DUNE workflows on the EAF for analyzing data from ProtoDUNE-II

Lewhat Aylay

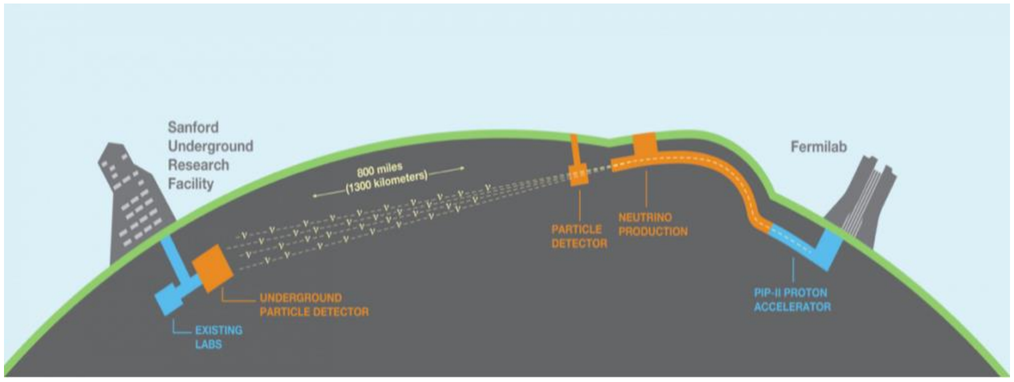
Florida State University

August 12, 2022

1 Introduction

1.1 Background

The Deep Underground Neutrino Experiment (DUNE) is an international experiment that works to unlock the mysteries of neutrinos. The Deep Underground Neutrino Experiments has three science goals, which are to search the origin of matter and if neutrinos could be the reason the universe is made of matter. Another science goal of DUNE is to learn more about neutron stars and black holes and watch for neutrinos emerging from an exploding star. Lastly, the third science goal of DUNE is to shed light on the unification of nature’s forces [1]. There are two neutrino detectors placed in the world, one detector is at the Fermi National Accelerator Laboratory where particle interactions near the source of the beam is recorded. Where another detector is installed at the Sanford Underground Research Laboratory. In addition, two prototype far detectors are also installed at the European research center where data has been collected since 2018.


Sanford Underground Research Facility
800 miles (1300 kilometers)
Fermilab
PARTICLE DETECTOR
NEUTRINO PRODUCTION
UNDERGROUND PARTICLE DETECTOR
EXISTING LABS
PIP-II PROTON ACCELERATOR

DUNE at LBNF

DUNE Computing, a formal Computing Consortium was formed because of this experiment being collaborative and global. A lot of computational challenges that are faced, are faced by similar issues, therefore having a global community allows for maximizations on the use of common tools for data movement and storage, job control and monitoring, accounting, and authentication [2]. As more data is collected each day, DUNE computing works towards growing and evolving, therefore more advancements and development of tools are made to how this data is not only collected but analyzed as well.

1.2 Motivation

Individuals analyzing data from ProtoDUNE-II are currently using workflows that are contained in one or more scripts or ROOT macros. A prototype EAF containerized environment is used to adapt these workflows to run on the EAF. When adapted to run on the EAF, similar results will be achieved in a faster way using Python and datal science tools.

