Cloud native approach for Machine Learning as a Service for High Energy Physics

Luca Giommi[1] (luca.giommi@bo.infn.it),
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

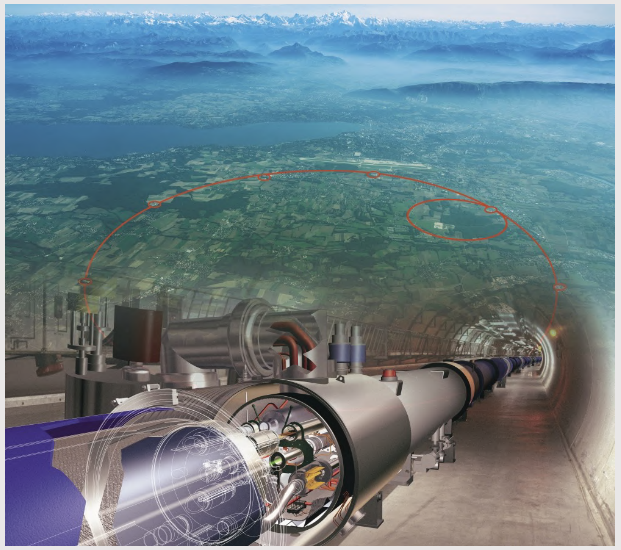
International Symposium on Grids & Clouds 2022, March 23th

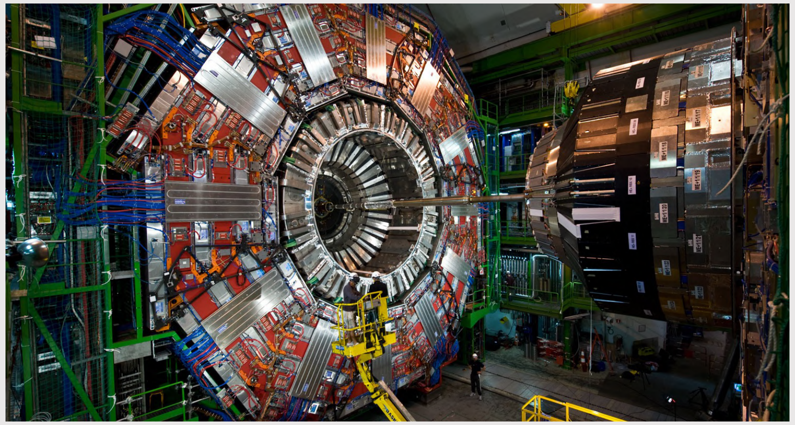
High Energy Physics at CERN

- The European Organization for Nuclear Research (CERN) was born in 1954 and it is based in the northwest suburb of Geneva on the Franco–Swiss border.

- At CERN is located the world's largest and most powerful particle accelerator called Large Hadron Collider (LHC).

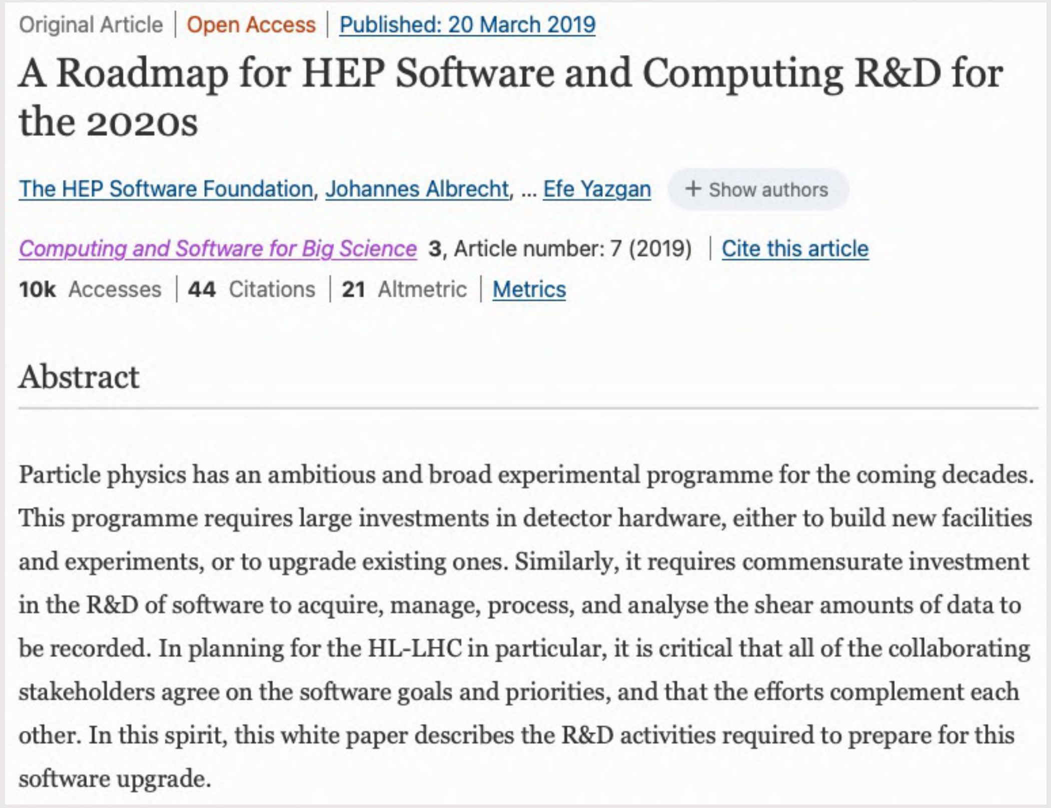
- Inside the accelerator, two high-energy particle beams travel at close to the speed of light before they are made to collide at four locations, corresponding to the positions of four particle detectors: ATLAS, CMS, ALICE and LHCb.





Why Machine Learning and ML as a Service?

