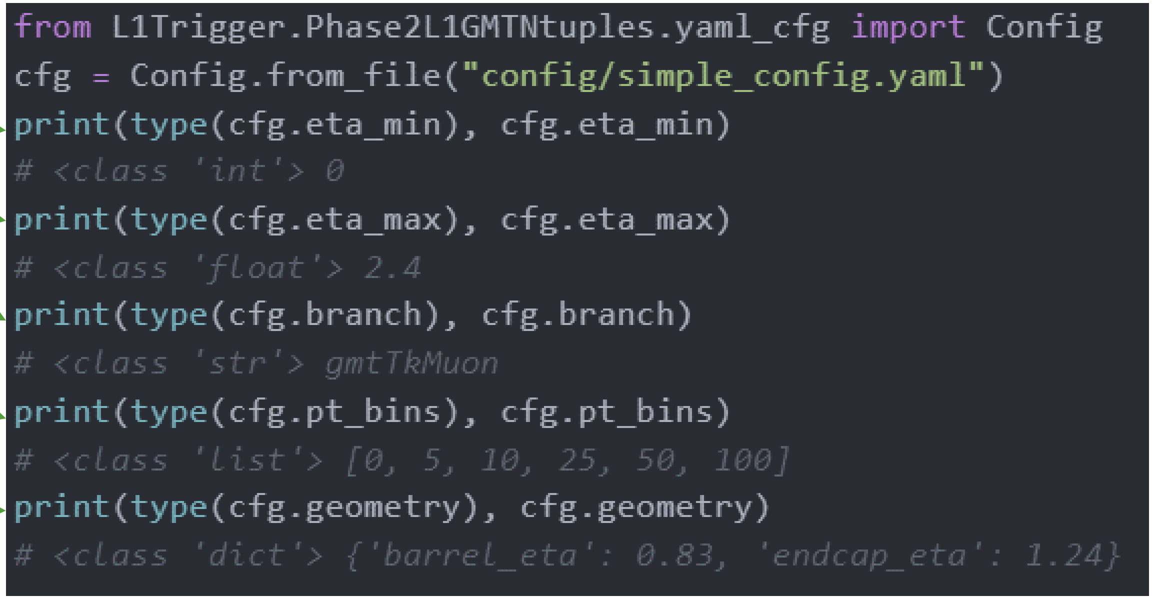
The yaml_cfg.py module provides an easy to implement config written using YAML

```python
from L1Trigger.Phase2L1GMTNtuples.yaml_cfg import Config
```

YAML is a type-based configuration format

The Config class loads in all defined variables with their corresponding types

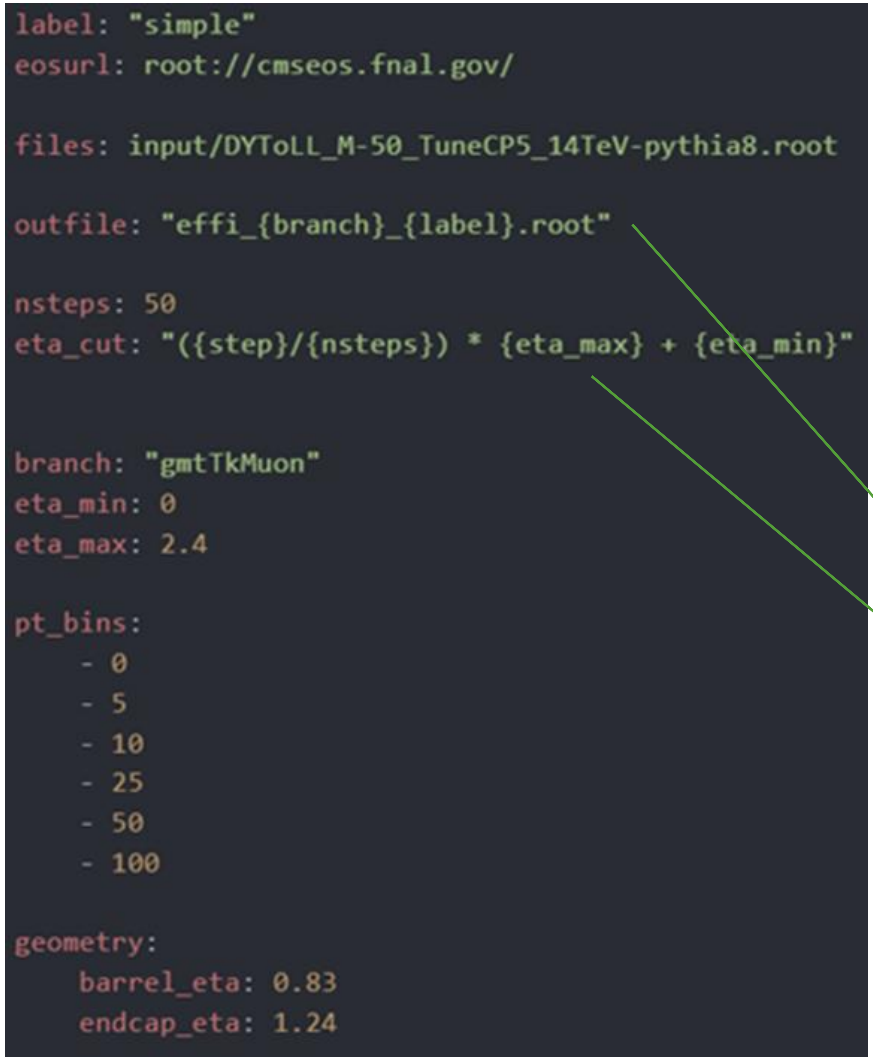
simple_script.py



```python
from L1Trigger.Phase2L1GMTNtuples.yaml_cfg import Config
cfg = Config.from_file("config/simple_config.yaml")
print(type(cfg.eta_min), cfg.eta_min)
# <class 'int'> 0
print(type(cfg.eta_max), cfg.eta_max)
# <class 'float'> 2.4
print(type(cfg.branch), cfg.branch)
# <class 'str'> gmtTkMuon
print(type(cfg.pt_bins), cfg.pt_bins)
# <class 'list'> [0, 5, 10, 25, 50, 100]
print(type(cfg.geometry), cfg.geometry)
# <class 'dict'> {'barrel_eta': 0.83, 'endcap_eta': 1.24}
```