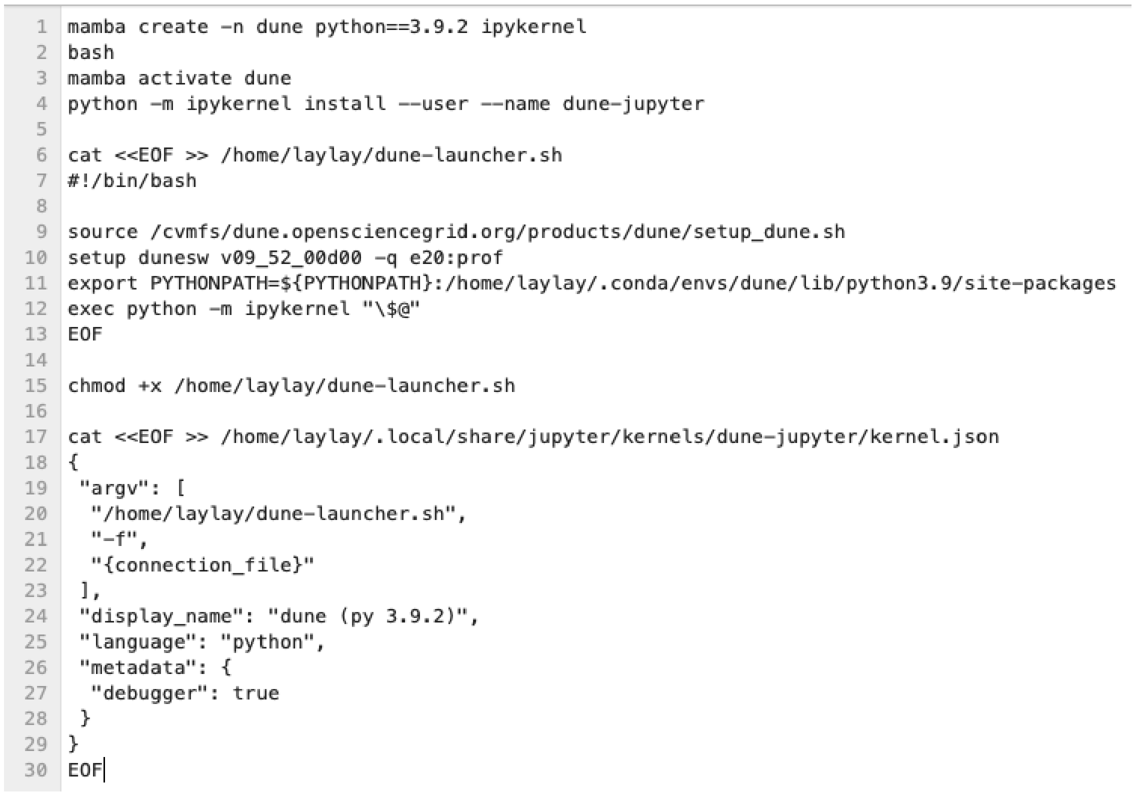
2 Methods

2.1 Elastic Analysis Facility

An Elastic Analysis Facility (EAF) is an infrastructure and services that provide integrated data, software, and computational recourse. This data, software and recourses are used to execute one or more elements of analysis workflow. These resources provided by this infrastructure and service is shared among the members of a virtual organizations. The elastic in Elastic Analysis Facility means that the analysis being done is structural and is assumed that the material being modeled is linear-elastic. EAF's goal is to provide various environments and customizations making it a user-oriented analysis facility. It also supports low latency applications which mean that it can process a large amount of data message with minimal delay which is good for DUNE and ProtoDUNE-II as it produces a lot of data. Lastly, the EAF can be documented and packaged to be deployed elsewhere.

2.2 Setting up the environment

The prototype Elastic Analysis Facility containerized environment used during this project was a notebook-based environment. The Python 3.9.2 kernel was manually created on the EAF using a script from the providers of the Elastic Analysis Facility. Mamba, a reimplementation of the conda package manager is a CLI tool that helps with managing coda s environment. This tool was used to create and activate the new environment and then mamba install was used to install the python packages that were needed. The usual DUNE software setup commands are then used as well as chmod to change the access permission as seen below.