- Machine Learning techniques in the HEP domain are ubiquitous, successfully used in many areas, and will play a significant role also in Run3 and High-Luminosity LHC upgrade.
- It would be useful a service that ease the usage of ML in HEP analysis
- Existing MLaaS solutions are many: offer many services and cover different use cases but they are not directly usable in HEP.
- Existing HEP R&D solutions don't cover the whole ML pipeline or they are not ''aaS'' solutions or they are diffult to generalize to other use cases.
- We propose a Machine Learning as a Service for HEP (MLaaS4HEP) solution as a product of R&D activities within the CMS experiment.


Original Article | Open Access | Published: 20 March 2019

A Roadmap for HEP Software and Computing R&D for the 2020s

The HEP Software Foundation, Johannes Albrecht, ... Efe Yazgan + Show authors

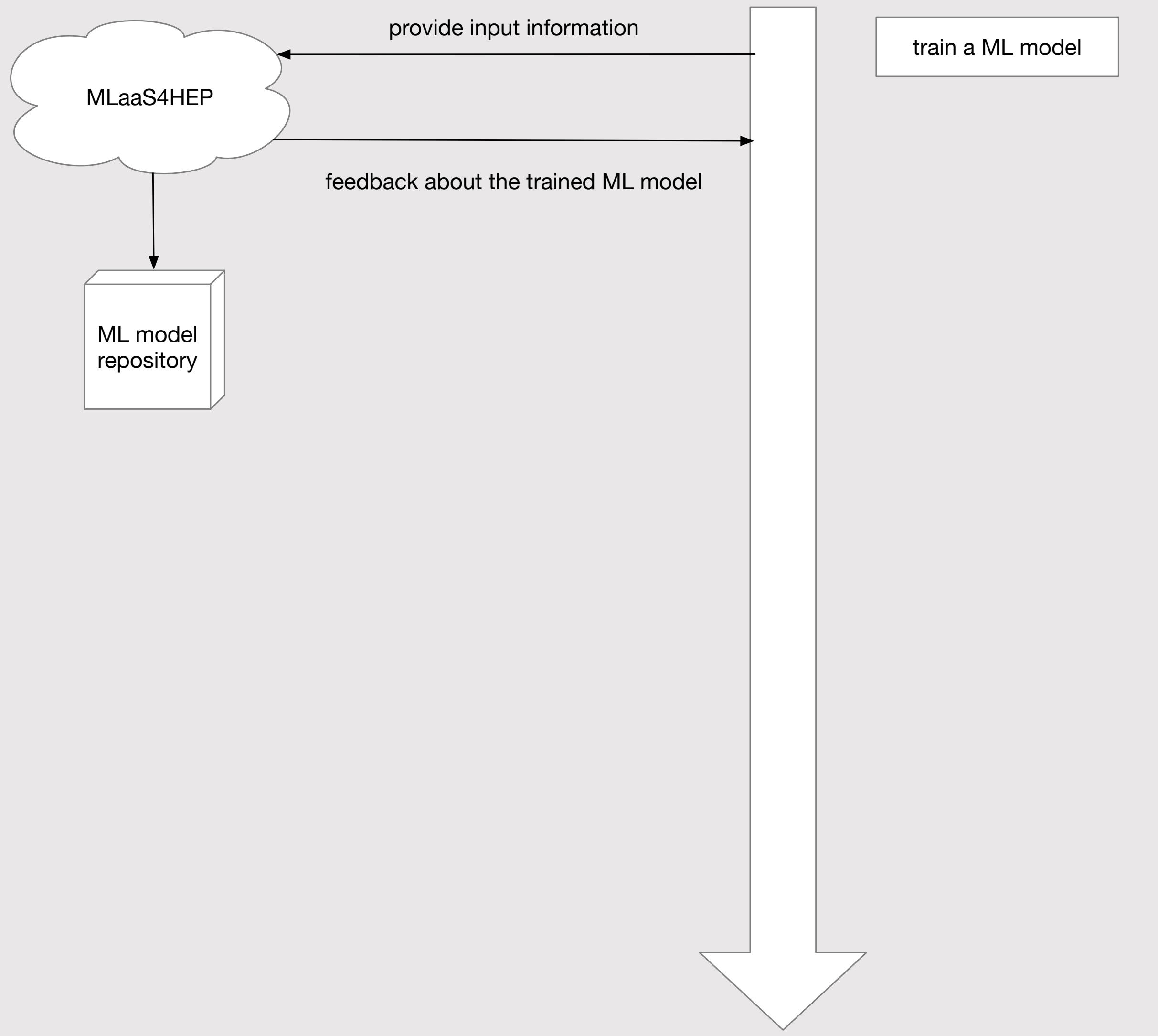
Computing and Software for Big Science **3**, Article number: 7 (2019) | Cite this article