Making a script approachable and scalable

configs/simple_config.yaml

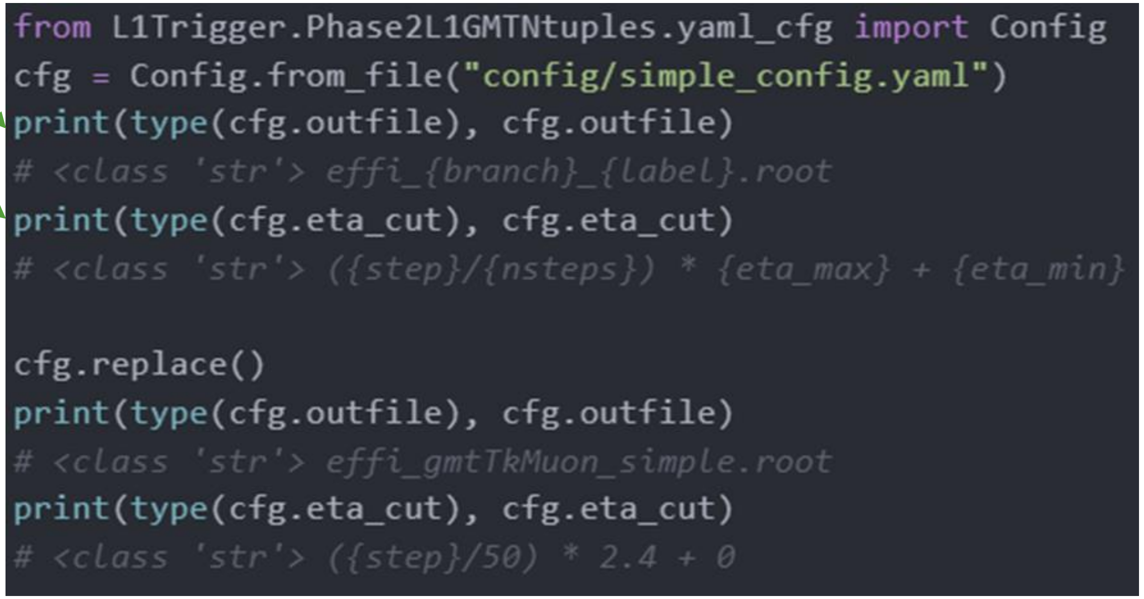


```yaml
label: "simple"
eosurl: root://cmseos.fnal.gov/

files: input/DYToLL_M-50_TuneCP5_14TeV-pythia8.root

outfile: "effi_{branch}_{label}.root"

nsteps: 50
eta_cut: "({step}/{nsteps}) * {eta_max} + {eta_min}"


branch: "gmtTkMuon"
eta_min: 0
eta_max: 2.4

pt_bins:
    - 0
    - 5
    - 10
    - 25
    - 50
    - 100

geometry:
    barrel_eta: 0.83
    endcap_eta: 1.24
```

Config also supports {} replace values when available

Will replace using any defined keys in current config

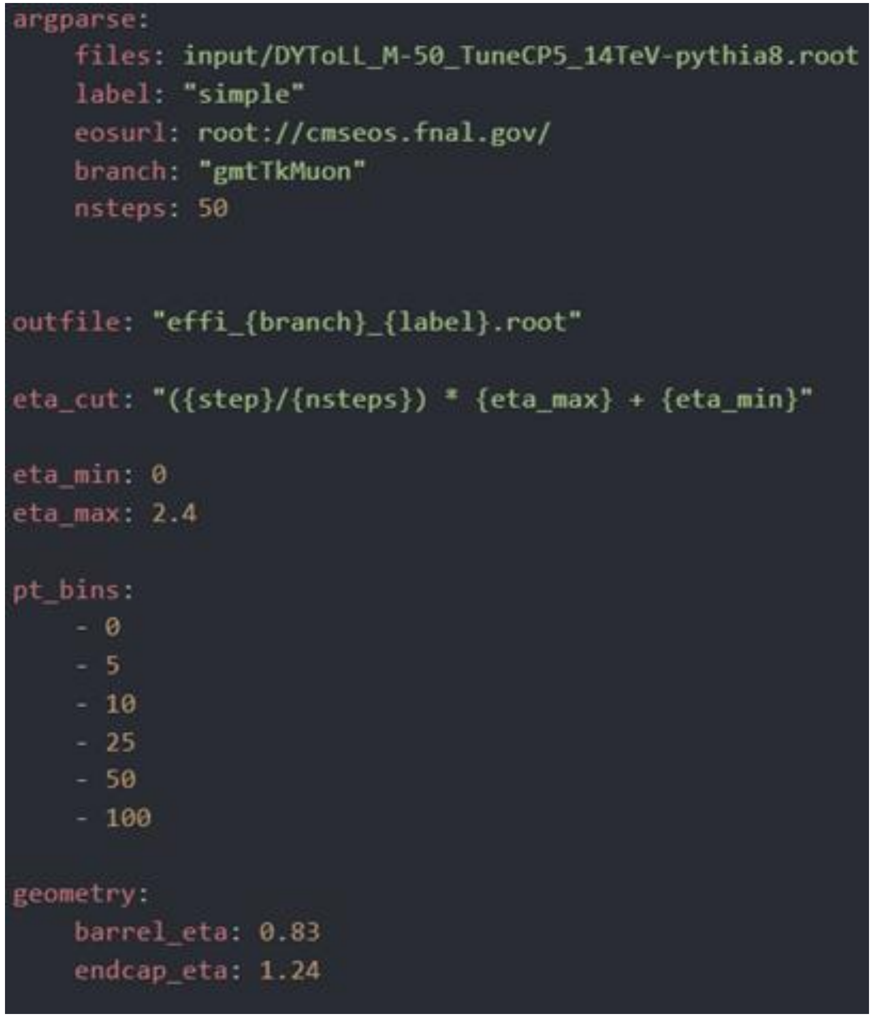
simple_script.py



```python
from L1Trigger.Phase2L1GMTNtuples.yaml_cfg import Config
cfg = Config.from_file("config/simple_config.yaml")
print(type(cfg.outfile), cfg.outfile)
# <class 'str'> effi_{branch}_{label}.root
print(type(cfg.eta_cut), cfg.eta_cut)
# <class 'str'> ({step}/{nsteps}) * {eta_max} + {eta_min}

cfg.replace()
print(type(cfg.outfile), cfg.outfile)
# <class 'str'> effi_gmtTkMuon_simple.root
print(type(cfg.eta_cut), cfg.eta_cut)
# <class 'str'> ({step}/50) * 2.4 + 0
```

Making a script approachable and scalable

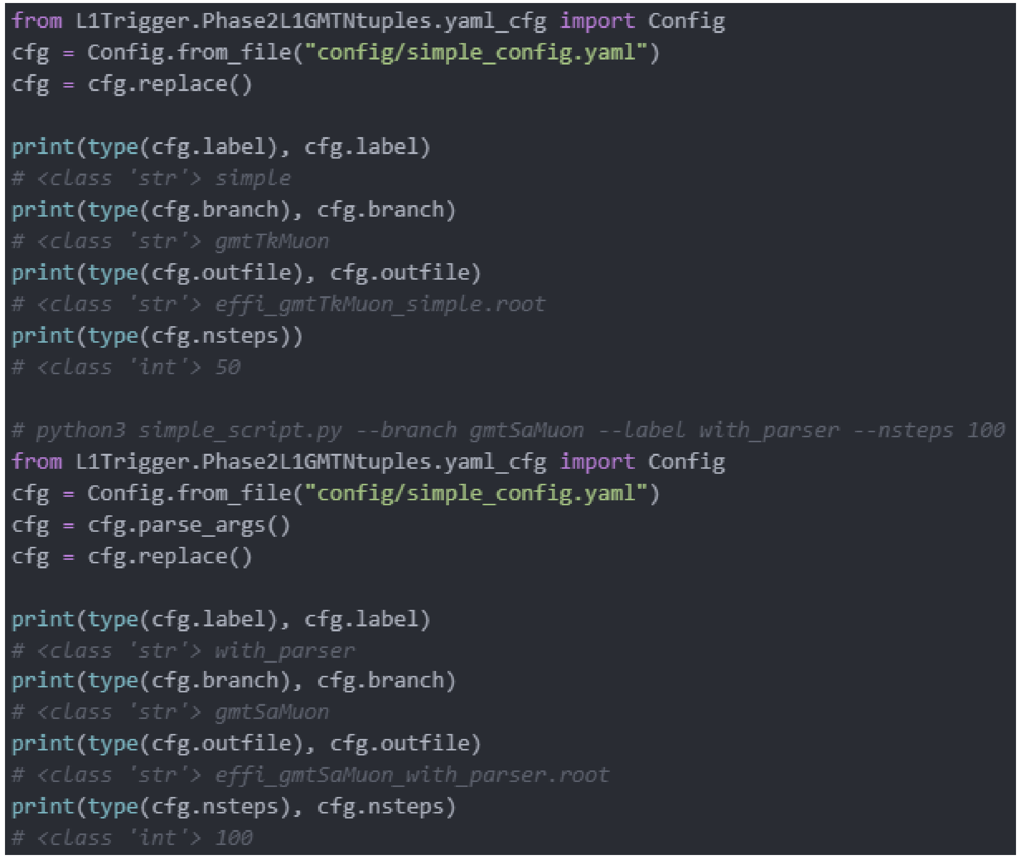
configs/simple_config.yaml



```yaml
argparse:
    files: input/DYToLL_M-50_TuneCP5_14TeV-pythia8.root
    label: "simple"
    eosurl: root://cmseos.fnal.gov/
    branch: "gmtTkMuon"
    nsteps: 50


outfile: "effi_{branch}_{label}.root"

eta_cut: "({step}/{nsteps}) * {eta_max} + {eta_min}"

eta_min: 0
eta_max: 2.4

pt_bins:
    - 0
    - 5
    - 10
    - 25
    - 50
    - 100

geometry:
    barrel_eta: 0.83
    endcap_eta: 1.24
```

