LHEprod
Yet a new repo for LHE studies

G. Boldrini - CNRS/IN2P3 - LLR, École polytechnique

giacomo.boldrini@cern.ch

Why do we need a new framework for LHE analyses?

- Previous one (D6EFTStudies) good for generating MG5 folders but **ntuplizer tailored to final states**
- Missing theory **uncertainties** (μ_F, μ_R and PDFs)
- **Ntuple structure not optimal**: xsecs stored in TH1Fs → cannot hadd!

Idea:

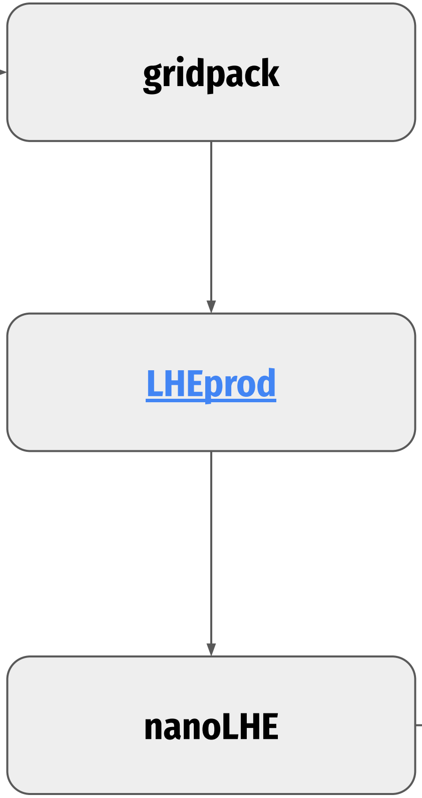
NanoAOD workflow stores all LHE information. However nanoGEN expects parton-shower. The module storing LHE infos is coupled to PS due to PS weights: we can decouple this requirement and make a **LHE-only reader.**

nanoAOD data structures: can use analyses libraries for analysis!

Introduction

genproductions

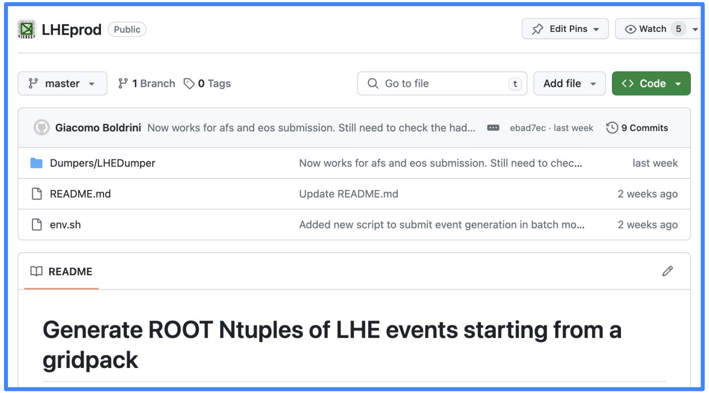
(*) Instruction on how to generate gridpacks are not included in this presentation.



gridpack

LHEprod

nanoLHE



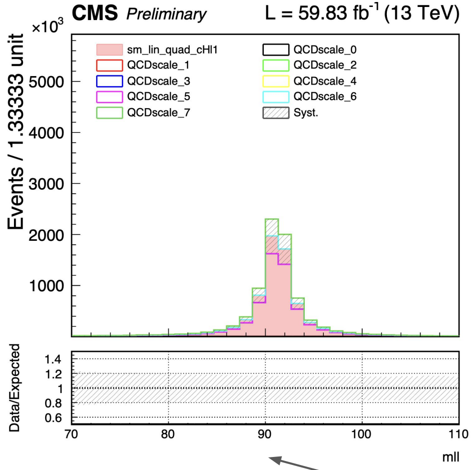
LHEprod Public

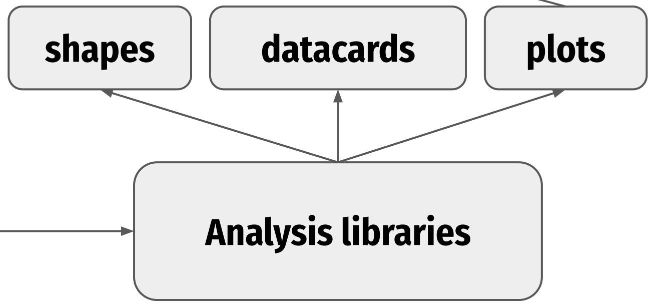
Giacomo Boldrini Now works for afs and eos submission. Still need to check the had... ebad7ec · last week 9 Commits