10k Accesses | **44** Citations | **21** Altmetric | Metrics

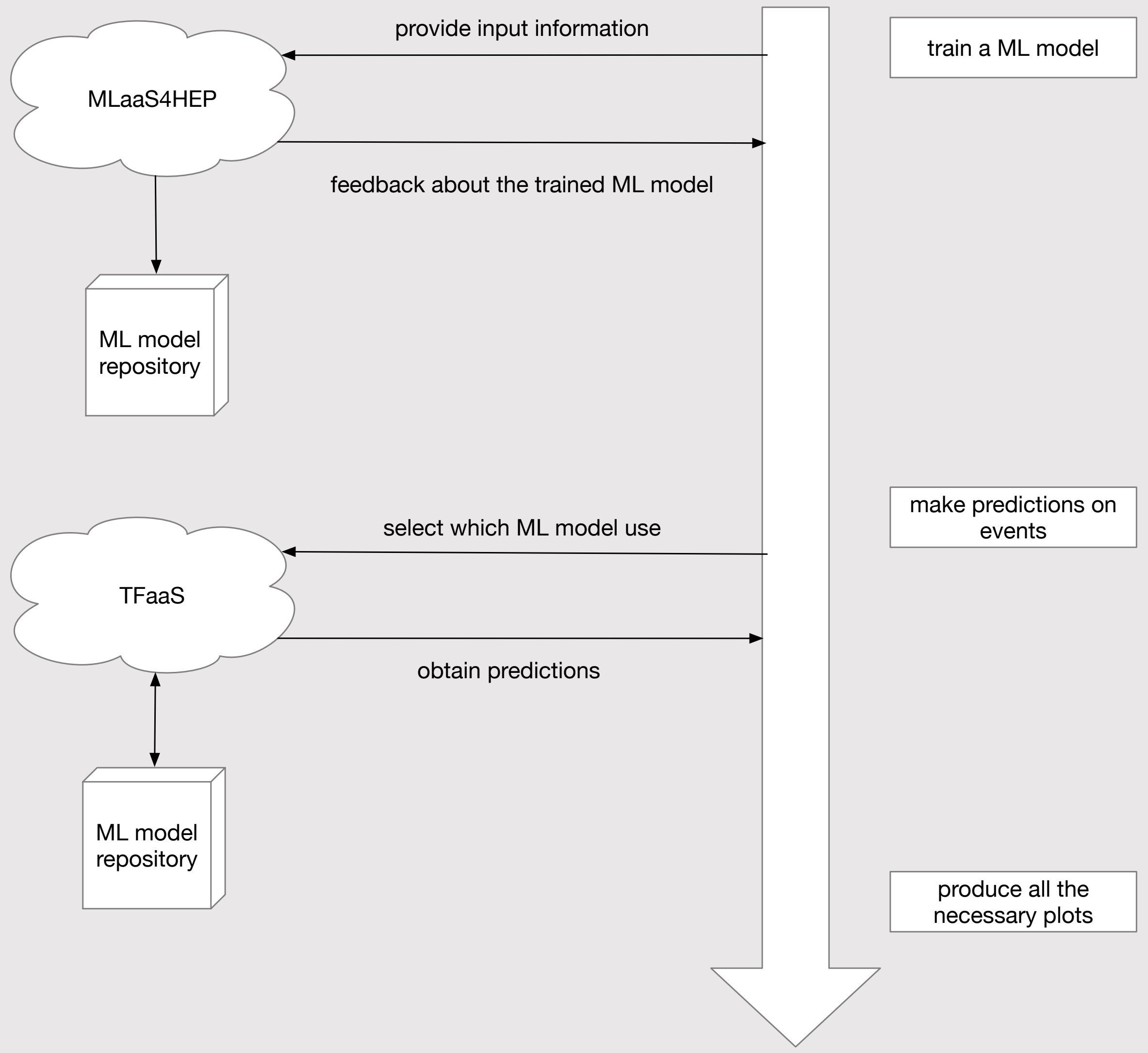
Abstract

Particle physics has an ambitious and broad experimental programme for the coming decades. This programme requires large investments in detector hardware, either to build new facilities and experiments, or to upgrade existing ones. Similarly, it requires commensurate investment in the R&D of software to acquire, manage, process, and analyse the shear amounts of data to be recorded. In planning for the HL-LHC in particular, it is critical that all of the collaborating stakeholders agree on the software goals and priorities, and that the efforts complement each other. In this spirit, this white paper describes the R&D activities required to prepare for this software upgrade.

Basic idea of the ML as a Service for HEP workflow


provide input information
train a ML model
MLaaS4HEP
feedback about the trained ML model
ML model repository

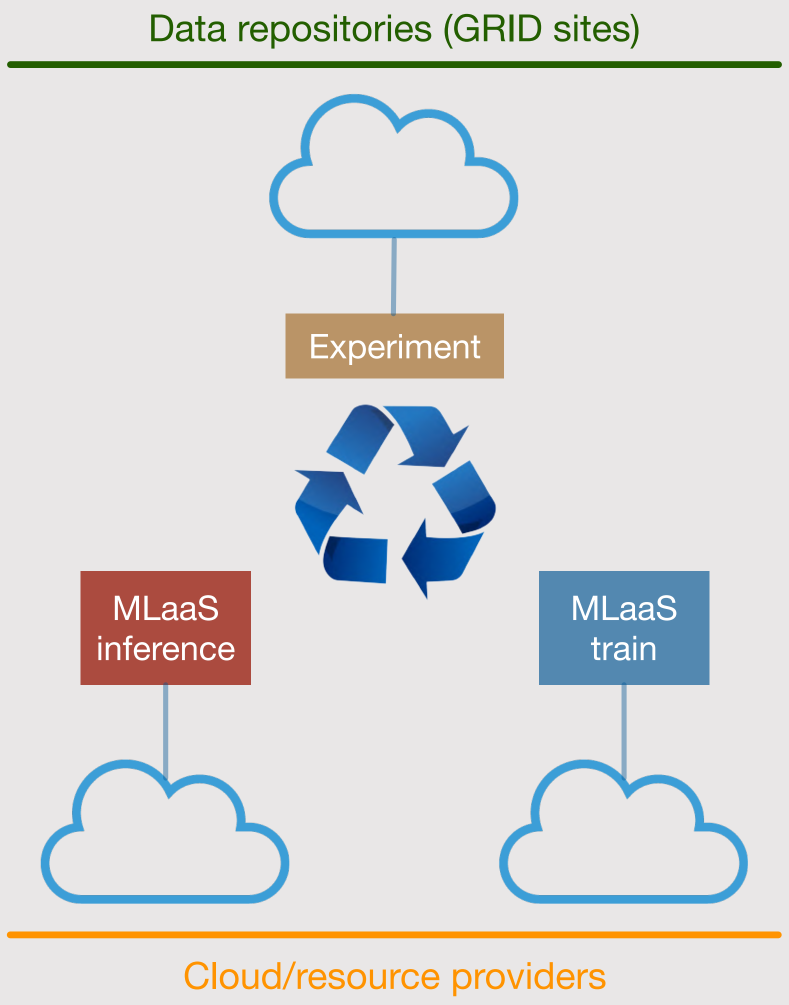
Basic idea of the ML as a Service for HEP workflow