Config supports easy command line argument parser

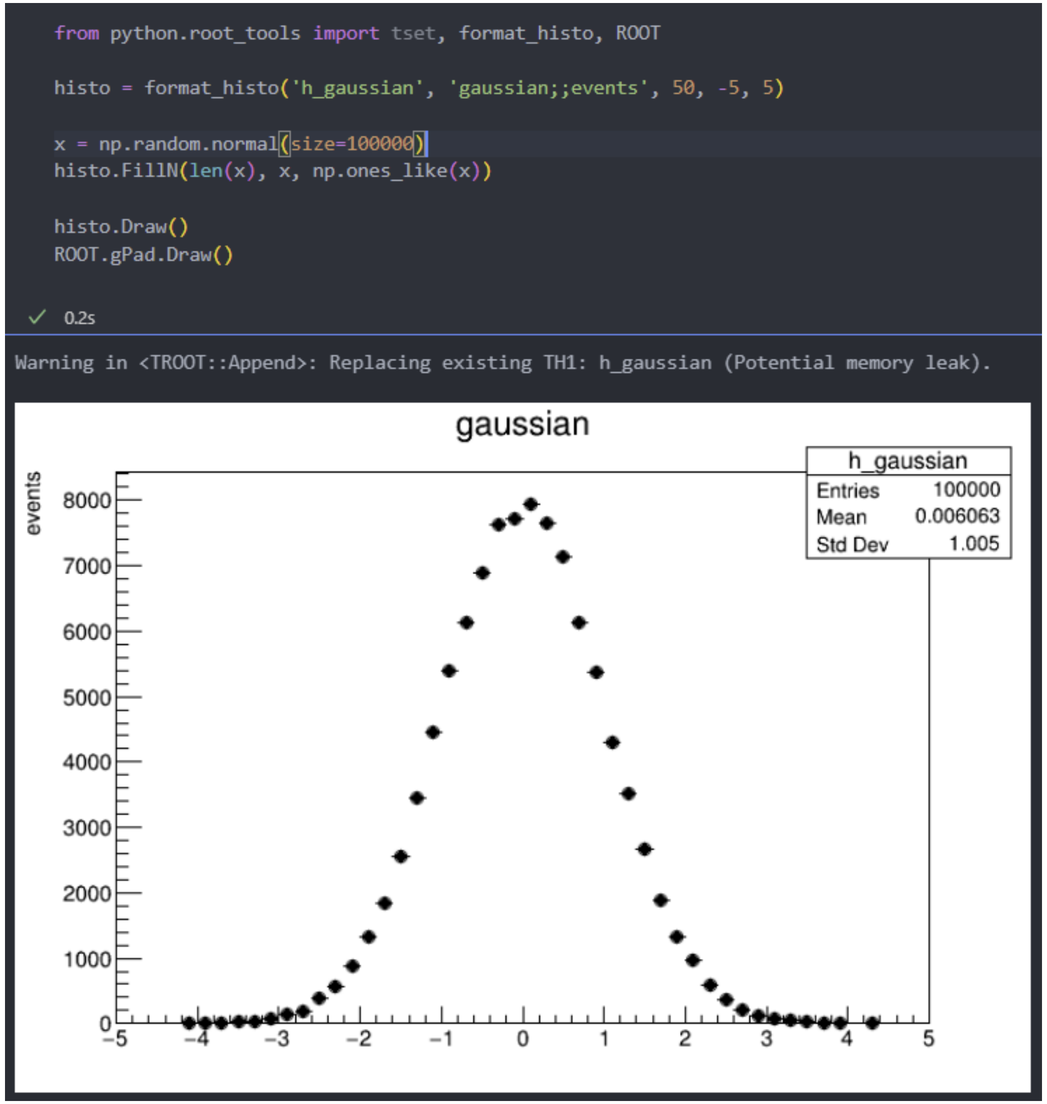
simple_script.py



```python
from L1Trigger.Phase2L1GMTNtuples.yaml_cfg import Config
cfg = Config.from_file("config/simple_config.yaml")
cfg = cfg.replace()

print(type(cfg.label), cfg.label)
# <class 'str'> simple
print(type(cfg.branch), cfg.branch)
# <class 'str'> gmtTkMuon
print(type(cfg.outfile), cfg.outfile)
# <class 'str'> effi_gmtTkMuon_simple.root
print(type(cfg.nsteps))
# <class 'int'> 50

# python3 simple_script.py --branch gmtSaMuon --label with_parser --nsteps 100
from L1Trigger.Phase2L1GMTNtuples.yaml_cfg import Config
cfg = Config.from_file("config/simple_config.yaml")
cfg = cfg.parse_args()
cfg = cfg.replace()

print(type(cfg.label), cfg.label)
# <class 'str'> with_parser
print(type(cfg.branch), cfg.branch)
# <class 'str'> gmtSaMuon
print(type(cfg.outfile), cfg.outfile)
# <class 'str'> effi_gmtSaMuon_with_parser.root
print(type(cfg.nsteps), cfg.nsteps)
# <class 'int'> 100
```

A more Pythonic ROOT

Along with Configs, the module root_tools.py provides a more pythonic interface with PyROOT



```
from python.root_tools import tset, format_histo, ROOT

histo = format_histo('h_gaussian', 'gaussian;;events', 50, -5, 5)

x = np.random.normal(size=100000)
histo.FillN(len(x), x, np.ones_like(x))

histo.Draw()
ROOT.gPad.Draw()
```

✓ 0.2s

```
Warning in <TROOT::Append>: Replacing existing TH1: h_gaussian (Potential memory leak).
```

