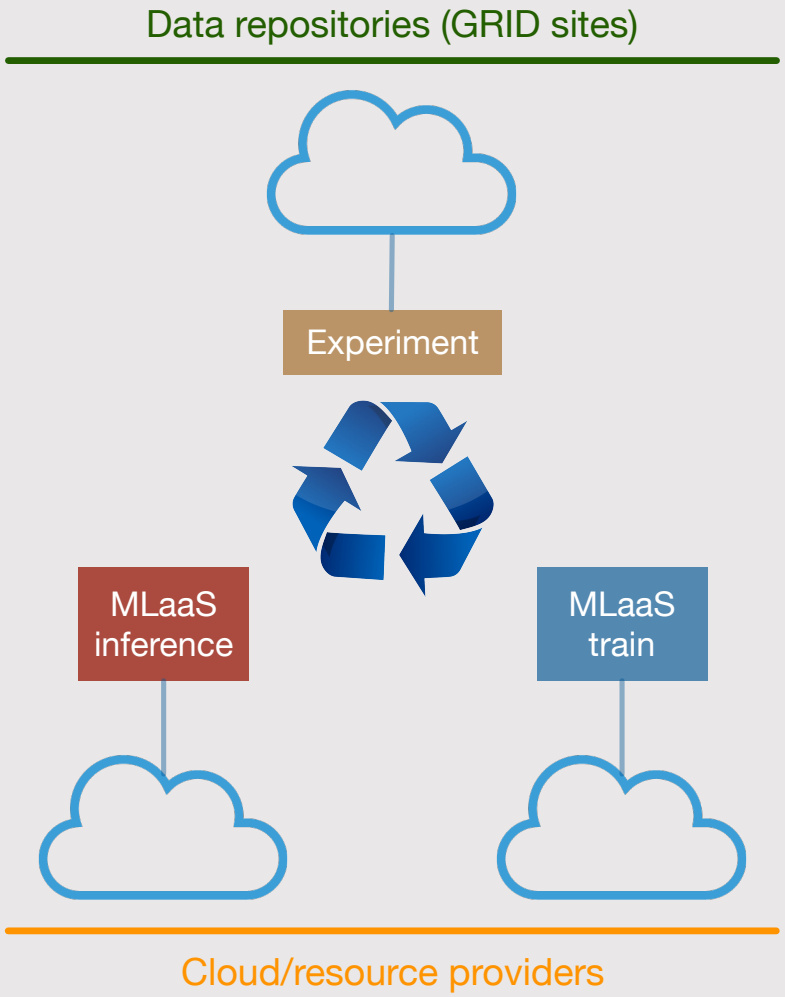
MLaaS4HEP: Machine Learning as a Service for HEP

Luca Giommi[1] (luca.giommi@cern.ch),
Valentin Kuznetsov[2], Daniele Bonacorsi[1], Daniele Spiga[3], Mattia Paladino[1]
[1] University of Bologna and INFN Bologna, Italy
[2] Cornell University, USA
[3]INFN Perugia, Italy