MLaaS4HEP
provide input information
feedback about the trained ML model
ML model repository
train a ML model
TFaaS
select which ML model use
obtain predictions
ML model repository
make predictions on events
produce all the necessary plots

MLaaS for HEP

MLaaS for HEP aims at providing the following:

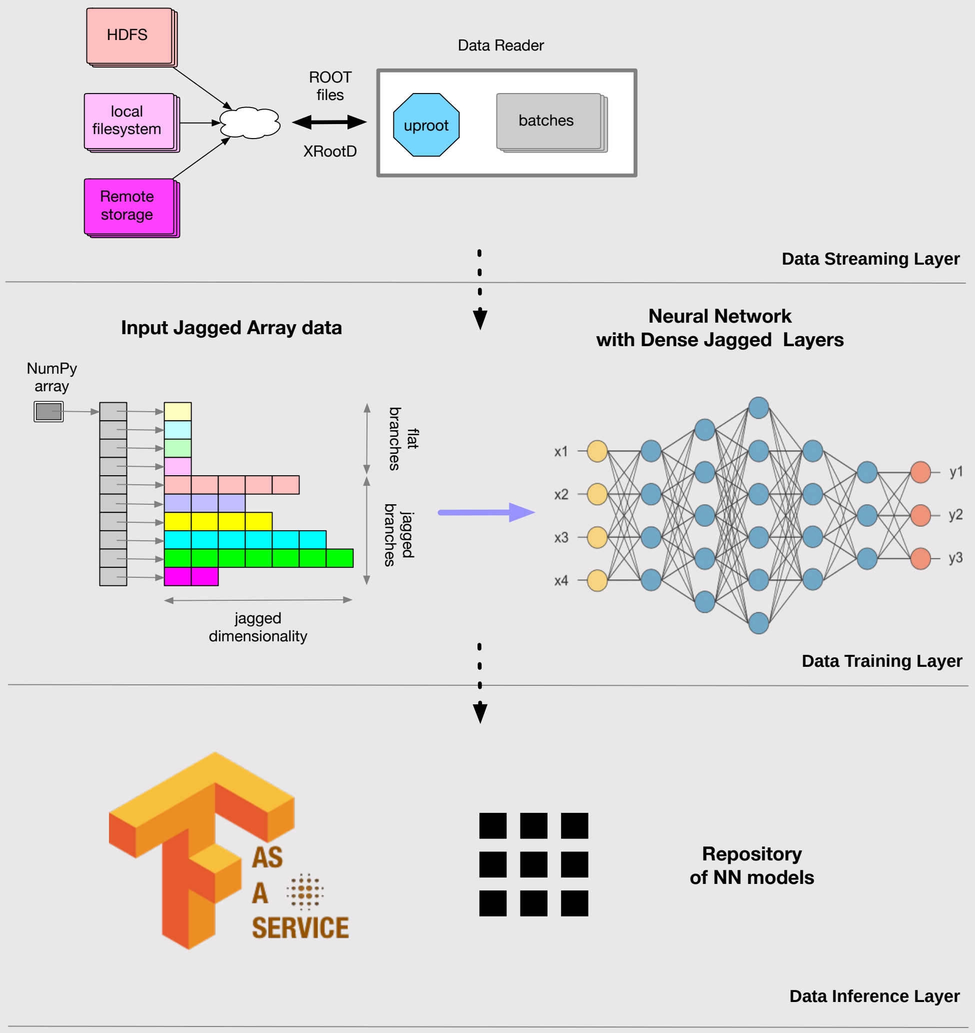
- **natively read HEP data**, e.g. be able to read ROOT files of arbitrary size from local or remote distributed data-sources via XrootD
- **use heterogeneous resources** both for training and inference, like local CPU, GPUs, farms, cloud resources, etc.
- **use different ML libs and frameworks** (Keras, 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
- **Data Training Layer** is responsible for feeding HEP ROOT data into existing ML frameworks
- **Data Inference Layer (TFaaS)** provides access to pre-trained HEP model for HEP users

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]



Data Streaming and Training Layers

- The development of the DIANA-HEP **uproot** library provides the ability to read ROOT data in Python, access them as NumPy arrays, and implements XrootD access to read remote files