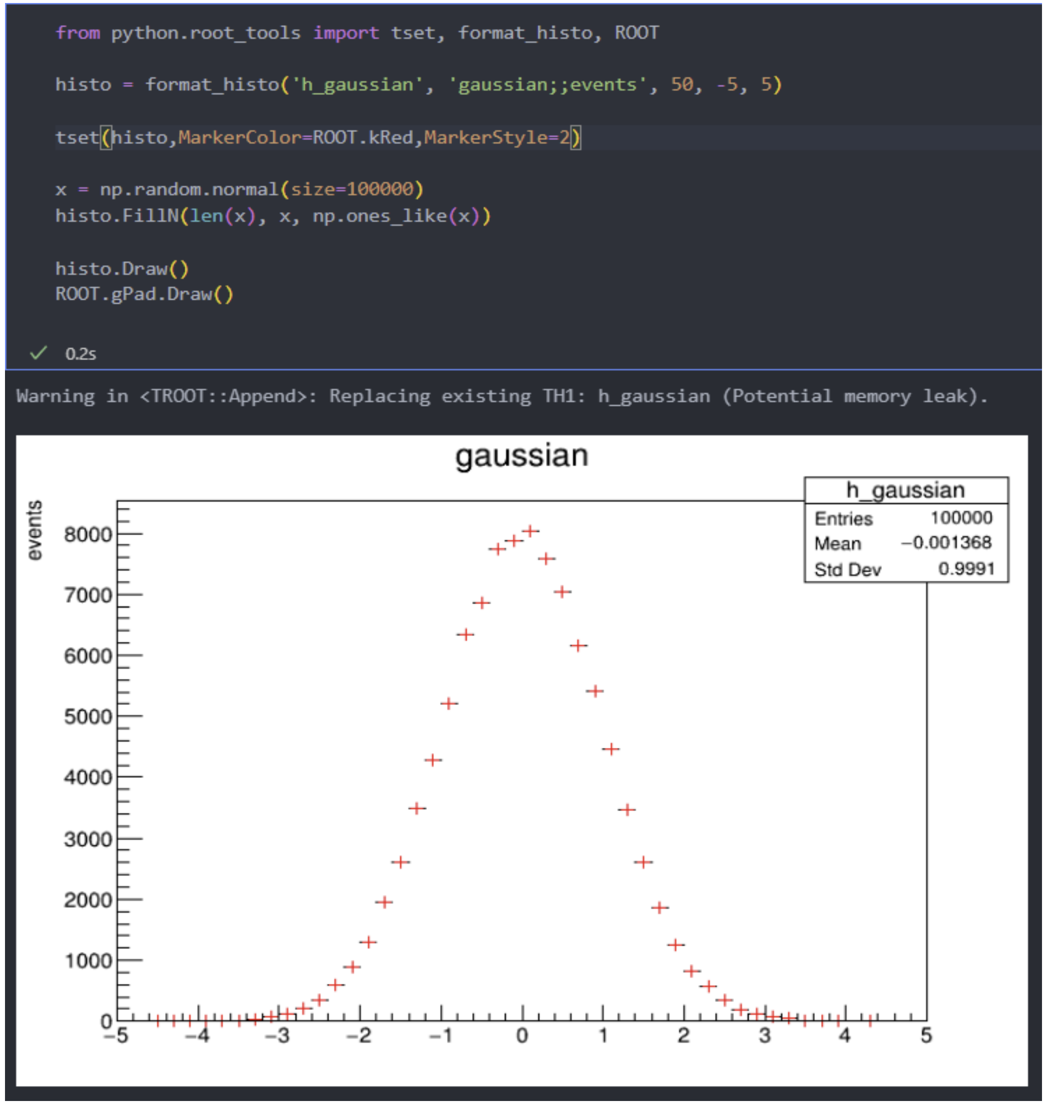
```
h_gaussian
Entries     100000
Mean      0.006063
Std Dev      1.005
```

A more Pythonic ROOT

Along with Configs, the module root_tools.py provides a more pythonic interface with PyROOT

root_tools.tset allows any kwarg to be passed to the TObject

tobj.Set{key}({value})



```
from python.root_tools import tset, format_histo, ROOT

histo = format_histo('h_gaussian', 'gaussian;;events', 50, -5, 5)

tset(histo,MarkerColor=ROOT.kRed,MarkerStyle=2)

x = np.random.normal(size=100000)
histo.FillN(len(x), x, np.ones_like(x))

histo.Draw()
ROOT.gPad.Draw()
```

✓ 0.2s

```
Warning in <TROOT::Append>: Replacing existing TH1: h_gaussian (Potential memory leak).
```

A more Pythonic ROOT

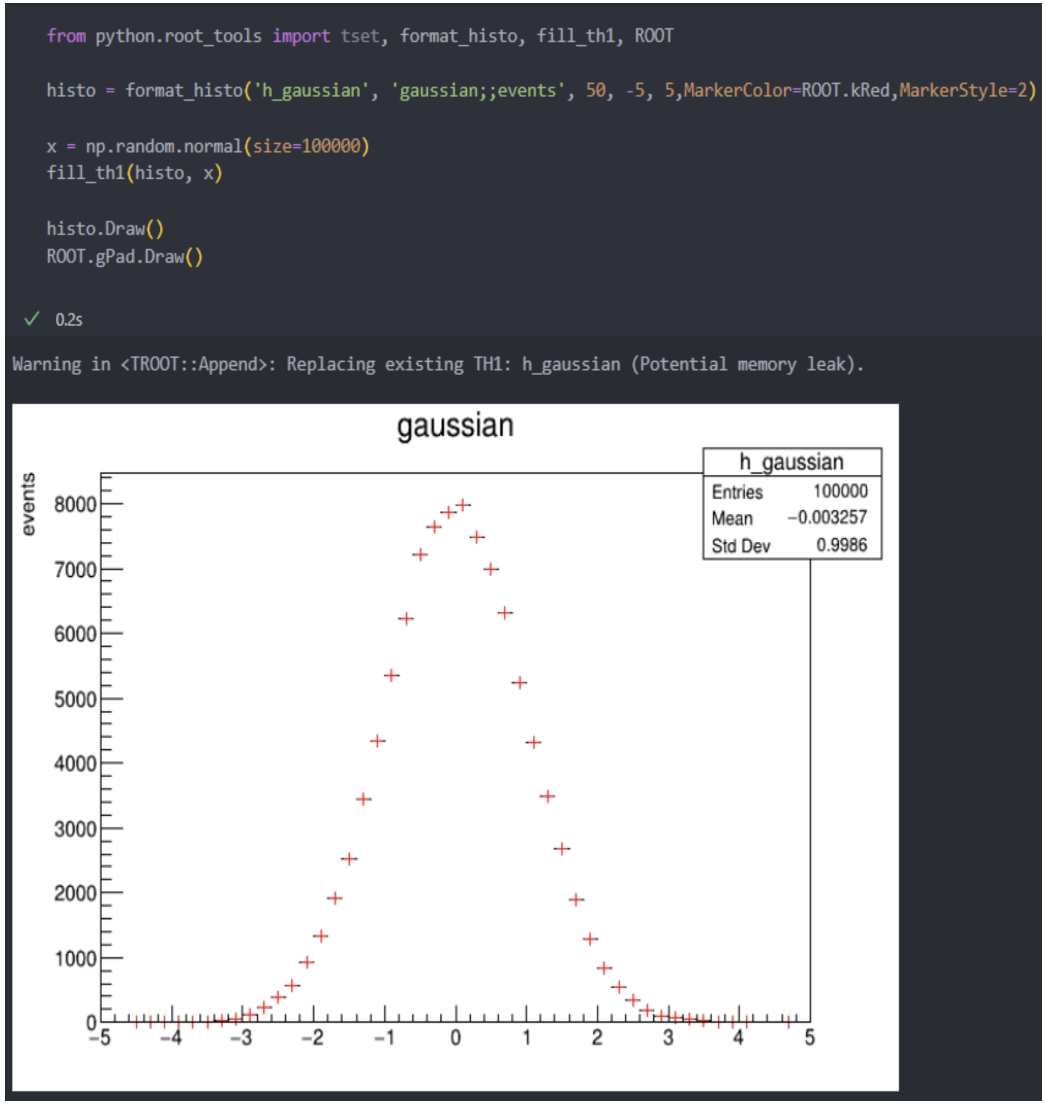
Along with Configs, the module root_tools.py provides a more pythonic interface with PyROOT

root_tools.tset allows any kwarg to be passed to the TObject

tobj.Set{key}({value})

format_histo allows for any kwarg to be passed to set any attribute for the TH1F

root_tools.fill_th1 is also provided to wrap filling a TH1F with a numpy or awkward array



```
from python.root_tools import tset, format_histo, fill_th1, ROOT

histo = format_histo('h_gaussian', 'gaussian;;events', 50, -5, 5,MarkerColor=ROOT.kRed,MarkerStyle=2)

x = np.random.normal(size=100000)
fill_th1(histo, x)

histo.Draw()
ROOT.gPad.Draw()
```