Dumpers/LHEDumper	Now works for afs and eos submission. Still need to chec...	last week
README.md	Update README.md	2 weeks ago
env.sh	Added new script to submit event generation in batch mo...	2 weeks ago

README

Generate ROOT Ntuples of LHE events starting from a gridpack





shapes

datacards

plots

Analysis libraries

(*) Will use mkShapesRDF but you can use the nanoAOD framework you prefer

LHE level DY production + SMEFT

From generation to constraints to dim6 operators

Tutorial

Connect to lxplus

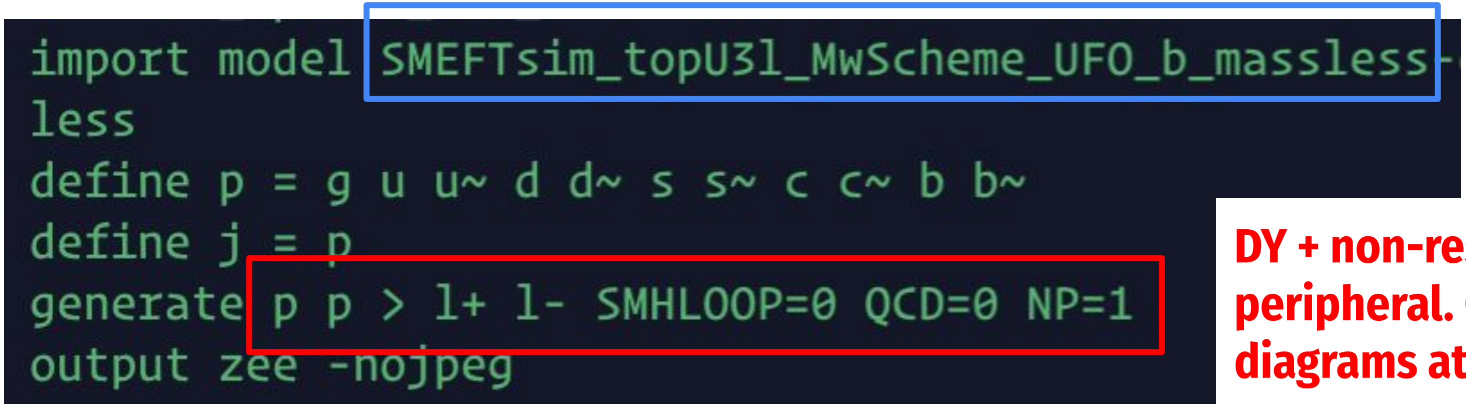
```
> ssh <username>@lxplus.cern.ch # el9
> ssh <username>@lxplus8.cern.ch # el8
> ssh <username>@lxplus7.cern.ch # el7
```

Download a simple DY gridpack at LO

```
> wget https://gboldrin.web.cern.ch/gboldrin/generators/zee_slc7_amd64_gcc700_CMSSW_10_6_19_tarball.tar.xz
```

Inspect the gridpack:

```
> mkdir zee_gridpack
> tar -axf
zee_slc7_amd64_gcc700_CMSSW_10_6_19_tarball.tar.xz
--directory zee_gridpack/
> cd zee_gridpack/InputCards
> cat zee_proc_card.dat
```



```
import model SMEFTsim_topU3l_MwScheme_UFO_b_massless-
less
define p = g u u~ d d~ s s~ c c~ b b~
define j = p
generate p p > l+ l- SMHLOOP=0 QCD=0 NP=1
output zee -nojpeg
```