MLaaS for HEP

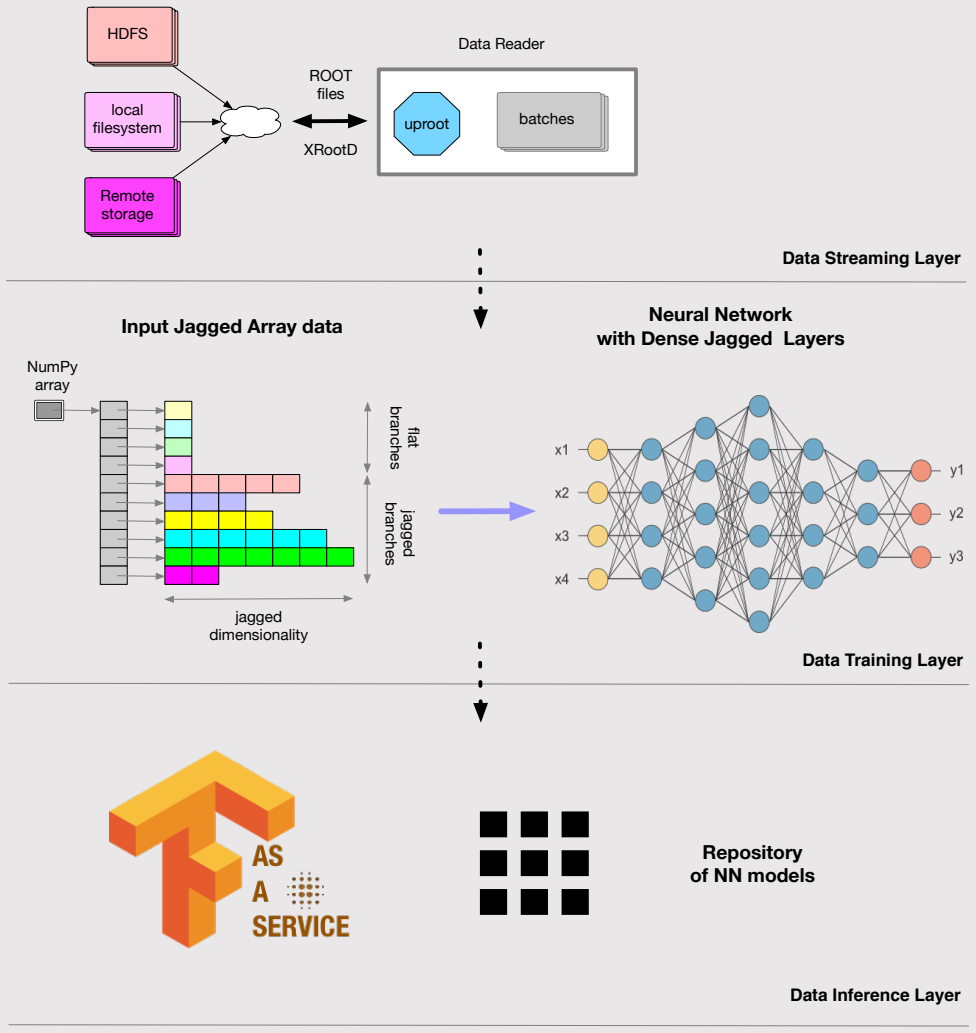
MLaaS for HEP aims at providing the following:

- **natively read HEP data**, e.g. be able to read ROOT files of arbitrary size from local or remote distributed data-sources via XrootD
- **use heterogeneous resources** both for training and inference, like local CPU, GPUs, farms, cloud resources, etc.
- **use different ML libs and frameworks** (Keras, TF, PyTorch, etc.)
- **serve pre-trained HEP models**, like a models repository, and access it easily from any place, any code, any framework.



MLaaS4HEP R&D

- **Data Streaming Layer** is responsible for local and remote data access of HEP ROOT files (based on the uproot library)
- **Data Training Layer** is responsible for feeding HEP ROOT data into existing ML frameworks
- **Data Inference Layer (TFaaS)** provides access to pre-trained TF models via HTTP protocol
  - A TFaaS server hosted by CERN is online: https://cms-tfaas.cern.ch/



Data streaming and training tools: github.com/vkuznet/MLaaS4HEP
Data inference tool: github.com/vkuznet/TFaaS
Paper published in the CSBS journal: doi.org/10.1007/s41781-021-00061-3

```
./workflow.py --files=files.txt --labels=labels.txt --model=model.py --params=params.json --preproc=preproc.json
```

MLaaS workflow | Input ROOT files | Labels of ROOT files | User model | MLaaS parameters | Preprocessing operations (Optional)

Keras model (model.py)

```
from tensorflow import keras
from keras.models import Sequential
from keras.layers import Dense, Dropout

def model(idim):
    "Simple Keras model for testing purposes"
    ml_model = Sequential([Dense(128,
activation='relu',input_shape=(idim,)),
                           Dropout(0.5),
                           Dense(64, activation='relu'),
                           Dropout(0.5),
                           Dense(1, activation='sigmoid')])
    ml_model.compile(optimizer=keras.optimizers.Adam(lr=1e-3),
                     loss=keras.losses.BinaryCrossentropy(),
                     keras.metrics.AUC(name='auc')])
```

MLaaS4HEP parameters (params.json)

```
{
    "nevts": 3000,
    "shuffle": true,
    "chunk_size": 1000,
    "epochs": 3,
    "batch_size": 100,
    "identifier": "",
    "branch": "boostedAk8/events",
    "selected_branches":"",
    "exclude_branches": "",
    "hist": "pdfs",
    "redirector": "root://xrootd.ba.infn.it",
    "verbose": 1
  }
```

Input ROOT files (files.txt)

```
PATH/flatTree_ttHJetTobb_M125_13TeV_amcatnloFXFX_madspin_pythia8.root
PATH/flatTree_TT_TuneCUETP8M2T4_13TeV-powheg-pythia8.root
```