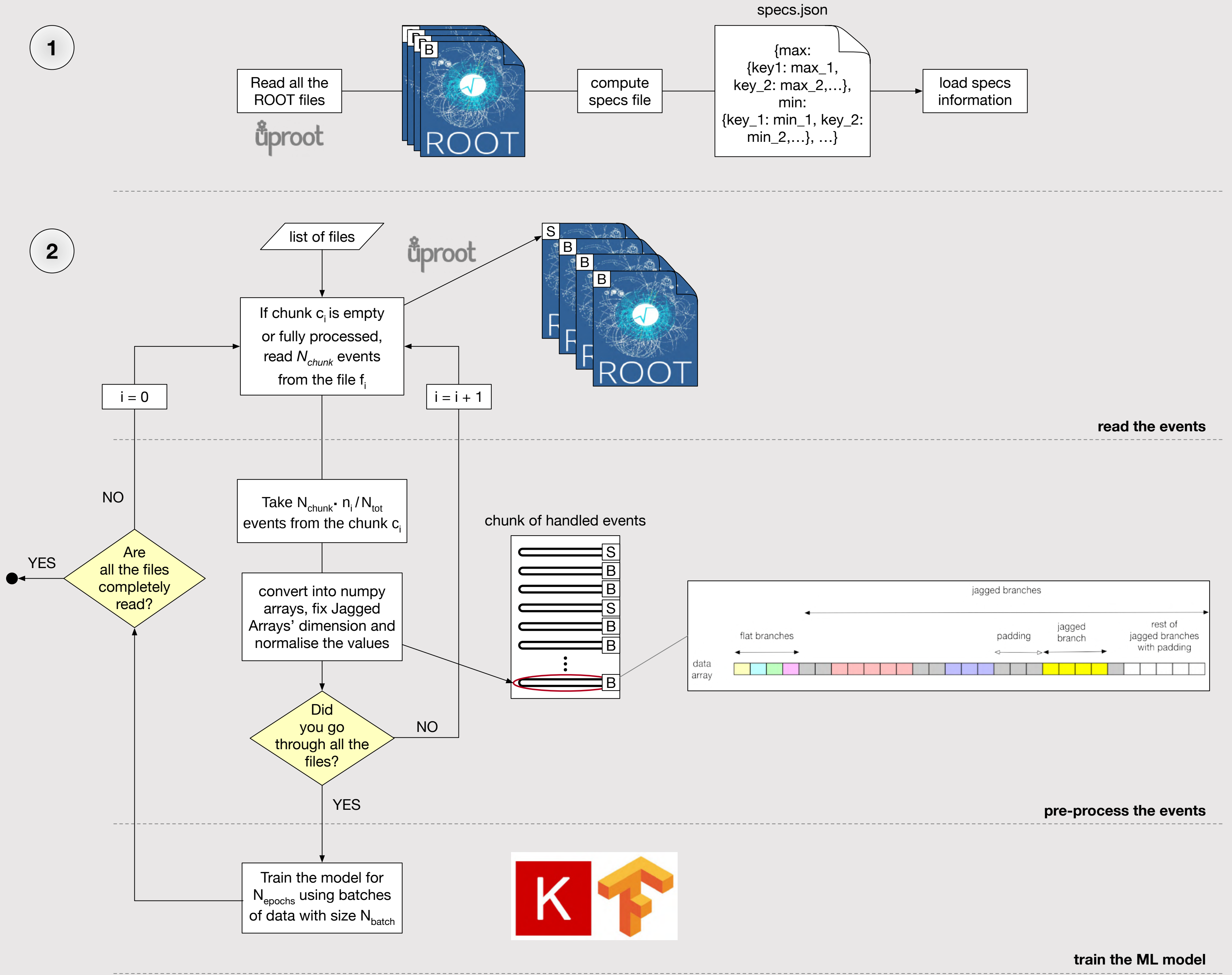
- MLaaS4HEP extends uproot library and provide APIs to feed data read from local and remote distributed ROOT files into existing ML frameworks

  - a <u>Python Generator</u> is created to read ROOT files and deliver them as chunks

    - such implementation provides efficient access to large datasets since it does not require loading the entire dataset into the RAM of the training node
    - normalization and pre-processing operations can be applied to the events

  - random reads from multiple files are supported, taking the right proportion of data from each file

- The non-flat ROOT branches are read and represented by uproot as Jagged/Awkward Arrays


1
Read all the ROOT files
uproot
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
i = 0
i = i + 1
read the events
NO
Take N_chunk · 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
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-process the events
Train the model for N_epochs using batches of data with size N_batch
K
train the ML model

```
./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)

```python
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')])
```

MLaaS parameters (params.json)

```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.

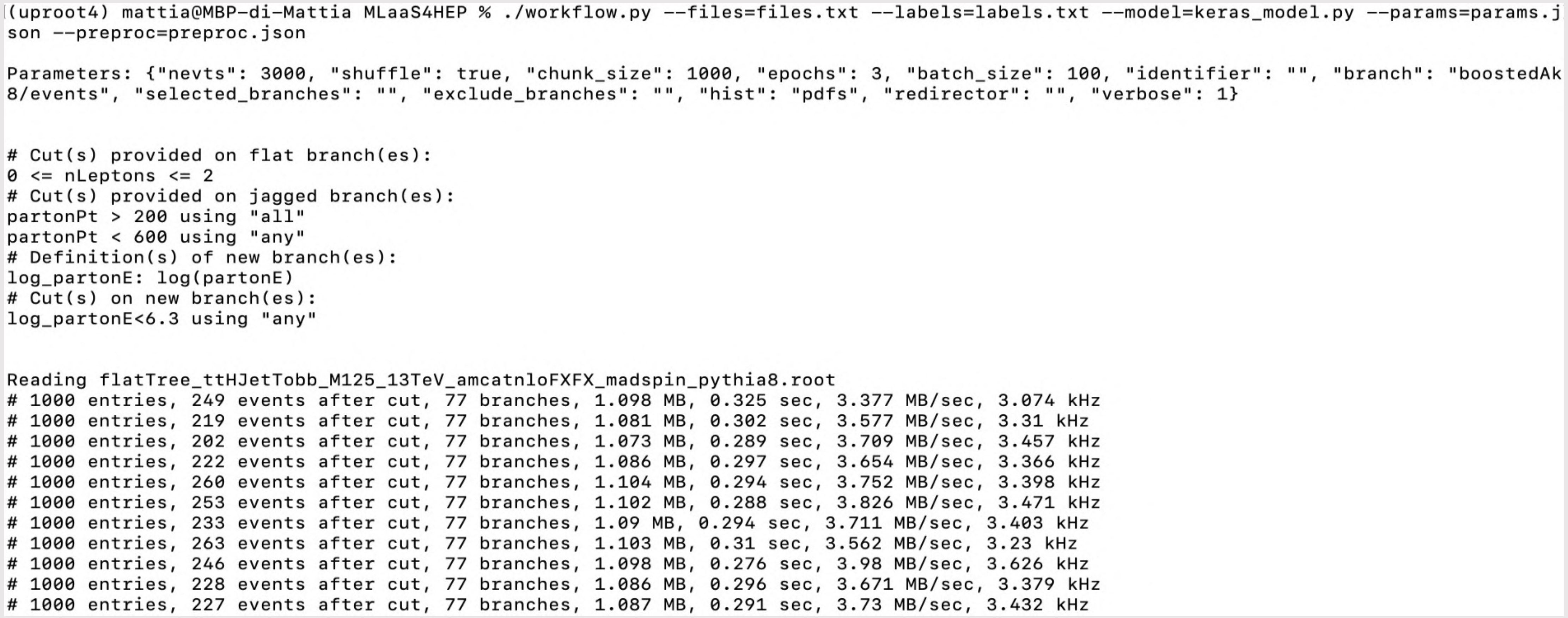
- The migration to the updated version of uproot allowed to create new branches and to apply cuts, both on new and on existing branches.