UFO MODEL

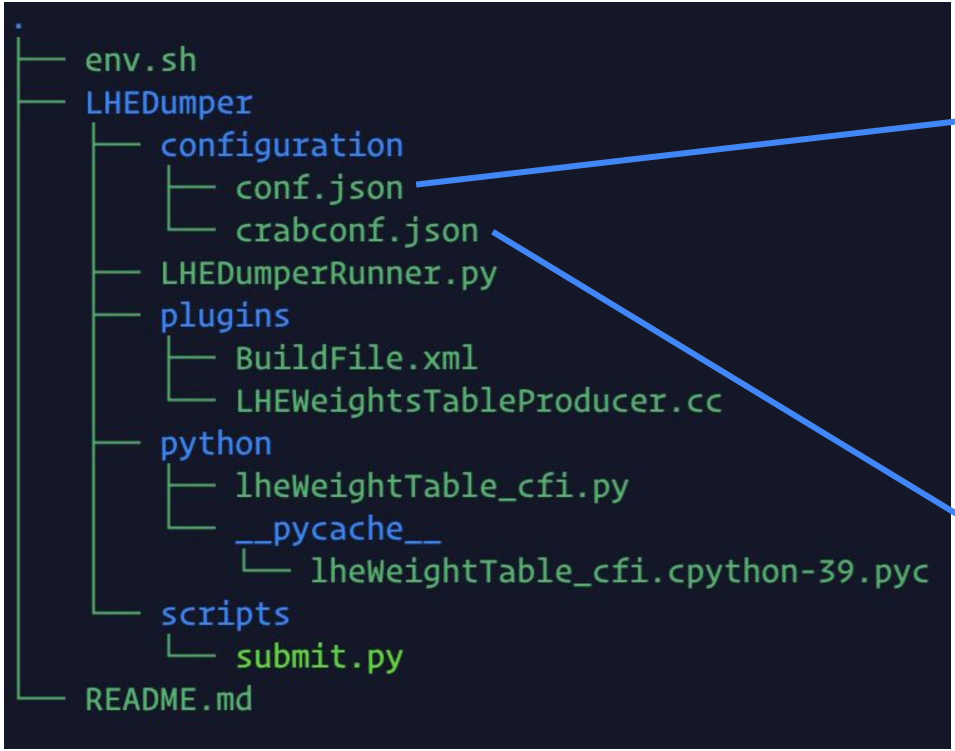
DY + non-resonant + peripheral. QED-induced diagrams at LO O(α_{EW}^2)

Tutorial

Setup the CMS software environment:

```
> cmsrel CMSSW_13_1_0 # working on el8
> cd CMSSW_13_1_0/src
> git clone git@github.com:UniMiBAnalyses/LHEprod.git
> cd LHEprod
> source env.sh; cmsenv # do this every time you connect
to lxplus
> scram b -j 8
```

Tutorial



```
.
├── env.sh
├── LHEDumper
│   ├── configuration
│   │   ├── conf.json
│   │   └── crabconf.json
│   ├── LHEDumperRunner.py
│   ├── plugins
│   │   ├── BuildFile.xml
│   │   └── LHEWeightsTableProducer.cc
│   ├── python
│   │   ├── lheWeightTable_cfi.py
│   │   └── __pycache__
│   │       └── lheWeightTable_cfi.cpython-39.pyc
│   └── scripts
│       └── submit.py
└── README.md
```

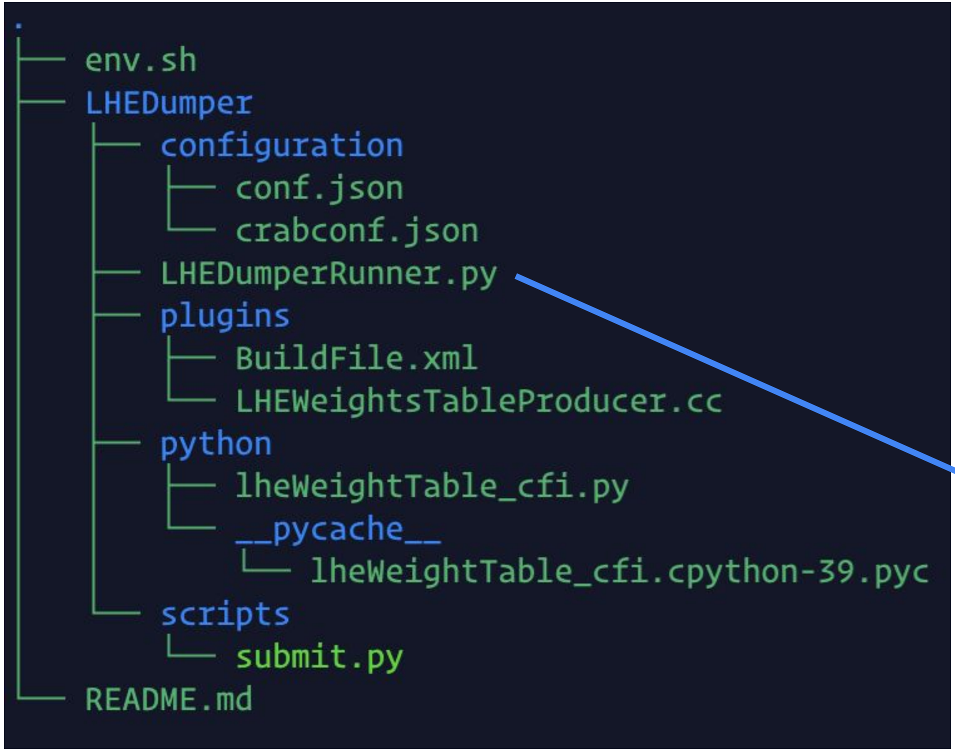
Parameter setup for the jobs

```
{
    "SCRAM": "el9_amd64_gcc12",
    "CMSSW": "CMSSW_13_3_0",
    "EOS_MGM_URL": "root://eosuser.cern.ch",
    "LHE_PREFIX": "nAOD_LHE"
}
```