Labels of ROOT files (labels.txt)

```
1
0
```

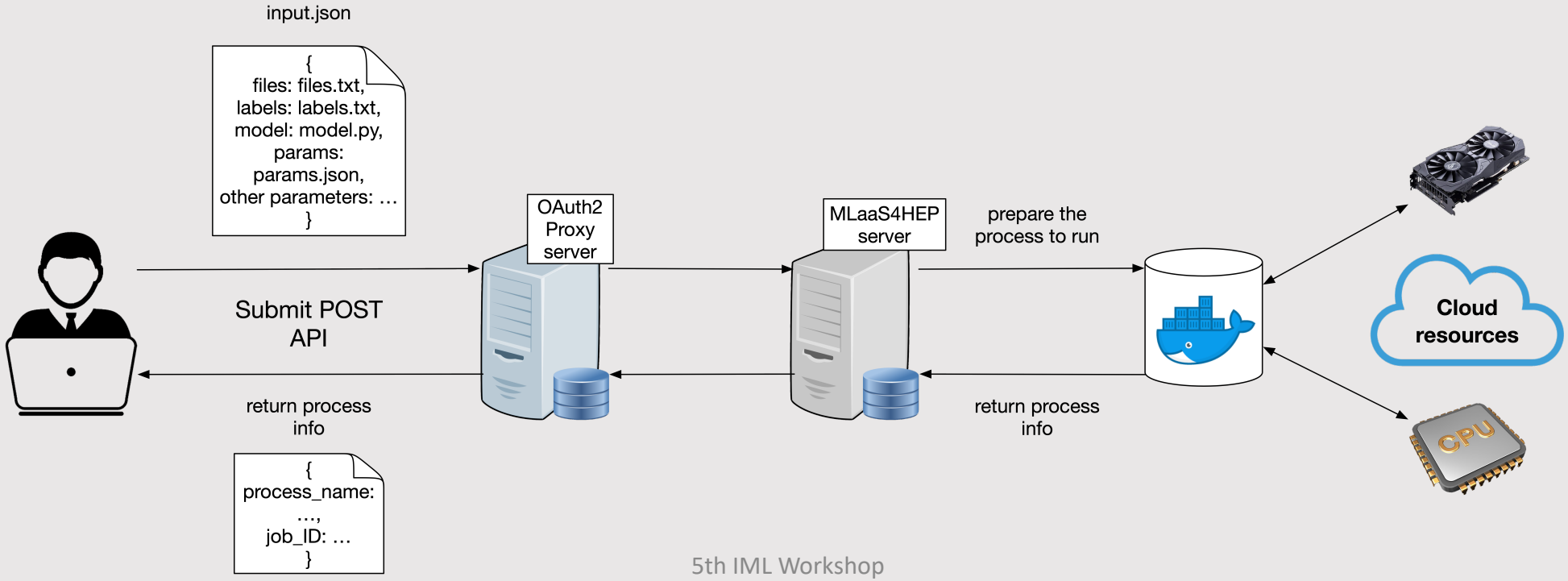
Preprocessing operations

- The MLaaS4HEP code has been updated to support uproot4 and to allow users to perform pre-processing operations on the input ROOT data.
- The migration to the updated version of uproot allowed to create new branches and to apply cuts, both on new and on existing branches.
- For more details, see the documentation here.

```
{
 "new_branch": {

    "log_partonE": {
       "def": "log(partonE)",
       "type": "jagged",
       "cut_1": ["log_partonE<6.31", "any"],
       "cut_2": ["log_partonE>5.85", "all"],
       "remove": "False",
       "keys_to_remove": ["partonE"]},

    "nJets_square": {
       "def": "nJets**2",
       "type": "flat",
       "cut": "1<=nJets_square<=16",
       "remove": "False",
       "keys_to_remove": ["nJets"]}},

 "flat_cut": {
    "nLeptons": {
    "cut": "0<=nLeptons<=2",
    "remove": "False"}},

 "jagged_cut": {
    "partonPt": {
    "cut": ["partonPt>200", "all"],
    "remove": "False"}}}
```

Towards MLaaS4HEP cloudification

- The goal is to create a cloud service that could use cloud resources and could be integrated into the INFN Cloud portfolio of services.
- We built a MLaaS4HEP server with some basic APIs using the (Python-based) Flask framework: https://github.com/lgiommi/MLaaS4HEP_server
- We integrated an oauth2-proxy to manage the authentication and authorization phases for the access to MLaaS4HEP server APIs