✓ 0.2s

```
Warning in <TROOT::Append>: Replacing existing TH1: h_gaussian (Potential memory leak).
```

L1 Muon Gen Matching Efficiency

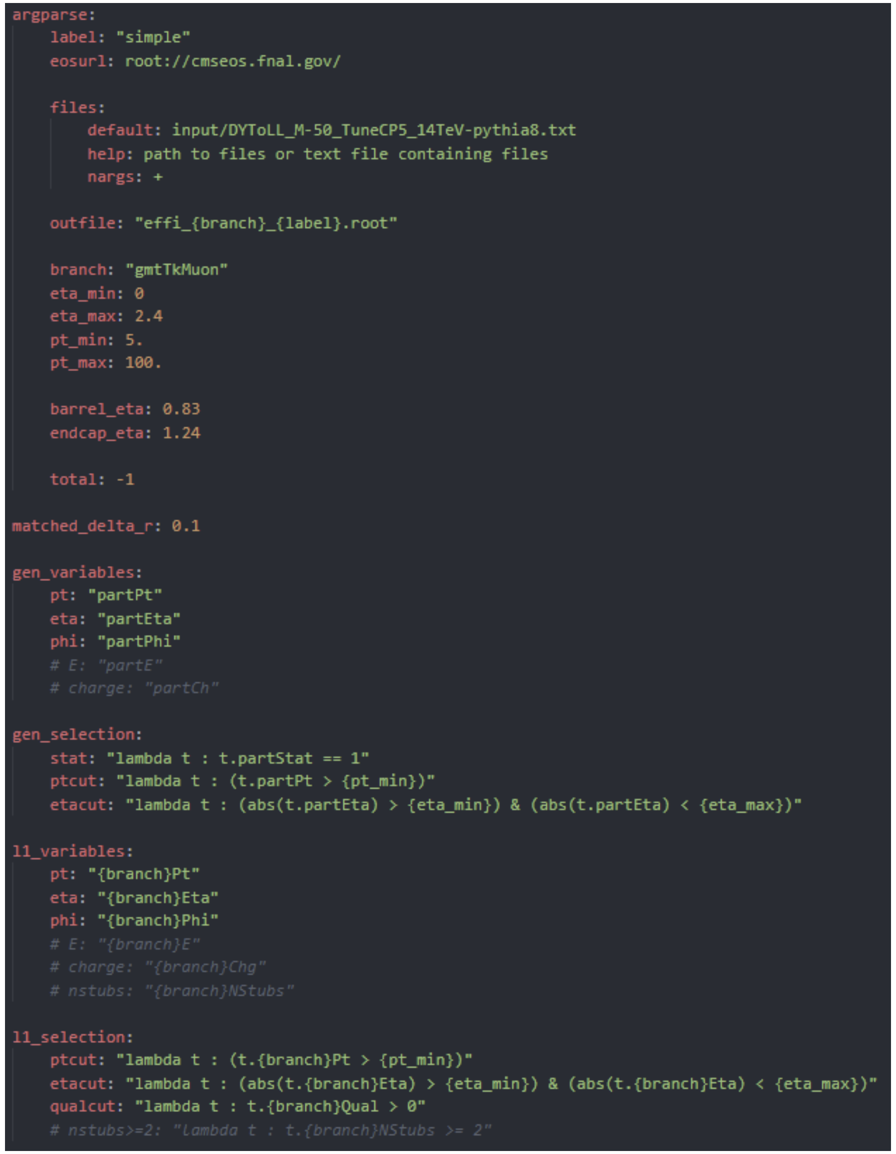
Yaml config for the effiMuonSimple.py script