Parameter setup for crab jobs

```
{
    "storageSite": "T3_IT_MIB",
    "outputPrimaryDataset": "nAOD_LHE",
    "requestName": "prova_nanoAOD_LHE_framework",
    "maxmemory": 2500,
    "outputDatasetTag": "Nanov12LHEOnly"
}
```

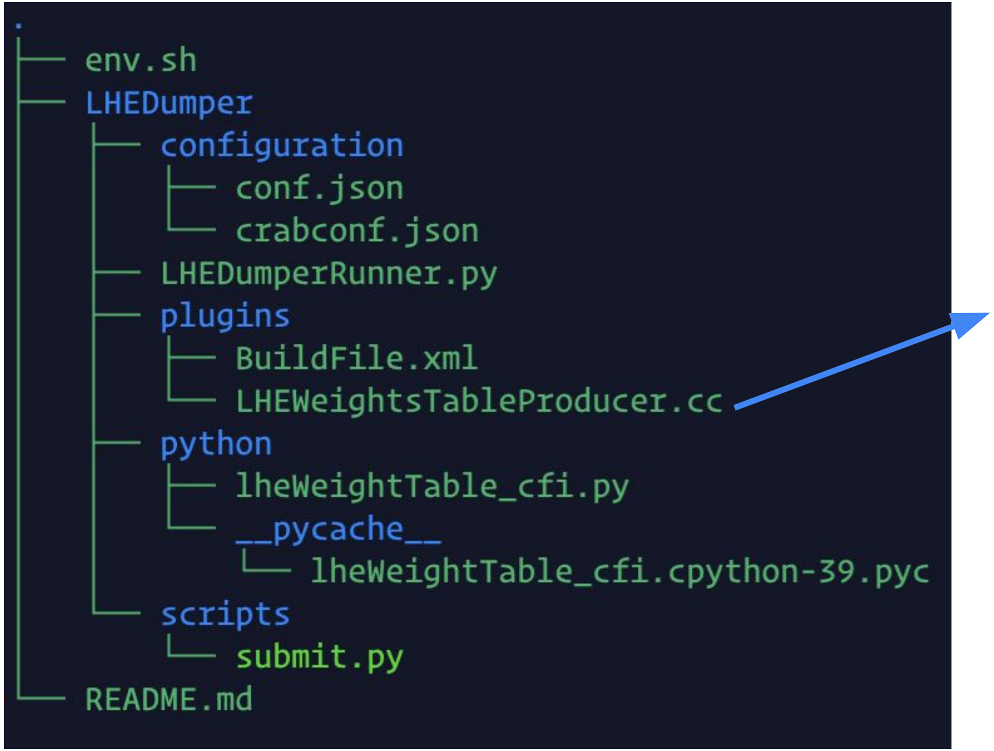
Tutorial



```
.
├── env.sh
├── LHEDumper
│   ├── configuration
│   │   ├── conf.json
│   │   └── crabconf.json
│   ├── LHEDumperRunner.py
│   ├── plugins
│   │   ├── BuildFile.xml
│   │   └── LHEWeightsTableProducer.cc
│   ├── python
│   │   ├── lheWeightTable_cfi.py
│   │   └── __pycache__
│   │       └── lheWeightTable_cfi.cpython-39.pyc
│   └── scripts
│       └── submit.py
└── README.md
```

cmsRun configuration script that runs the event generation from a gridpack and stores the LHE information in a .root file with same content like nanoAOD file format