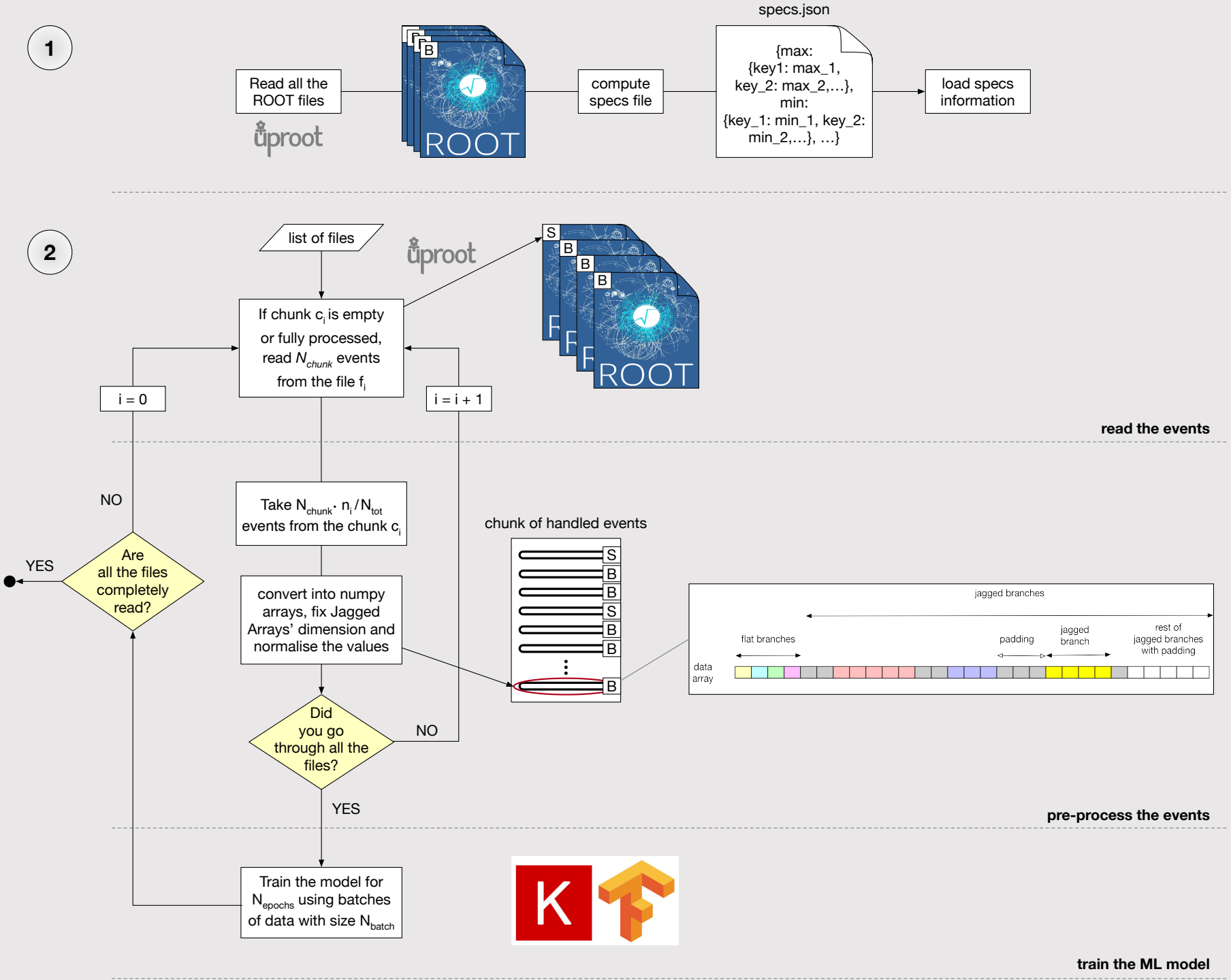
MLaaS4HEP publications

- V. Kuznetsov, L. Giommi, D. Bonacorsi, *MLaaS4HEP: Machine Learning as a Service for HEP*. Comput Softw Big Sci 5, 17. DOI: 10.1007/s41781-021-00061-3
- L. Giommi, D. Spiga, V. Kuznetsov, D. Bonacorsi, M. Paladino, *Cloud native approach for Machine Learning as a Service for High Energy Physics*. Submitted to the proceedings (PoS ISGC2022, 012)
- L. Giommi, V. Kuznetsov, D. Bonacorsi, D. Spiga, *Machine Learning as a Service for High Energy Physics on heterogeneous computing resources*. PoS ISGC2021, 019. DOI: 10.22323/1.378.0019
- L. Giommi, D. Bonacorsi, V. Kuznetsov, *Prototype of Machine Learning "as a Service" for CMS Physics in Signal vs Background discrimination*. PoS LHCP2018, 093. DOI: 10.22323/1.321.0093