- 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"}}}
```

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
```

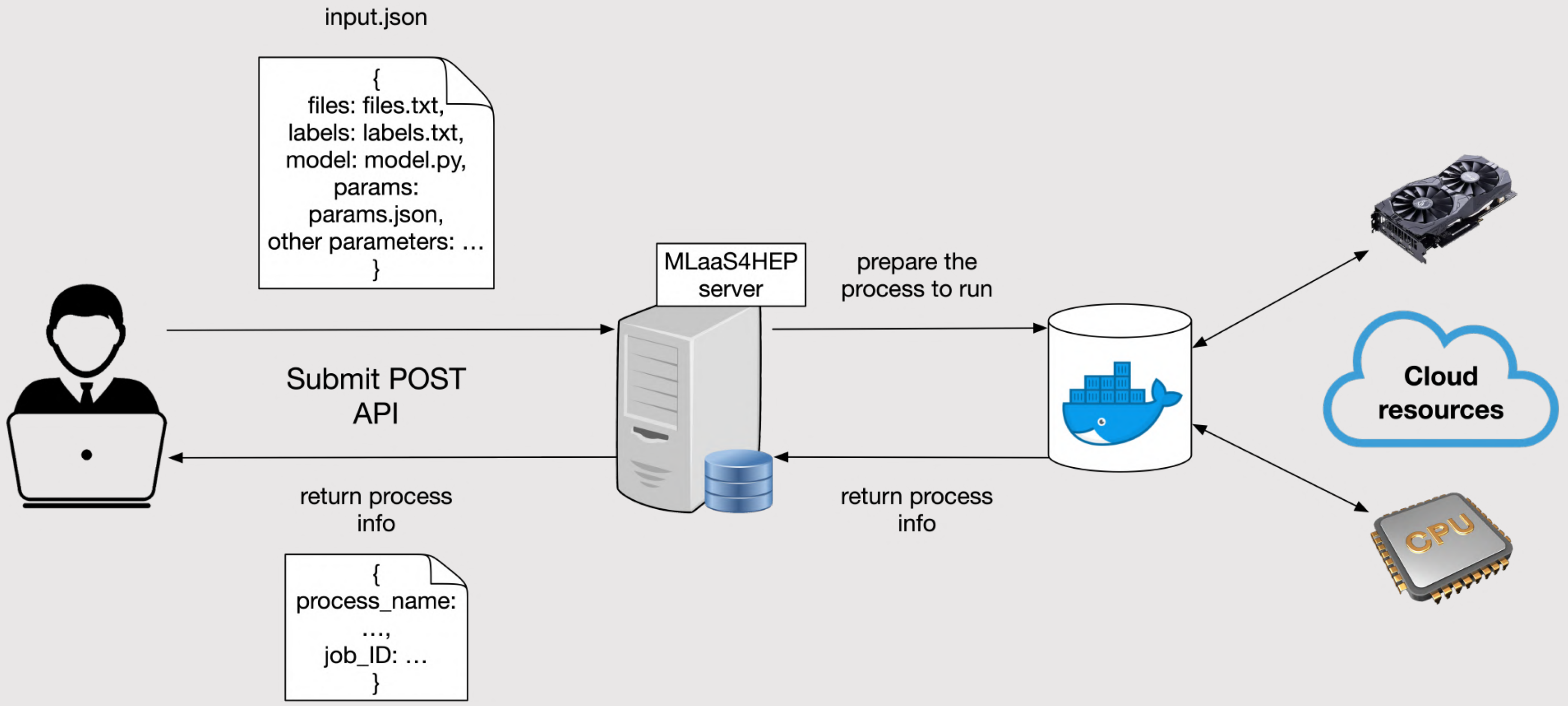
https://www.dropbox.com/s/95rzem8ofhgy8ei/DEMO_CUT.mp4?dl=0

Data Inference Layer

- ➢ The Data Inference Layer is implemented as TensorFlow as a Service (TFaaS), written in the **Go** programming language
    - The Go programming language natively supports concurrency and it is very well integrated with the TF library
- ➢ TFaaS is a generic (experiment and HEP agnostic) inference framework to serve pre-trained TF 1.X and TF 2.X models via HTTP protocol
- ➢ Both Python and C++ clients were developed on top of the REST APIs (end-points) and other clients can be developed thanks to HTTP protocol used by the TFaaS Go RESTful implementation
    - C++ client library talks to TFaaS using ProtoBuffer data-format, all others use JSON (see examples)
- ➢ TFaaS allows a rapid development or continuous training of TF models and their validation: clients can test multiple TF models at the same time
- ➢ A TFaaS server hosted by CERN is online: https://cms-tfaas.cern.ch/
    - Deployment: k8s using one node with 4 cores and 8GB of RAM

Towards MLaaS4HEP cloudification

- ➢ Now our goal is to create a cloud service that could use cloud resources and could be integrated into the INFN Cloud.
- ➢ The first step is to build a MLaaS4HEP server
    - The user should be able to submit a MLaaS4HEP workflow request to the server using an HTTP call