Takes input from a list of files

- Extra arguments given are passed to argparse.add_argument

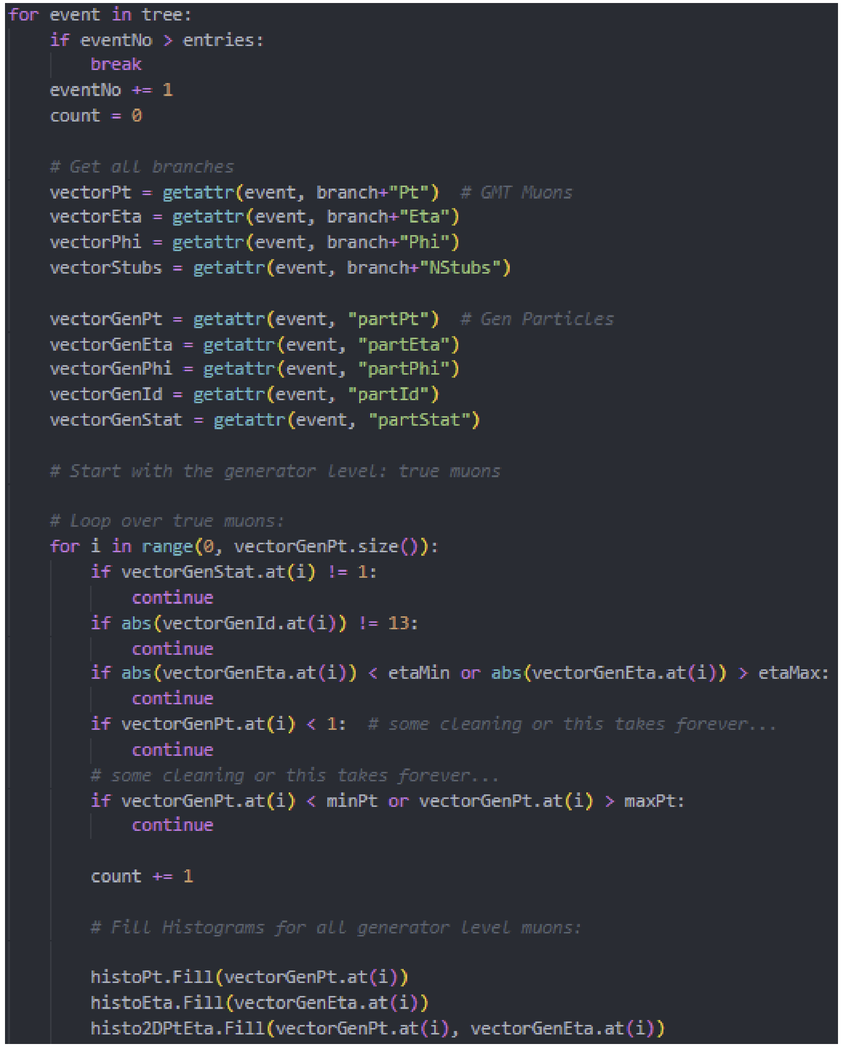
Defined gen and L1 particle selection

Configs can be easily changed from any text editor

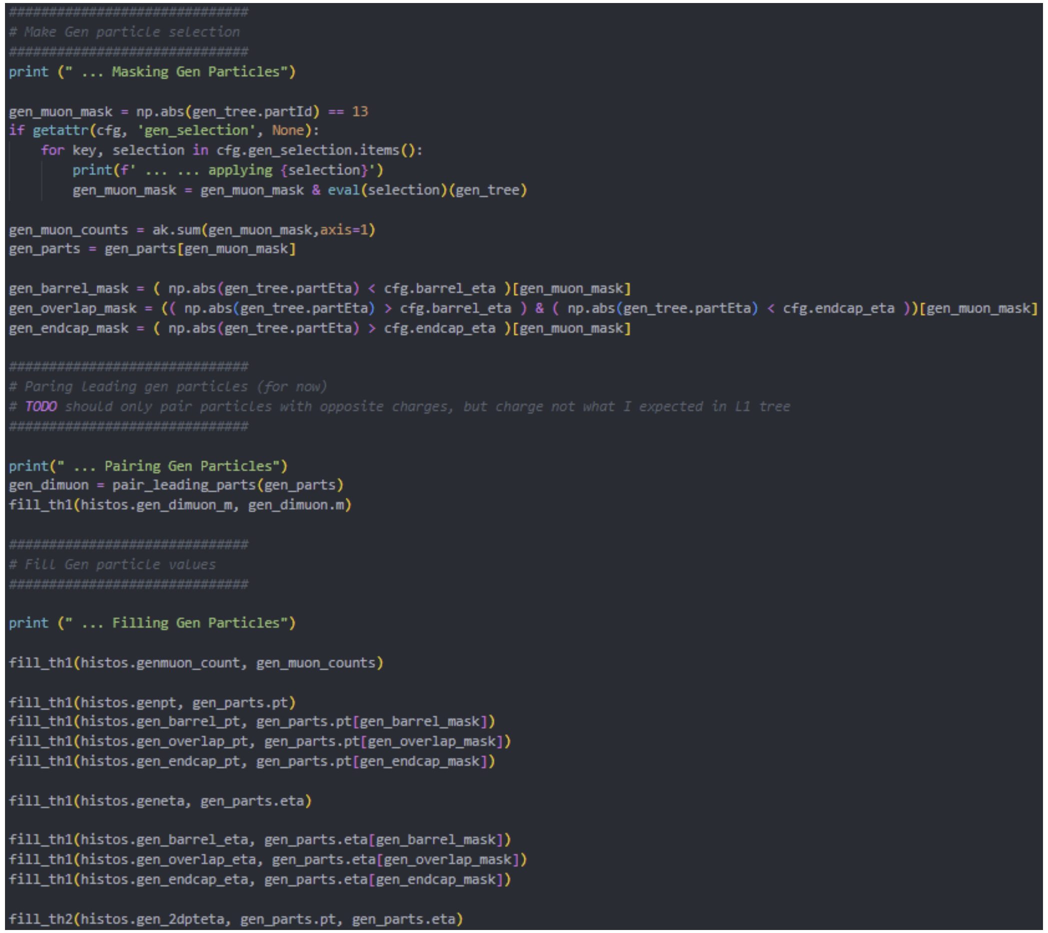


```yaml
argparse:
    label: "simple"
    eosurl: root://cmseos.fnal.gov/

    files:
        default: input/DYToLL_M-50_TuneCP5_14TeV-pythia8.txt
        help: path to files or text file containing files
        nargs: +

    outfile: "effi_{branch}_{label}.root"

    branch: "gmtTkMuon"
    eta_min: 0
    eta_max: 2.4
    pt_min: 5.
    pt_max: 100.

    barrel_eta: 0.83
    endcap_eta: 1.24

    total: -1

matched_delta_r: 0.1

gen_variables:
    pt: "partPt"
    eta: "partEta"
    phi: "partPhi"
    # E: "partE"
    # charge: "partCh"

gen_selection:
    stat: "lambda t : t.partStat == 1"
    ptcut: "lambda t : (t.partPt > {pt_min})"
    etacut: "lambda t : (abs(t.partEta) > {eta_min}) & (abs(t.partEta) < {eta_max})"

l1_variables:
    pt: "{branch}Pt"
    eta: "{branch}Eta"
    phi: "{branch}Phi"
    # E: "{branch}E"
    # charge: "{branch}Chg"
    # nstubs: "{branch}NStubs"

l1_selection:
    ptcut: "lambda t : (t.{branch}Pt > {pt_min})"
    etacut: "lambda t : (abs(t.{branch}Eta) > {eta_min}) & (abs(t.{branch}Eta) < {eta_max})"
    qualcut: "lambda t : t.{branch}Qual > 0"
    # nstubs>=2: "lambda t : t.{branch}NStubs >= 2"
```