Thanks for the attention

luca.giommi@cern.ch

Backup slides


1
Read all the ROOT files
uproot
B
ROOT
compute specs file
specs.json
{max:
{key1: max_1,
key_2: max_2,…},
min:
{key_1: min_1, key_2: min_2,…}, …}
load specs information
2
list of files
uproot
If chunk c_i is empty or fully processed, read N_{chunk} events from the file f_i
S
B
B
B
ROOT
i = 0
i = i + 1
read the events
NO
Take $N_{chunk} \cdot n_i / N_{tot}$ events from the chunk c_i
chunk of handled events
YES
Are all the files completely read?
convert into numpy arrays, fix Jagged Arrays' dimension and normalise the values
S
B
B
S
B
B
jagged branches
flat branches
padding
jagged branch
rest of jagged branches with padding
data array
Did you go through all the files?
NO
YES
pre-processing the events
Train the model for N_{epochs} using batches of data with size N_{batch}
K
train the ML model

MLaaS parameters

Read remote root files

Write and load the specs

```
./workflow.p  --files=files.txt --labels=  bels.  xt –model=model.py --params=params.json
DataGenerato . <MLaaS4HEP.generator.Roo ataGen  ator object at 0x7f0cb58d7fd0> [29/Jun/2020:17:53:  4] 1593445994.0
model parameters: {"nevts": 30000, "s  ffle": t ue, "chunk_size": 10000, "epochs": 2, "batch_size"  500, "identifier": ["runNo", "evtNo", "lumi"],
"branch": "boostedAk8/events", "sel  ted_branc  s": "", "exclude_branches": "", "hist": "pdfs", "r director": "root://xrootd.ba.infn.it", "verbose": 1}


Reading root://xrootd.ba.infn.it//PATH_FILES/ latTree_ttHJetTobb_M125_13TeV_amcatnloFXFX_madspin  ythia8.root
# 10000 entries, 77 branches, 9.522203445434  7 MB, 1.0169336795806885 sec, 9.36364252323795 MB/s  c, 9.833482950553169 kHz
# 10000 entries, 77 branches, 9.53391551971  55 MB, 1.2977769374847412 sec, 7.346343770133804 M  sec, 7.705484441248654 kHz
# 10000 entries, 77 branches, 9.53866767883  08 MB, 1.4104814529418945 sec, 6.7627033726234735  B/sec, 7.089777734505208 kHz
--- first pass: 948348 events, (22-flat, 5  jagged) branches, 328 attrs
<MLaaS4HEP.reader.RootDataReader object at  0x7f840dbf4d50> init is complete in 4.8529925346374  1 sec


Reading root://xrootd.ba.infn.it//PATH_FILES/flatTree_TT_TuneCUETP8M2T4_13TeV-powheg-pythia8  oot
# 10000 entries, 77 branches, 8.875920295715332 MB, 0.9596493244171143 sec, 9.24912889518941  MB/sec, 10.42047313071777 kHz
# 10000 entries, 77 branches, 8.868906021118164 MB, 1.2938923835754395 sec, 6.8544386949790  MB/sec, 7.728618026459661 kHz
# 10000 entries, 77 branches, 8.869449615478516 MB, 1.1267895698547363 sec, 7.8714338974773  9 MB/sec, 8.874771534572496 kHz
--- first pass: 1003980 events, (22-flat, 52-jagged) branches, 312 attrs
<MLaaS4HEP.reader.RootDataReader object at 0x7f8410e15f90> init is complete in 4.535124778  47559 sec


write global-specs.json
load specs from global-specs.json for root://xrootd.ba.infn.it//$PATH_FILES/flatTree_ttHJetTobb_M125_13TeV_amcatnloFXFX_madspin_pythia8.root
load specs from global-specs.json for root://xrootd.ba.infn.it//$PATH_FILES/flatTree_TT_TuneCUETP8M2T4_13TeV-powheg-pythia8.root
init RootDataGenerator in 11.186564683914185 sec
```

```
label 1, file <flatTree_ttHJetTobb_M125_13TeV_amcatnloFXFX_madspin_pythia8.root>, going to read 4858 events
read chunk [0:4857] from /$PATH_FILES/flatTree_ttHJetTobb_M125_13TeV_amcatnloFXFX_madspin_pythia8.root
# 10000 entries, 77 branches, 9.52220344543457 MB, 1.3816642761230469 sec, 6.891835889507034 MB/sec, 7.237648228164387 kHz
total read 4858 evts from /$PATH_FILES/flatTree_ttHJetTobb_M125_13TeV_amcatnloFXFX_madspin_pythia8.root


label 0, file <flatTree_TT_TuneCUETP8M2T4_13TeV-powheg-pythia8.root>, going to read 5142 events
read chunk [4858:9999] from /$PATH_FILES/flatTree_TT_TuneCUETP8M2T4_13TeV-powheg-pythia8.root
# 10000 entries, 77 branches, 8.875920295715332 MB, 1.7170112133026123 sec, 5.169401473297779 MB/sec, 5.8240737873606205 kHz
total read 5142 evts from /$PATH_FILES/flatTree_TT_TuneCUETP8M2T4_13TeV-powheg-pythia8.root
```