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

Manage user authentication with a proxy server

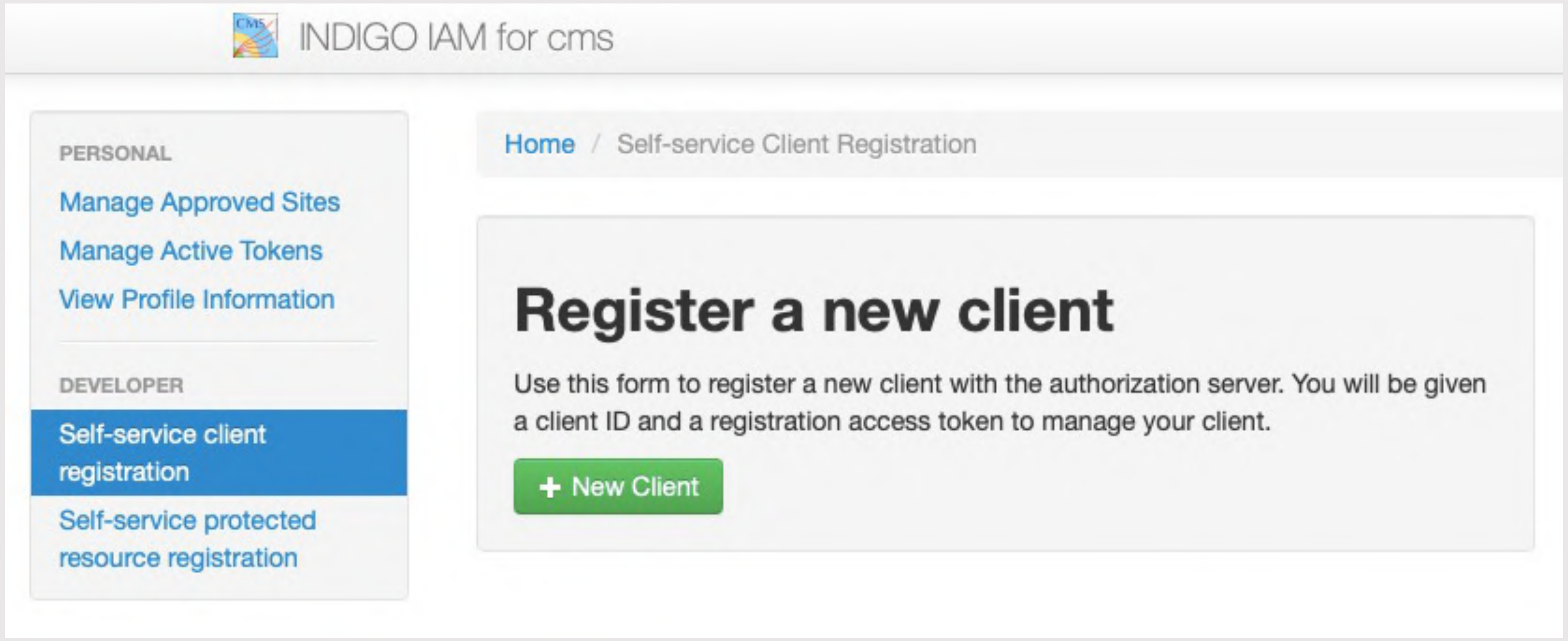
- One of the requirements for the MLaaS4HEP service to be integrated into the INFN Cloud is to have an authentication system to manage the user access.

- Successfully integrated dmwm/auth-proxy-server and oauth2-proxy locally
  - The first one is an R&D product in CMS while the second one is a generic tool

- Let's focus on the case of oauth2-proxy and on the steps needed to make a proxy server works
  - More details here

Integration of OAuth2 Proxy server

1) Register the client with the authorization server https://cms-auth.web.cern.ch/



INDIGO IAM for cms

Home / Self-service Client Registration

PERSONAL
- Manage Approved Sites
- Manage Active Tokens
- View Profile Information

DEVELOPER
- Self-service client registration
- Self-service protected resource registration

Register a new client

Use this form to register a new client with the authorization server. You will be given a client ID and a registration access token to manage your client.

+ New Client

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

```
oidc-agent
eval `oidc-agent`
oidc-gen -w device mlaas
TOKEN = $(oidc-token mlaas --aud=CLIENT_ID)
```

INDIGO - DataCloud

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)

```
provider = "oidc"
redirect_url = "http://localhost:4180/oauth2/callback"
oidc_issuer_url = "https://cms-auth.web.cern.ch/"
upstreams = [ "http://0.0.0.0:8080" ]
email_domains = [ "*" ]
client_id = "CLIENT_ID"
client_secret = "CLIENT_SECRET"
cookie_secret = "COOKIE_SECRET"
cookie_secure = false
skip_provider_button = true
ssl_insecure_skip_verify = true
```

```
provider="oidc"
https_address = ":4433"
redirect_url = "https://localhost:4433/oauth2/callback"
oidc_issuer_url = "https://cms-auth.web.cern.ch/"
upstreams = [ "http://127.0.0.1:8080/" ]
email_domains = [ "*" ]
client_id = "CLIENT_ID"
client_secret = "CLIENT_SECRET"
cookie_secret = "COOKIE_SECRET"
tls_cert_file = "./localhost.crt"
tls_key_file = "./localhost.key"
```

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

```
oauth2-proxy --config ./example_config.cfg --skip-jwt-bearer-tokens=true
```

```
docker run -v /home/giommi/luca:/etc --net=host quay.io/oauth2-proxy/oauth2-proxy:latest
--config=/etc/example_config.cfg  --skip-jwt-bearer-tokens=true
```