Using Awkward Arrays

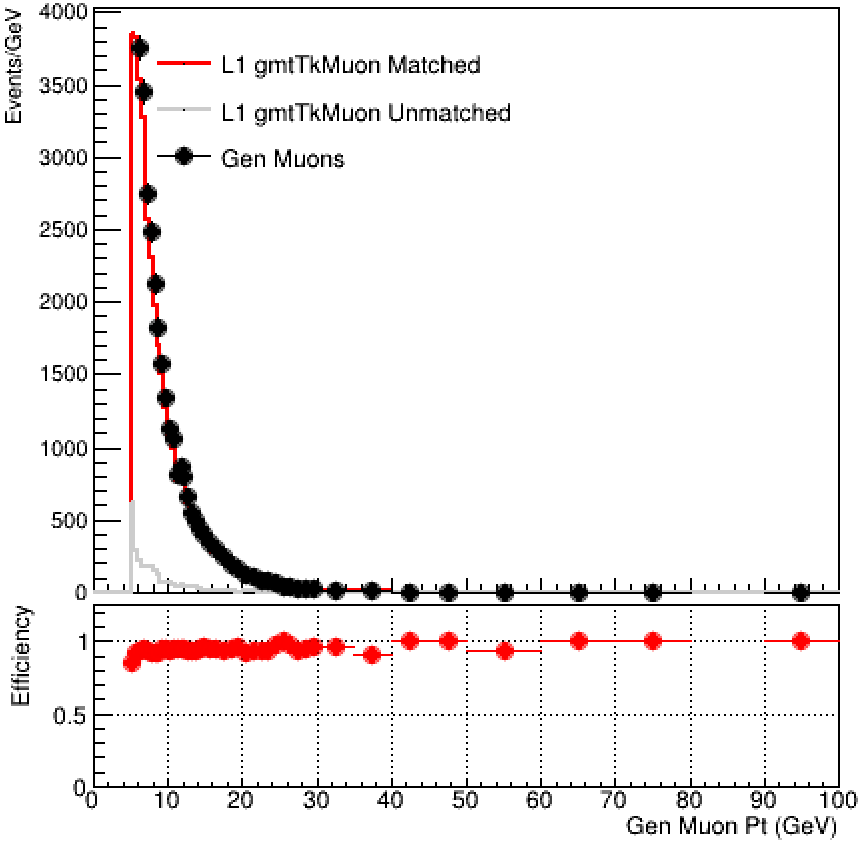


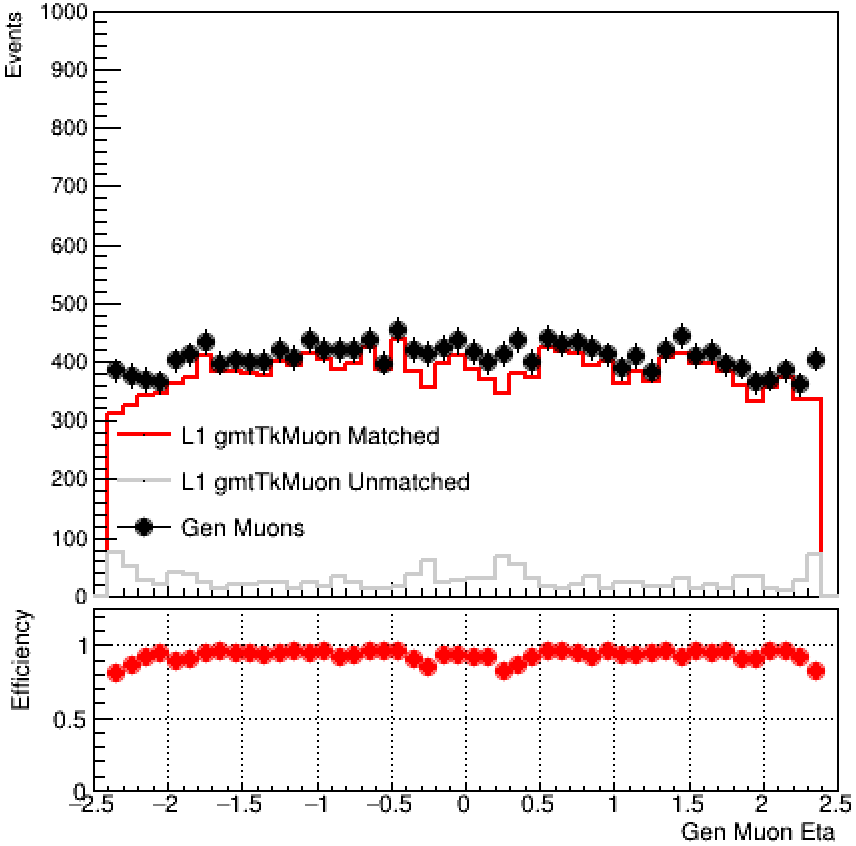
```
for event in tree:
    if eventNo > entries:
        break
    eventNo += 1
    count = 0

    # Get all branches
    vectorPt = getattr(event, branch+"Pt")  # GMT Muons
    vectorEta = getattr(event, branch+"Eta")
    vectorPhi = getattr(event, branch+"Phi")
    vectorStubs = getattr(event, branch+"NStubs")

    vectorGenPt = getattr(event, "partPt")  # Gen Particles
    vectorGenEta = getattr(event, "partEta")
    vectorGenPhi = getattr(event, "partPhi")
    vectorGenId = getattr(event, "partId")
    vectorGenStat = getattr(event, "partStat")

    # Start with the generator level: true muons

    # Loop over true muons:
    for i in range(0, vectorGenPt.size()):
        if vectorGenStat.at(i) != 1:
            continue
        if abs(vectorGenId.at(i)) != 13:
            continue
        if abs(vectorGenEta.at(i)) < etaMin or abs(vectorGenEta.at(i)) > etaMax:
            continue
        if vectorGenPt.at(i) < 1:  # some cleaning or this takes forever...
            continue
        # some cleaning or this takes forever...
        if vectorGenPt.at(i) < minPt or vectorGenPt.at(i) > maxPt:
            continue

        count += 1

        # Fill Histograms for all generator level muons:

        histoPt.Fill(vectorGenPt.at(i))
        histoEta.Fill(vectorGenEta.at(i))
        histo2DPtEta.Fill(vectorGenPt.at(i), vectorGenEta.at(i))
```



```
##############################
# Make Gen particle selection
##############################
print (" ... Masking Gen Particles")

gen_muon_mask = np.abs(gen_tree.partId) == 13
if getattr(cfg, 'gen_selection', None):
    for key, selection in cfg.gen_selection.items():
        print(f' ... ... applying {selection}')
        gen_muon_mask = gen_muon_mask & eval(selection)(gen_tree)

gen_muon_counts = ak.sum(gen_muon_mask,axis=1)
gen_parts = gen_parts[gen_muon_mask]

gen_barrel_mask = ( np.abs(gen_tree.partEta) < cfg.barrel_eta )[gen_muon_mask]
gen_overlap_mask = (( np.abs(gen_tree.partEta) > cfg.barrel_eta ) & ( np.abs(gen_tree.partEta) < cfg.endcap_eta ))[gen_muon_mask]
gen_endcap_mask = ( np.abs(gen_tree.partEta) > cfg.endcap_eta )[gen_muon_mask]

##############################
# Paring leading gen particles (for now)
# TODO should only pair particles with opposite charges, but charge not what I expected in L1 tree
##############################

print(" ... Pairing Gen Particles")
gen_dimuon = pair_leading_parts(gen_parts)
fill_th1(histos.gen_dimuon_m, gen_dimuon.m)

##############################
# Fill Gen particle values
##############################

print (" ... Filling Gen Particles")

fill_th1(histos.genmuon_count, gen_muon_counts)

fill_th1(histos.genpt, gen_parts.pt)
fill_th1(histos.gen_barrel_pt, gen_parts.pt[gen_barrel_mask])
fill_th1(histos.gen_overlap_pt, gen_parts.pt[gen_overlap_mask])
fill_th1(histos.gen_endcap_pt, gen_parts.pt[gen_endcap_mask])

fill_th1(histos.geneta, gen_parts.eta)

fill_th1(histos.gen_barrel_eta, gen_parts.eta[gen_barrel_mask])
fill_th1(histos.gen_overlap_eta, gen_parts.eta[gen_overlap_mask])
fill_th1(histos.gen_endcap_eta, gen_parts.eta[gen_endcap_mask])

fill_th2(histos.gen_2dpteta, gen_parts.pt, gen_parts.eta)
```

L1 Muon Gen Matching Efficiency





L1 Muon Gen Di-Muon

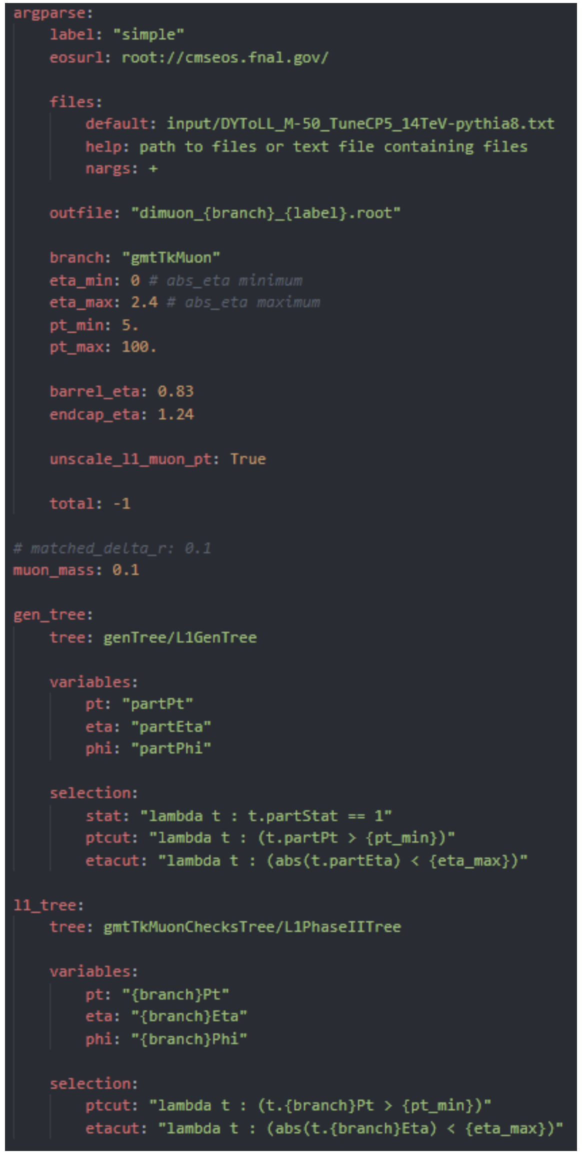
Yaml config for the diMuonSimple.py script