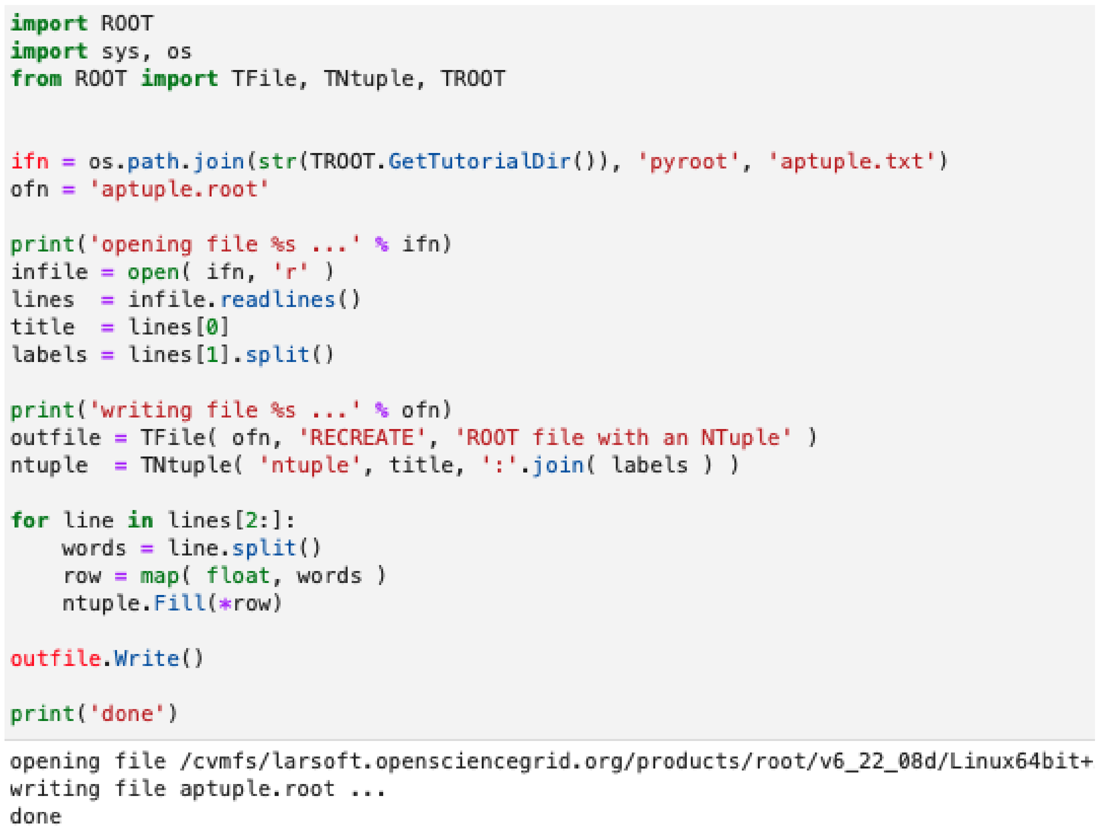
```
mamba create -n dune python==3.9.2 ipykernel
bash
mamba activate dune
python -m ipykernel install --user --name dune-jupyter

cat <<EOF >> /home/laylay/dune-launcher.sh
#!/bin/bash

source /cvmfs/dune.opensciencegrid.org/products/dune/setup_dune.sh
setup dunesw v09_52_00d00 -q e20:prof
export PYTHONPATH=${PYTHONPATH}:/home/laylay/.conda/envs/dune/lib/python3.9/site-packages
exec python -m ipykernel "\$@"
EOF

chmod +x /home/laylay/dune-launcher.sh

cat <<EOF >> /home/laylay/.local/share/jupyter/kernels/dune-jupyter/kernel.json
{
 "argv": [
  "/home/laylay/dune-launcher.sh",
  "-f",
  "{connection_file}"
 ],
 "display_name": "dune (py 3.9.2)",
 "language": "python",
 "metadata": {
  "debugger": true
 }
}
EOF
```

Script used to create Python 3.9.2 Kernel

As mentioned previously, in the future users will not have manually implement this. The EAF providers are currently working on enabling easy user customization of the environment which will include of preamble scripts via the custom kernel.

2.3 Verifying the environment using PyROOT

The environment was verified using PyROOT tutorial, PyROOT allows the creation of binding between Python and C++ in a dynamic and automatic way. With PyROOT, you can access the full ROOT C++ functionality from Python while benefiting the performance of the ROOT C++ libraries [3]. This was essential as it was not only used to verify the environment but to familiarize with PyROOT as the old workflows are contained in ROOT macros. One of the tutorials that was used for verification below reads the “aptule.txt” file row by row, then creates the Ntuple row by row. Through these tutorials, the environment was verified to be working.



```
import ROOT
import sys, os
from ROOT import TFile, TNtuple, TROOT


ifn = os.path.join(str(TROOT.GetTutorialDir()), 'pyroot', 'aptuple.txt')
ofn = 'aptuple.root'

print('opening file %s ...' % ifn)
infile = open( ifn, 'r' )
lines  = infile.readlines()
title  = lines[0]
labels = lines[1].split()

print('writing file %s ...' % ofn)
outfile = TFile( ofn, 'RECREATE', 'ROOT file with an NTuple' )
ntuple  = TNtuple( 'ntuple', title, ':'.join( labels ) )

for line in lines[2:]:
    words = line.split()
    row = map( float, words )
    ntuple.Fill(*row)

outfile.Write()

print('done')
```