Tutorial

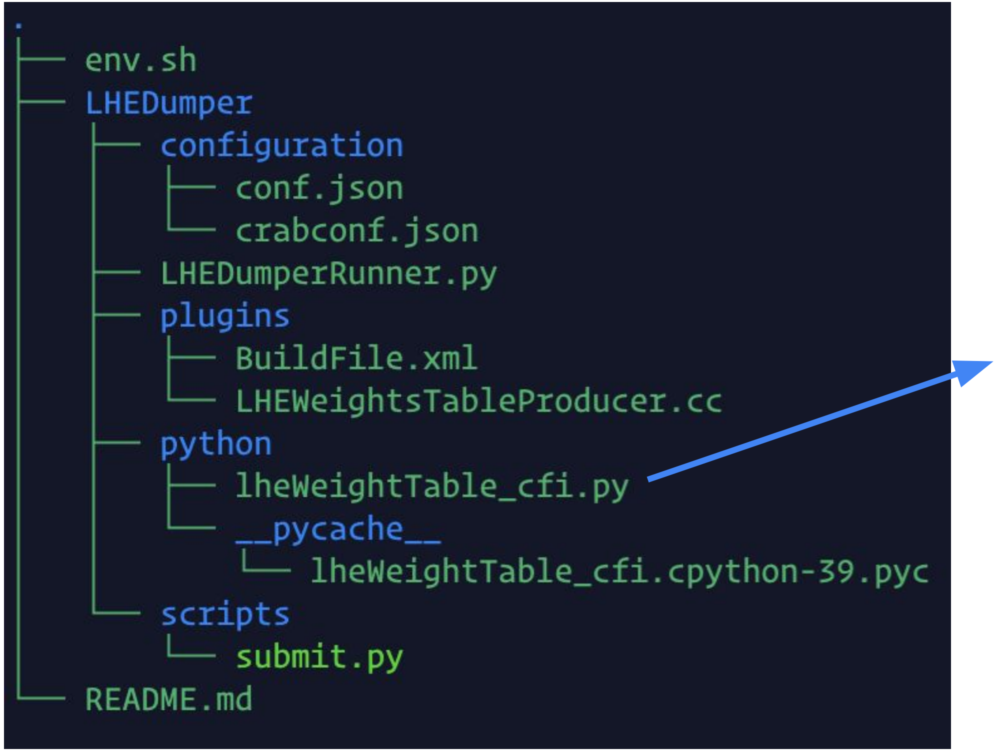


```
.
├── env.sh
├── LHEDumper
│   ├── configuration
│   │   ├── conf.json
│   │   └── crabconf.json
│   ├── LHEDumperRunner.py
│   ├── plugins
│   │   ├── BuildFile.xml
│   │   └── LHEWeightsTableProducer.cc
│   ├── python
│   │   ├── lheWeightTable_cfi.py
│   │   └── __pycache__
│   │       └── lheWeightTable_cfi.cpython-39.pyc
│   └── scripts
│       └── submit.py
└── README.md
```

This is the real plugin, it is a modified version of GenWeightsTableProducer.cc where everything related to Parton Shower weights has been dropped.
It will save

- **Every parton level state 4-momenta**
- **LHE particles pdgID**
- **LHE particle status**
- **Event xsection weight (xsec / nevents)**
- **QCD scale variations**
- **PDF variations**
- **Reweighting weights (if any)**