Create the chunk

MLaaS4HEP performance results

- Based on the resource we used and if the ROOT files were local or remote, we obtained:
  - **specs computing phase (chunk size = 100k events)**
    - Event throughput: 8.4k – 13.7k evts/s
    - Total time using all the 28.5M events: 35 – 57 min
  - **chunks creation in the training phase (chunk size = 100k events)**
    - Event throughput: 1.1k – 1.2k evts/s
    - Total time using all the 28.5M events: 6.5 – 7.5 hrs

WITH UPROOT3 AND W/O PREPROCESSING OPERATIONS

- The time to train the ML model is not included in the performance. It is independent from the MLaaS4HEP framework but depends on the underlying ML framework, the complexity of the used ML model, and the available hardware resources.

- We estimate that projecting these results for datasets at the **TB scale** and using the same hardware resources, the specs computing phase will take O(100) hours and the training phase will take O(1k) hours (plus the time required to train the ML model).
  - Further optimization of the MLaaS4HEP pipeline will be required to process TB or PB scale datasets and it may involve parallelization of I/O, distributed ML training, etc.

Running MLaaS4HEP

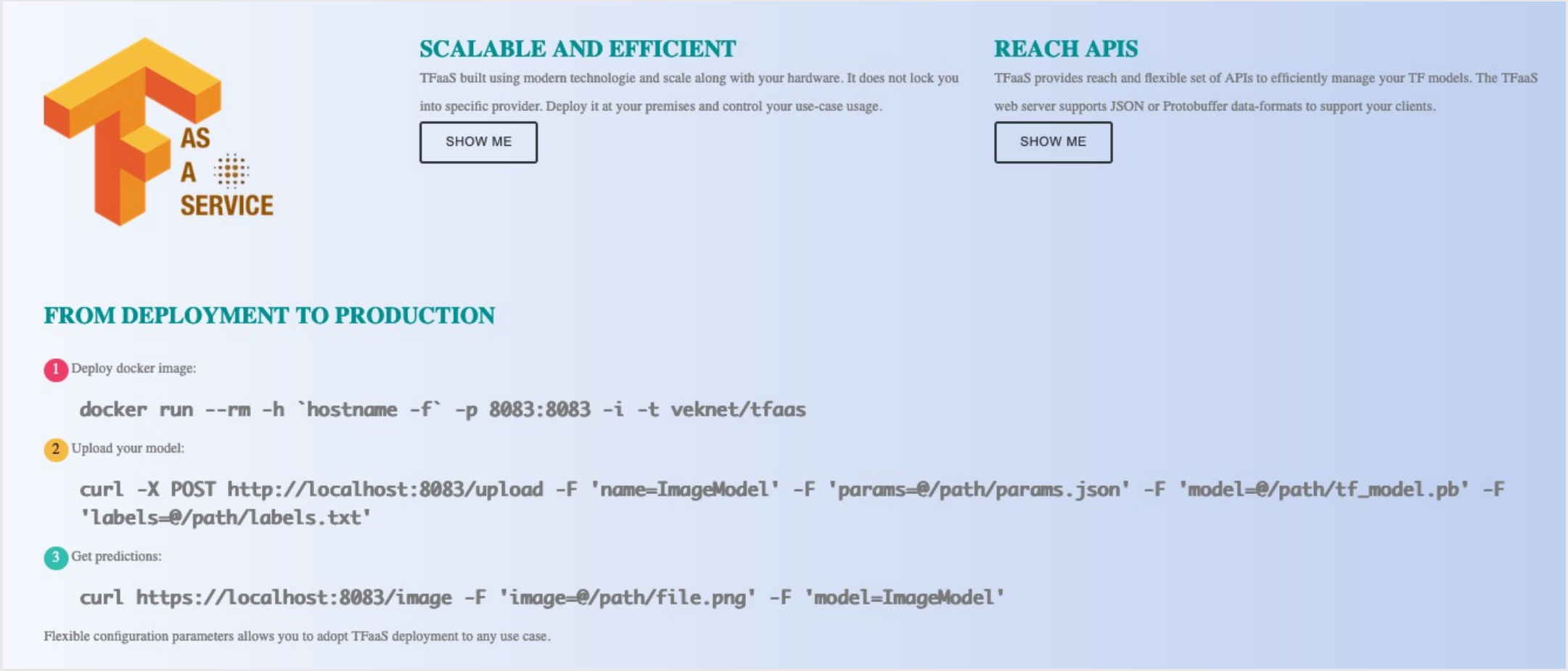
DEMO

```
(uproot4) mattia@MBP-di-Mattia MLaaS4HEP % ./workflow.py --files=files.txt --labels=labels.txt --model=keras_model.py --params=params.j
son --preproc=preproc.json

Parameters: {"nevts": 3000, "shuffle": true, "chunk_size": 1000, "epochs": 3, "batch_size": 100, "identifier": "", "branch": "boostedAk
8/events", "selected_branches": "", "exclude_branches": "", "hist": "pdfs", "redirector": "", "verbose": 1}


# Cut(s) provided on flat branch(es):
0 <= nLeptons <= 2
# Cut(s) provided on jagged branch(es):
partonPt > 200 using "all"
partonPt < 600 using "any"