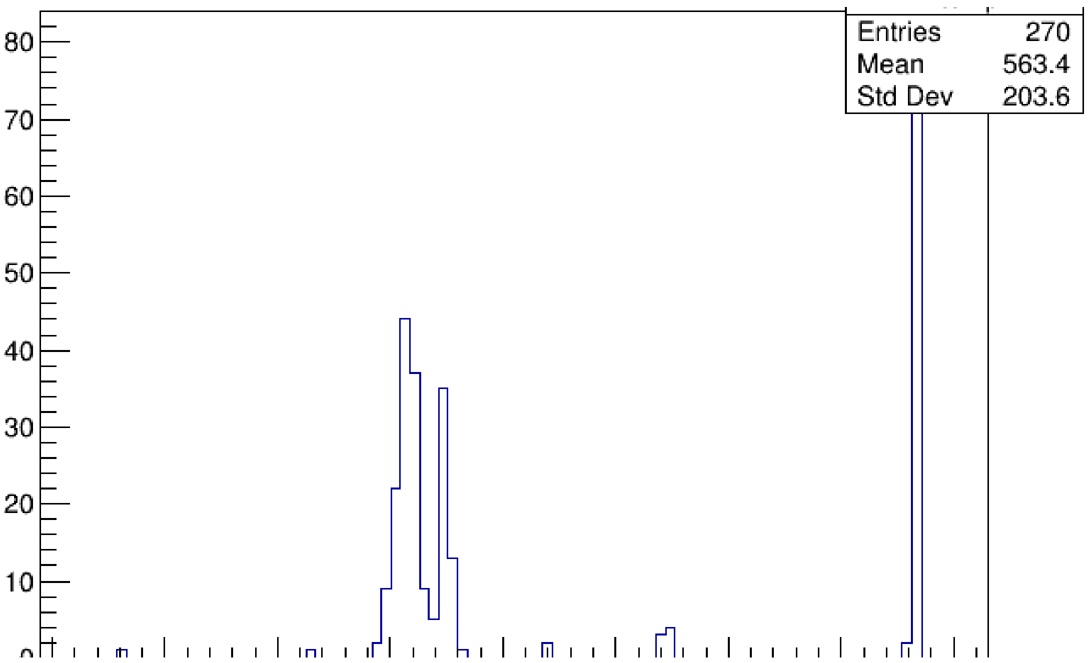
```
opening file /cvmfs/larsoft.opensciencegrid.org/products/root/v6_22_08d/Linux64bit+
writing file aptuple.root ...
done
```

PyROOT tutorial example [3]

Results

2.4 *Event Selection Analysis*

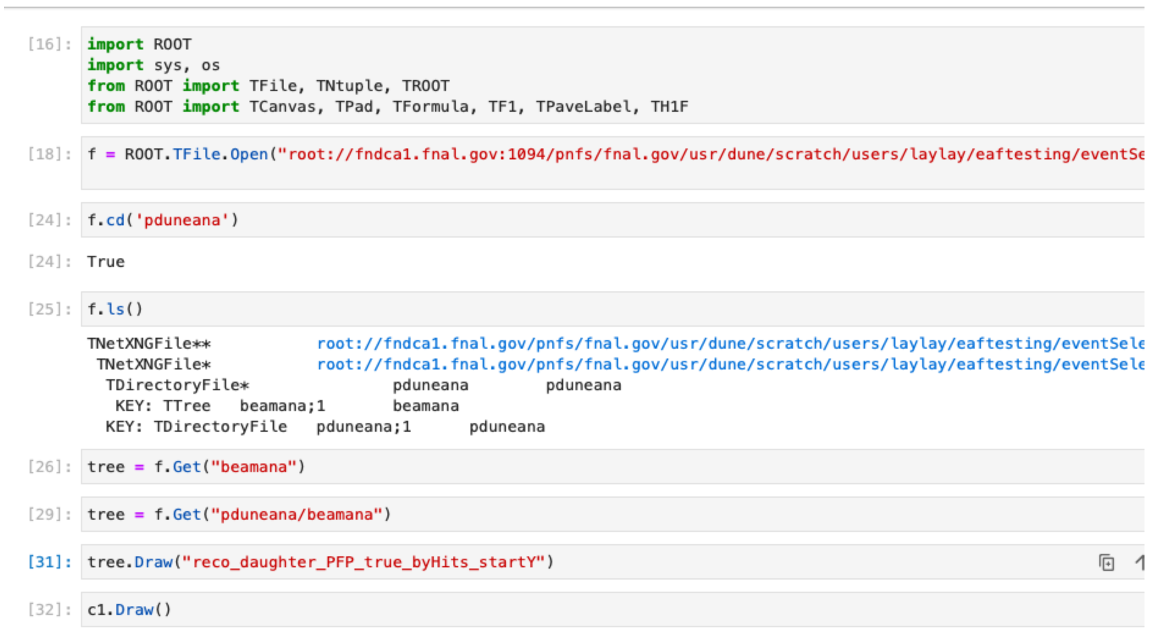
With the use of EventSelection for ProtoDune-SP, a Monte Carlo file and EvenSelection fcl file were used for input to get an output file and produce the plot below.



The plot above was produced using c++ and ROOT's R interface.