Tutorial

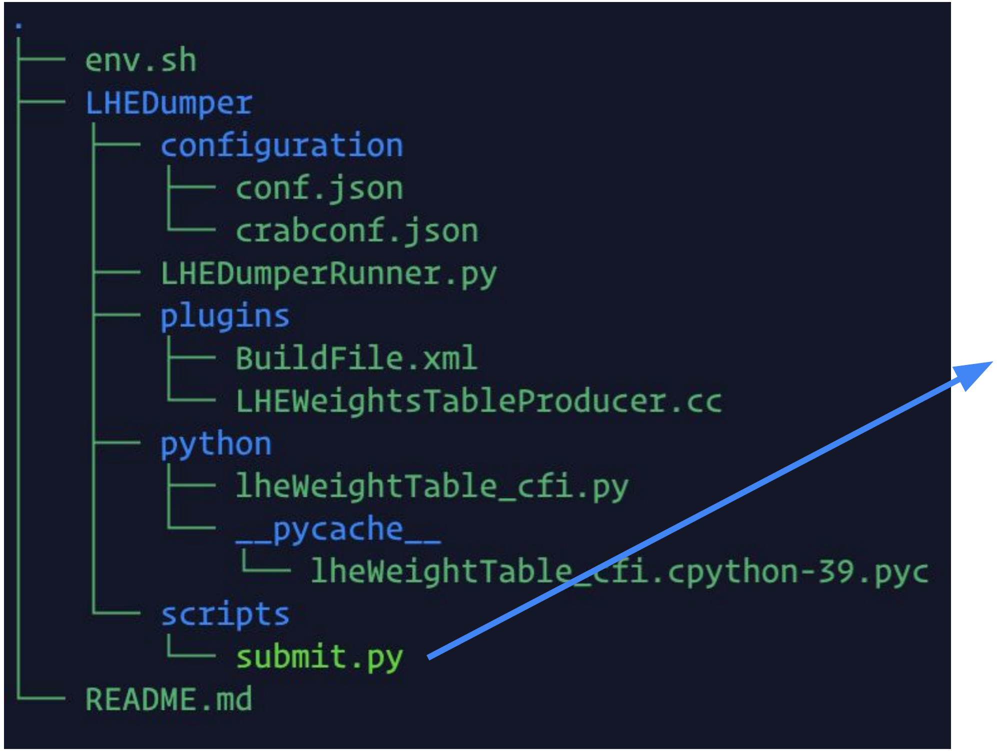


```
.
├── env.sh
├── LHEDumper
│   ├── configuration
│   │   ├── conf.json
│   │   └── crabconf.json
│   ├── LHEDumperRunner.py
│   ├── plugins
│   │   ├── BuildFile.xml
│   │   └── LHEWeightsTableProducer.cc
│   ├── python
│   │   ├── lheWeightTable_cfi.py
│   │   └── __pycache__
│   │       └── lheWeightTable_cfi.cpython-39.pyc
│   └── scripts
│       └── submit.py
└── README.md
```

The LHEWeightsTableProcuder.cc plugin configuration file to be integrated into cmsRun configuration files.
It stores some baseline config settings:

- **Source type (ExternalLHEProducer)**
- **PDF sets to consider**
- **named weights IDs (if any)**
- **named weight Labels (if any)**
- **lhe weight precision (14 digits)**
- **Max number of PDF variations (150)**
- **debug (extra print info)**

Tutorial



```
.
├── env.sh
├── LHEDumper
│   ├── configuration
│   │   ├── conf.json
│   │   └── crabconf.json
│   ├── LHEDumperRunner.py
│   ├── plugins
│   │   ├── BuildFile.xml
│   │   └── LHEWeightsTableProducer.cc
│   ├── python
│   │   ├── lheWeightTable_cfi.py
│   │   └── __pycache__
│   │       └── lheWeightTable_cfi.cpython-39.pyc
│   └── scripts
│       └── submit.py
└── README.md
```

Script to submit the event generation from a gridpack:

- on batch
- on crab

Save output on afs, eos or tier (in case of crab)

```
 -gp GRIDPACK, --gridpack GRIDPACK
            Path to the gridpack you want to generate events with. [REQUIRED]
 -o OUTPUT, --output OUTPUT
            Output folder where .root files will be stored. If using crab something like /store/user/<username>/...
[REQUIRED]
 -ne NEVENTS, --nevents NEVENTS
            Number of events per job requested (def=1000)
 -nj NJOBS, --njobs NJOBS
            Number of jobs requested (def=1)
 -nt NTHREADS, --nthreads NTHREADS
            Number of threads x job (def=1)
 -t TIER, --tier TIER  Tier for production. can be [afs, eos, crab]. Afs will submit jobs with HTcondor and save root files on
afs while eos option will xrdcp to eos. crab instead will save on the
            specified tier in the crab config file
 -q QUEUE, --queue QUEUE
            Condor queue (def=longlunch)
 -m MERGE, --merge MERGE
            Collect output root files and merge them. Provide an output path at -m (default=None)
 -cm CRABMERGE, --crabmerge CRABMERGE
            For crab merge, you need to also specify the crab direcotry in your local afs so we can gather the
necessary information of the published dataset
 --conf CONF      Load configuration file (default=configuration/conf.json)
 --crabconf CRABCONF   Crab config json file (default=configuration/crabconf.json)
 --datasetname DATASETNAME
            Crab dataset name under config.Data.outputPrimaryDataset. Can also specify in crabconfig
 --requestname REQUESTNAME
            Name of the crab request under config.General.requestName
 --datasettag DATASETTAG
            Name of the crab dataset tag under config.Data.outputDatasetTag
 --maxmemory MAXMEMORY
            Max memory of the crab request under config.JobType.maxMemoryMB
```