# Definition(s) of new branch(es):
log_partonE: log(partonE)
# Cut(s) on new branch(es):
log_partonE<6.3 using "any"


Reading flatTree_ttHJetTobb_M125_13TeV_amcatnloFXFX_madspin_pythia8.root
# 1000 entries, 249 events after cut, 77 branches, 1.098 MB, 0.325 sec, 3.377 MB/sec, 3.074 kHz
# 1000 entries, 219 events after cut, 77 branches, 1.081 MB, 0.302 sec, 3.577 MB/sec, 3.31 kHz
# 1000 entries, 202 events after cut, 77 branches, 1.073 MB, 0.289 sec, 3.709 MB/sec, 3.457 kHz
# 1000 entries, 222 events after cut, 77 branches, 1.086 MB, 0.297 sec, 3.654 MB/sec, 3.366 kHz
# 1000 entries, 260 events after cut, 77 branches, 1.104 MB, 0.294 sec, 3.752 MB/sec, 3.398 kHz
# 1000 entries, 253 events after cut, 77 branches, 1.102 MB, 0.288 sec, 3.826 MB/sec, 3.471 kHz
# 1000 entries, 233 events after cut, 77 branches, 1.09 MB, 0.294 sec, 3.711 MB/sec, 3.403 kHz
# 1000 entries, 263 events after cut, 77 branches, 1.103 MB, 0.31 sec, 3.562 MB/sec, 3.23 kHz
# 1000 entries, 246 events after cut, 77 branches, 1.098 MB, 0.276 sec, 3.98 MB/sec, 3.626 kHz
# 1000 entries, 228 events after cut, 77 branches, 1.086 MB, 0.296 sec, 3.671 MB/sec, 3.379 kHz
# 1000 entries, 227 events after cut, 77 branches, 1.087 MB, 0.291 sec, 3.73 MB/sec, 3.432 kHz
```

https://www.dropbox.com/s/95rzem8ofhgy8ei/DEMO_CUT.mp4?dl=0

How to use TFaaS

DEMO



https://www.dropbox.com/s/f5907j23ba4jjms/Tfaas_video.mp4?dl=0

Current MLaaS4HEP server

- We built a MLaaS4HEP server with some basic APIs using the (Python-based) Flask framework: https://github.com/lgiommi/MLaaS4HEP_server

- Current APIs: submit, status_docker, logs
  - curl -H "Content-Type: application/json" -d @input.json http://localhost:8080/submit
  - curl http://localhost:8080/status_docker?process_name=luca_1
  - curl 'http://localhost:8080/logs?process_name=luca_1&log_file=logs.txt'