2.5 *Event Selection Analysis – Notebook*

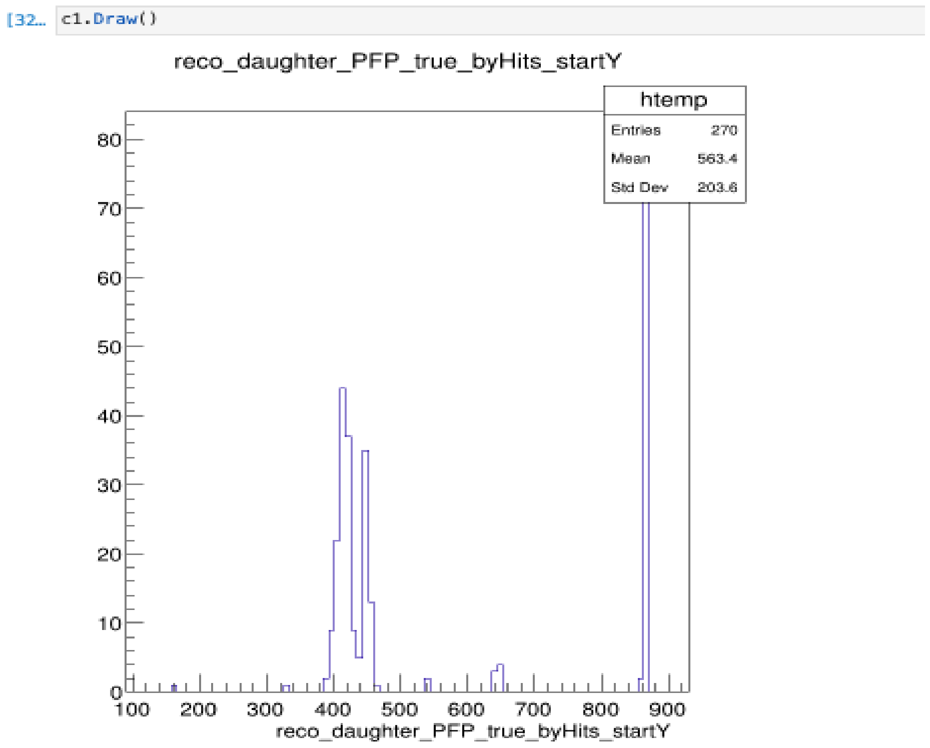
The code for the Event Selection for PDSP analysis is then ported to the notebook environment using Python to produce a similar plot.



```
[16]: import ROOT
      import sys, os
      from ROOT import TFile, TNtuple, TROOT
      from ROOT import TCanvas, TPad, TFormula, TF1, TPaveLabel, TH1F

[18]: f = ROOT.TFile.Open("root://fndca1.fnal.gov:1094/pnfs/fnal.gov/usr/dune/scratch/users/laylay/eaftesting/eventSe

[24]: f.cd('pduneana')

[24]: True

[25]: f.ls()

      TNetXNGFile**          root://fndca1.fnal.gov/pnfs/fnal.gov/usr/dune/scratch/users/laylay/eaftesting/eventSele
       TNetXNGFile*          root://fndca1.fnal.gov/pnfs/fnal.gov/usr/dune/scratch/users/laylay/eaftesting/eventSele
        TDirectoryFile*              pduneana        pduneana
         KEY: TTree    beamana;1       beamana
        KEY: TDirectoryFile   pduneana;1      pduneana

[26]: tree = f.Get("beamana")

[29]: tree = f.Get("pduneana/beamana")

[31]: tree.Draw("reco_daughter_PFP_true_byHits_startY")

[32]: c1.Draw()
```

The code above accesses the output file through xrootd to plot a specific entry

Using xrootd directly through the notebook and using Python calls to make the plot below.


```
[32.. c1.Draw()
```

The plot above was produced using Python on the notebook environment

3 Conclusion

Through this work, the platform was improved for future DUNE users on the EAF as various initial commissioning issues occurred. The next steps in the project are reproducing the full EventSelection analysis code on the environment in Python. As well as developing metrics to evaluate the performance of the workflow and creating relative documentation for other users analyzing data from ProtoDUNE-II can use. In addition, the environment should be distributed for testing and feedback within DUNE to see if any improvement needs to be made.

4 References

[1] "DUNE at LBNF." *Deep Underground Neutrino Experiment*, https://www.dunescience.org/.

[2] "Dune Computing Mission Statement." *DUNE*, https://computing wiki.dunescience.org/wiki/DUNE_Computing_Mission_Statement.

[3] ROOT. "Python Interface: PyRoot." *ROOT*, https://root.cern/manual/python/.