Tutorial

The LHEDumperRunner.py script can be executed locally and works out of the box given a gridpack. Some command line arguments need to be provided:

```
options.register('jobNum',
          0,
          "jobNum")
```

This is mandatory for crab jobs but useless for any other mode.

```
options.register('nthreads',
          1,
          "number of threads")
```

Can execute cmsRun in multithread. By default single-thread execution

```
options.register('input',
          'gridpack.tar.xz',
          "input gridpack path (without file: prefix)")
```

The input gridpack from which to generate events

```
options.register('nevents',
          10,
          "Number of events to generate")
```

Number of events to be generated

```
options.register('seed',
          10,
          "Random number used as initial seed for this generation")
```

Random number for the event generation. Be sure to change it if running locally!

Tutorial

Let’s generate 1000 events from our gridpacks locally

```
> cmsRun LHEDumperRunner.py
input=/afs/cern.ch/user/g/gboldrin/zee slc7_amd64_gcc700_CMS
SW_10_6_19_tarball.tar.xz  nevents=1000
```

```
         Running Generic Tarball/Gridpack

gridpack tarball path = /afs/cern.ch/user/g/gboldrin/zee_slc7_amd64_gcc700_CMSSW_10_6_19_tarball.tar.xz
%MSG-MG5 number of events requested = 1000
%MSG-MG5 random seed used for the run = 11
%MSG-MG5 thread count requested = 1
%MSG-MG5 residual/optional arguments =
…
```

A root file named nAOD_LHE.root will be created

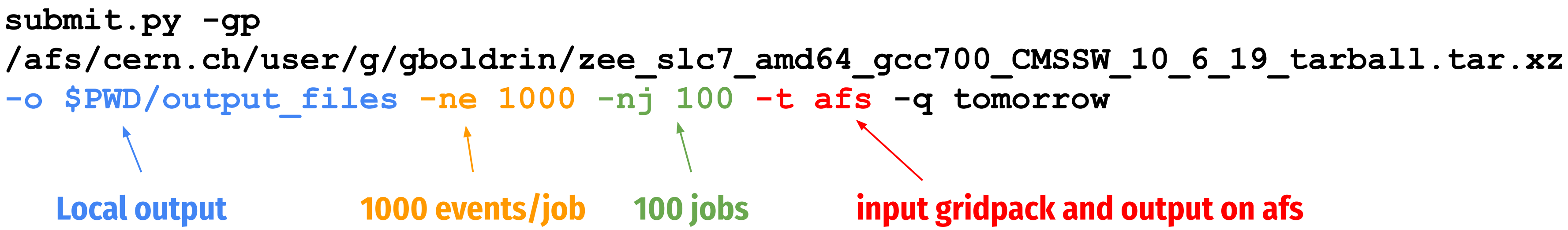
```
[gboldrin@lxplus807 LHEDumper]$ ls
configuration  LHEDumperRunner.py   nAOD_LHE.root  plugins  python  scripts
thread0
```