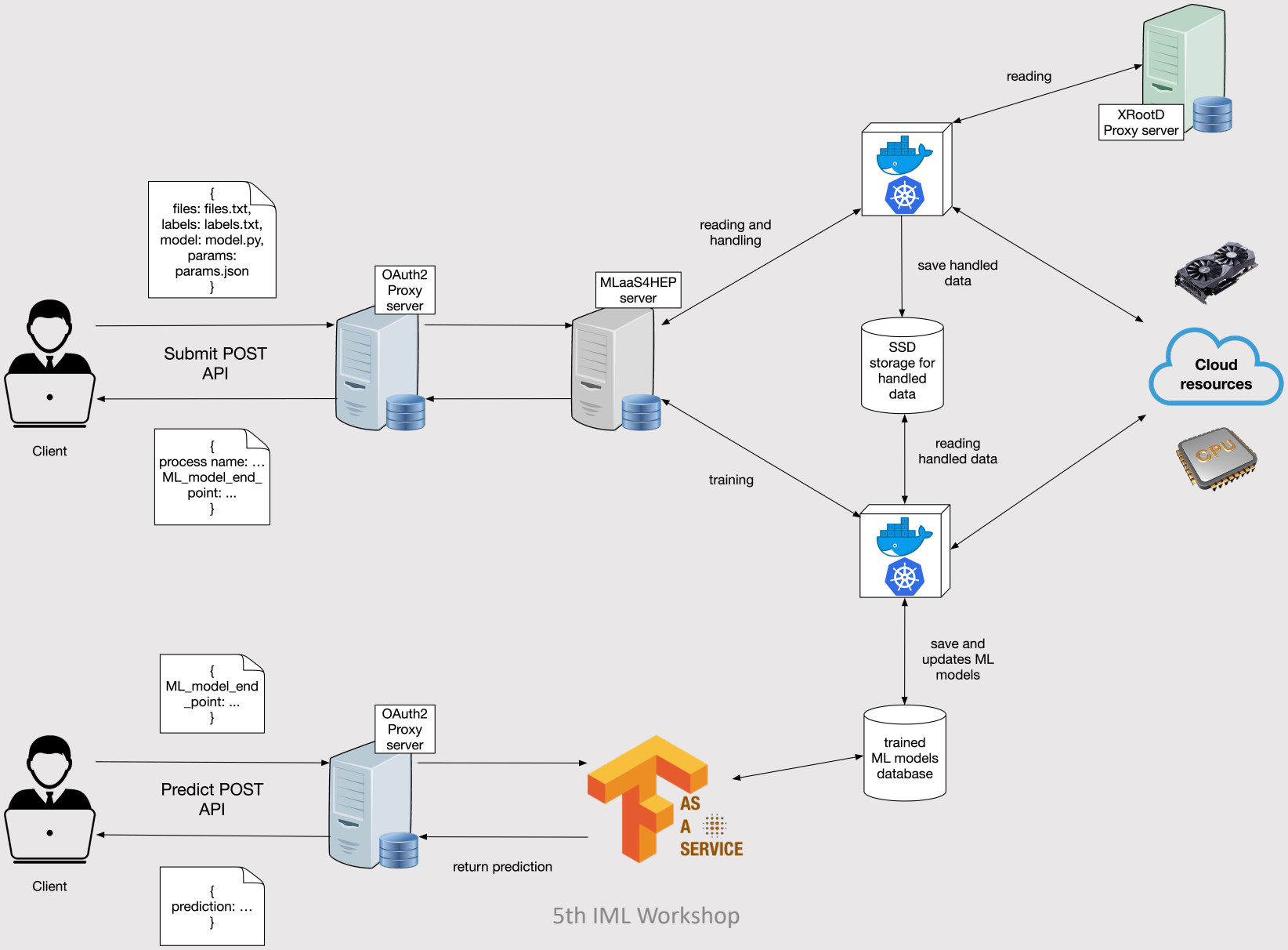
```
{
"name": "luca",
"device": "cpu",
"memory": "500m",
"cpus": 0.1,
"files": "files_test.txt",
"labels": "labels_test.txt",
"model": "keras_model.py",
"params": "params_test.json"
}
```

```
{
"process_name": "luca_1",
"job_id": 20547
}
```

Example of input.json file and output of a submit API

Integration of OAuth2 Proxy server

1) Register the client with the authorization server https://cms-auth.web.cern.ch/
2) Obtain a token for the client using oidc-agent
3) Prepare a configuration file for the oauth2-proxy server (also for the case of TLS connections)
4) Run the oauth2-proxy server using the prebuilt binary or the prebuilt docker image
5) Call MLaaS4HEP APIs passing the TOKEN and using the port where the proxy is running

```
curl -L -k -H 'Content-Type: application/json' -H "Authorization: Bearer ${TOKEN}" -d @input.json
http://localhost:4180/submit
```

Preliminary final architecture