Takes input from a list of files

- Extra arguments given are passed to argparse.add_argument

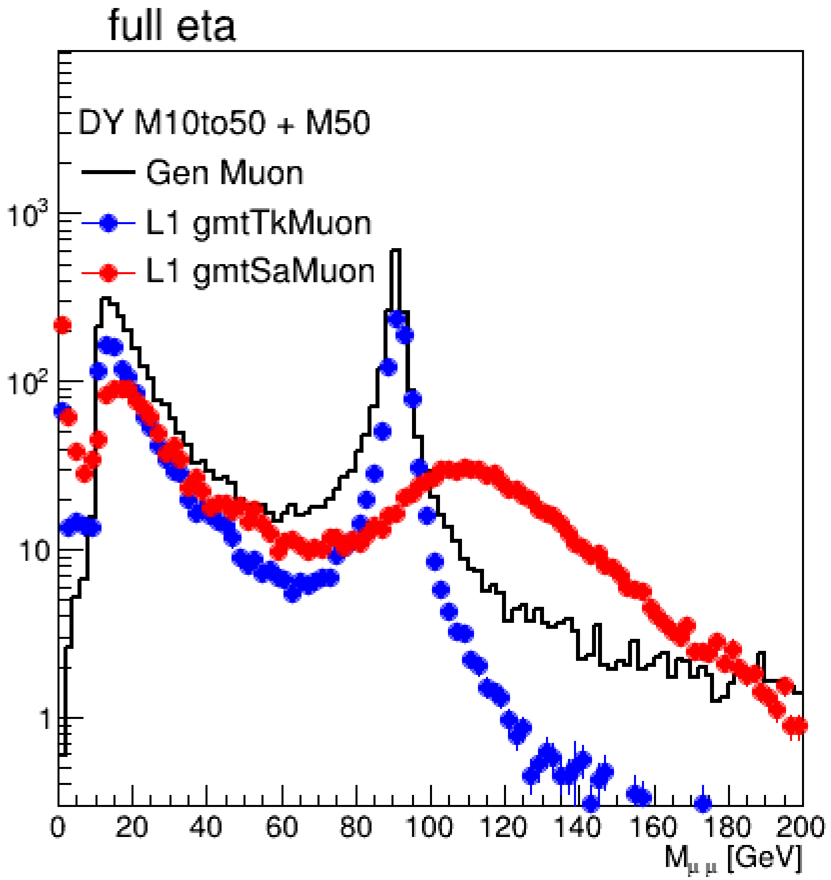
Defined gen and L1 particle selection

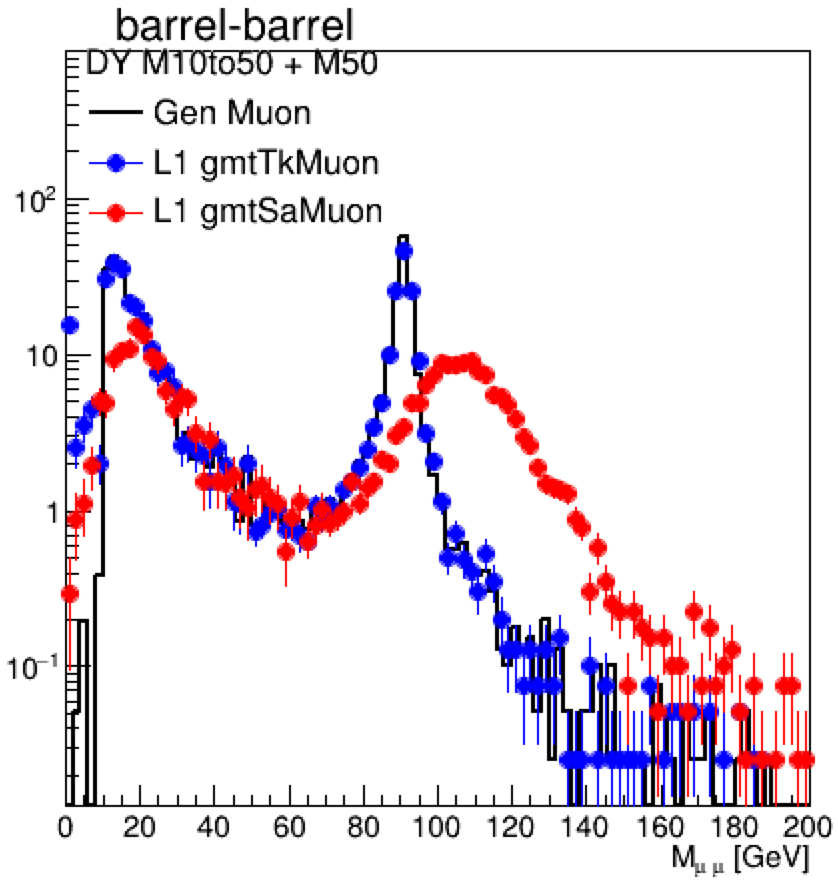
Configs can be easily changed from any text editor



```yaml
argparse:
    label: "simple"
    eosurl: root://cmseos.fnal.gov/

    files:
        default: input/DYToLL_M-50_TuneCP5_14TeV-pythia8.txt
        help: path to files or text file containing files
        nargs: +

    outfile: "dimuon_{branch}_{label}.root"

    branch: "gmtTkMuon"
    eta_min: 0 # abs_eta minimum
    eta_max: 2.4 # abs_eta maximum
    pt_min: 5.
    pt_max: 100.

    barrel_eta: 0.83
    endcap_eta: 1.24

    unscale_l1_muon_pt: True

    total: -1

# matched_delta_r: 0.1
muon_mass: 0.1

gen_tree:
    tree: genTree/L1GenTree

    variables:
        pt: "partPt"
        eta: "partEta"
        phi: "partPhi"

    selection:
        stat: "lambda t : t.partStat == 1"
        ptcut: "lambda t : (t.partPt > {pt_min})"
        etacut: "lambda t : (abs(t.partEta) < {eta_max})"

l1_tree:
    tree: gmtTkMuonChecksTree/L1PhaseIITree

    variables:
        pt: "{branch}Pt"
        eta: "{branch}Eta"
        phi: "{branch}Phi"

    selection:
        ptcut: "lambda t : (t.{branch}Pt > {pt_min})"
        etacut: "lambda t : (abs(t.{branch}Eta) < {eta_max})"
```

L1 Muon Gen Di-Muon





Summary

Framework is usable now for CMSSW 12_5_X software

Implementing various studies including

- Gen Matching Efficiency
- Di-Muon Mass Reconstruction
- Rate Calculations

More pythonic and user-friendly interfaces with PyROOT

Utilizing efficient TTree processing using uproot and awkward

Future goals:

Continue to implement better pythonic interfaces with PyROOT

Include better file management for files opened over remote (i.e. root://eoscms.cern.ch/)

More improvements will be made as more studies are needed