Let’s inspect one of these root files

```
>>> import ROOT
>>> f = ROOT.TFile("nAOD_LHE.root")
>>> f.ls()
    TFile**         nAOD_LHE.root
     TFile*         nAOD_LHE.root
      KEY: TTree    Events;1        Events
      KEY: TTree    LuminosityBlocks;1      LuminosityBlocks
      KEY: TTree    Runs;1  Runs
      KEY: TTree    MetaData;1      Job metadata
      KEY: TTree    ParameterSets;1 Parameter sets
>>> t = f.Get("Events")
>>> [i.GetName() for i in t.GetListOfBranches()]
['run', 'luminosityBlock', 'event', 'bunchCrossing', 'LHE_Njets', 'LHE_Nb', 'LHE_Nc',
'LHE_Nuds', 'LHE_Nglu', 'LHE_NpNLO', 'LHE_NpLO', 'LHE_HT', 'LHE_HTIncoming', 'LHE_Vpt',
'LHE_AlphaS', 'nLHEPart', 'LHEPart_pdgId', 'LHEPart_status', 'LHEPart_spin',
'LHEPart_pt', 'LHEPart_eta', 'LHEPart_phi', 'LHEPart_mass', 'LHEPart_incomingpz',
'LHEWeight_originalXWGTUP', 'nLHEPdfWeight', 'LHEPdfWeight', 'nLHEReweightingWeight',
'LHEReweightingWeight', 'nLHEScaleWeight', 'LHEScaleWeight']
```

Tutorial

If you want to generate a lot of events, batch submission will save a lot of time. The script submit.py is designed to simply submit the LHEDumperRunner.py script on lxbatch or crab



```
submit.py -gp
/afs/cern.ch/user/g/gboldrin/zee_slc7_amd64_gcc700_CMSSW_10_6_19_tarball.tar.xz
-o $PWD/output_files -ne 1000 -nj 100 -t afs -q tomorrow
```

Local output

1000 events/job

100 jobs

input gridpack and output on afs

```
--> Info: condor logs will be saved at
/afs/cern.ch/user/g/gboldrin/CMSSW_13_1_0/src/LHEprod/LHEDumper/.condorsub/condo
r_log
--> Warning setup condor jobs with SCRAM=el9_amd64_gcc12 and CMSSW=CMSSW_13_3_0
Submitting
job(s)..................................................................
..........................
100 job(s) submitted to cluster 7221168.
```

Tutorial

Similarly, you can submit on condor if the input gridpack is on eos or if the output folder to store the .root is eos-based:

```
submit.py -gp
/eos/user/g/gboldrin/gridpacks/zee/zee_slc7_amd64_gcc700_CMSSW_10_6_19_tarball.ta
r.xz -o /eos/user/g/gboldrin/prova -ne 1000 -nj 100 -t eos -q tomorrow
```

eos output

1000 events/job

100 jobs

input gridpack and output on eos

```
--> Info: condor logs will be saved at
/afs/cern.ch/user/g/gboldrin/CMSSW_13_1_0/src/LHEprod/LHEDumper/.condorsub/condo
r_log
--> Warning setup condor jobs with SCRAM=el9_amd64_gcc12 and CMSSW=CMSSW_13_3_0
Submitting
job(s)................................................................
..........................
100 job(s) submitted to cluster 9263826.
```

Tutorial

Every output file will contain 1000 events. **The cross section of the process will now be** σ = `sum(LHEWeight_originalXWGTUP) ~ 1000 x LHEWeight_originalXWGTUP.` In order to use correctly all the files we need to re-assign the event weight such that $\omega = \sigma$ / (Nevents) = σ / (1000 x 100).

This can be done with **submit.py merge option or simply by adding the baseW to each file**. In any case a new branch **'baseW'** will be created:

MERGE

```
submit.py -o $PWD/output_files  -m output.root
```

Take all files in this directory

hadd all files into this file + add baseW

BASEW

```
submit.py -o $PWD/output_files  --basew
```

Take all files in this directory

Add baseW to each of the files

Tutorial

Now that we have the ntuplized LHE events, let’s analyze them.

We will use mkShapesRDF

```
> cd $CMSSW_BASE/src # if you followed the example you should be in CMSSW_13_1_0/src
> git clone https://github.com/giorgiopizz/mkShapesRDF.git
> cd mkShapesRDF
> git checkout shapes-dev
> ./install.sh
> source start.sh # always run when in new shell
```

Tutorial

Now that we have the ntuplized LHE events, let’s analyze them.
We will use mkShapesRDF

```
> cd $CMSSW_BASE/src # if you followed the example you should be in CMSSW_13_1_0/src
> git clone https://github.com/giorgiopizz/mkShapesRDF.git
> cd mkShapesRDF
> git checkout shapes-dev
> ./install.sh
> source start.sh # always run when in new shell
```