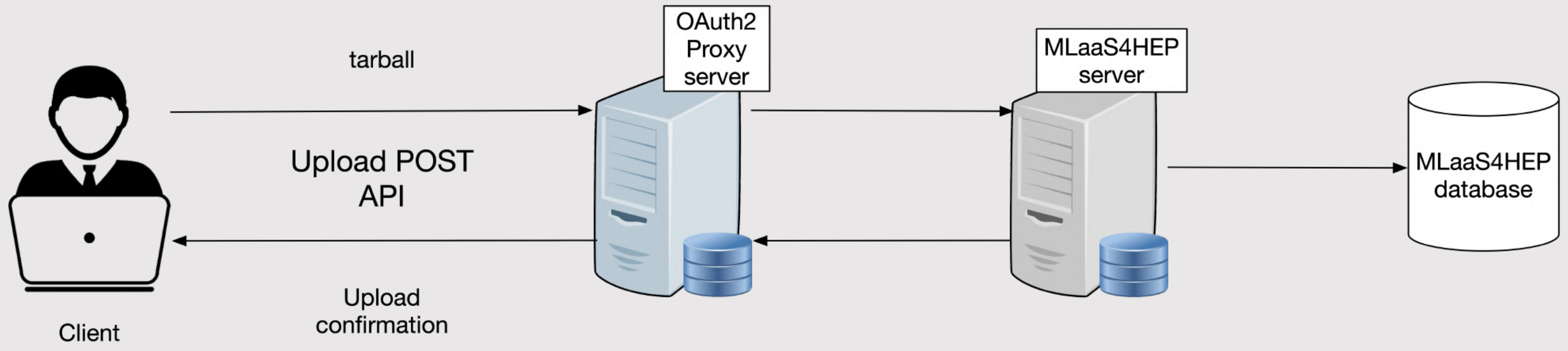
Integration of OAuth2 Proxy server

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

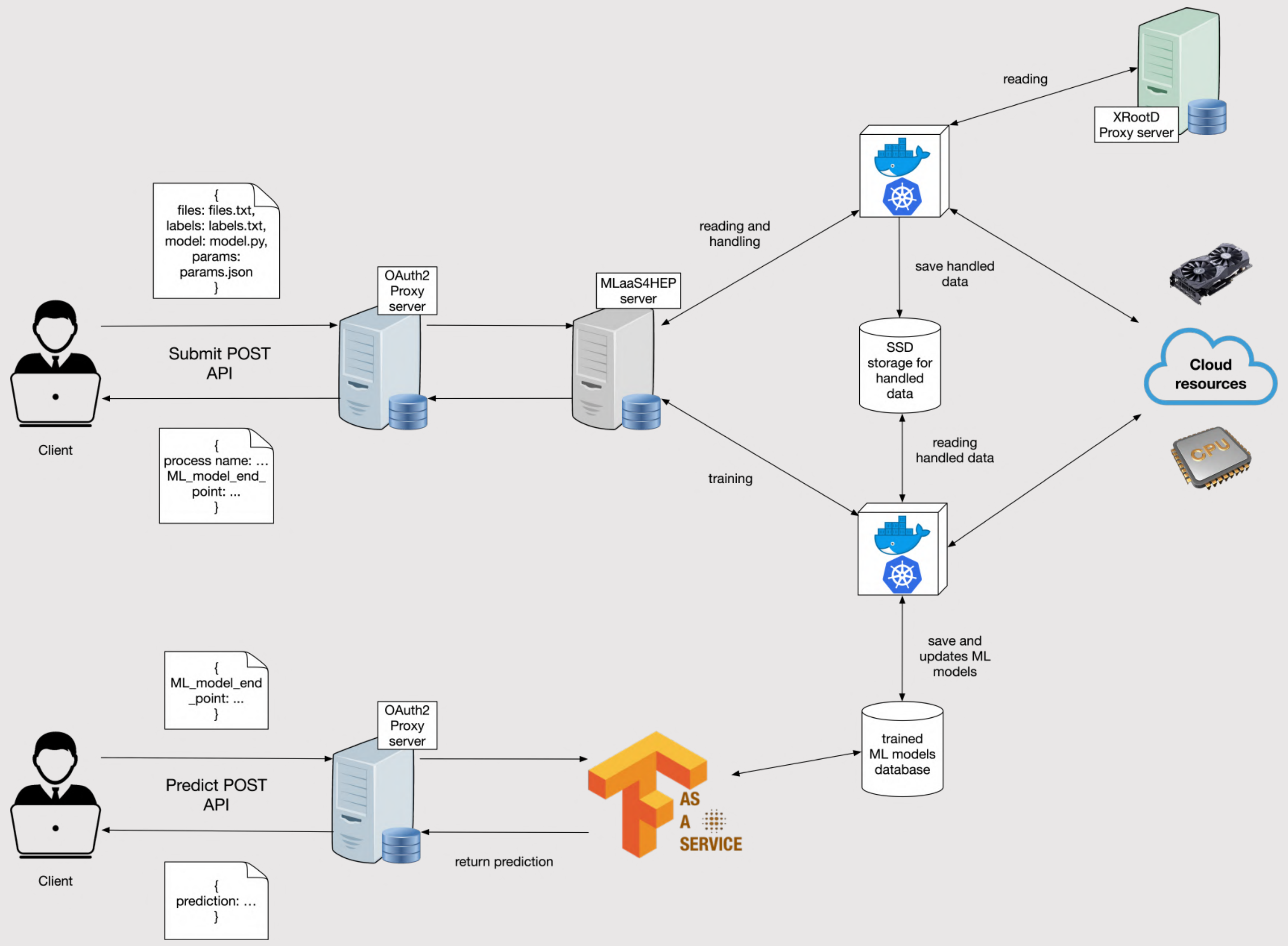
```
curl -L -k -H 'Content-Type: application/json' -H "Authorization: Bearer ${TOKEN}" -d @input.json
http://localhost:4180/submit
```

Preliminary final architecture

- In our research group we are still discussing about the final architecture of the MLaaS4HEP service, and we identified some needed key elements.
  - An OAuth2 Proxy server to manage user authentication
  - An XRootD Proxy server to store X.509 certificates needed for the access of remote ROOT files and used as cache
  - Divide the MLaaS4HEP workflow into two processes, one for reading + handling data and one for training. These two processes need a shared SSD storage for handled data
  - Database for trained ML models to share between MLaaS4HEP and TFaaS servers



Preliminary final architecture



Conclusions

- R&D project about a MLaaS4HEP solution
  - A CMS physics use case allowed us to validate the framework and test its performance
- To integrate MLaaS4HEP in the INFN Cloud we started working on its cloudification
  - Currently we created a MLaaS4HEP server and we integrated an OAuth2 Proxy server for user authentication
- The project is still ongoing and we planned to add other elements
  - SSD storage, XRootD Proxy server
- To deliver an efficient service to the physicists' community we need to guarantee its scalability and to provide a scheduler for the multiple requests
  - For these reasons we planned to use Kubernetes

Thanks for the attention

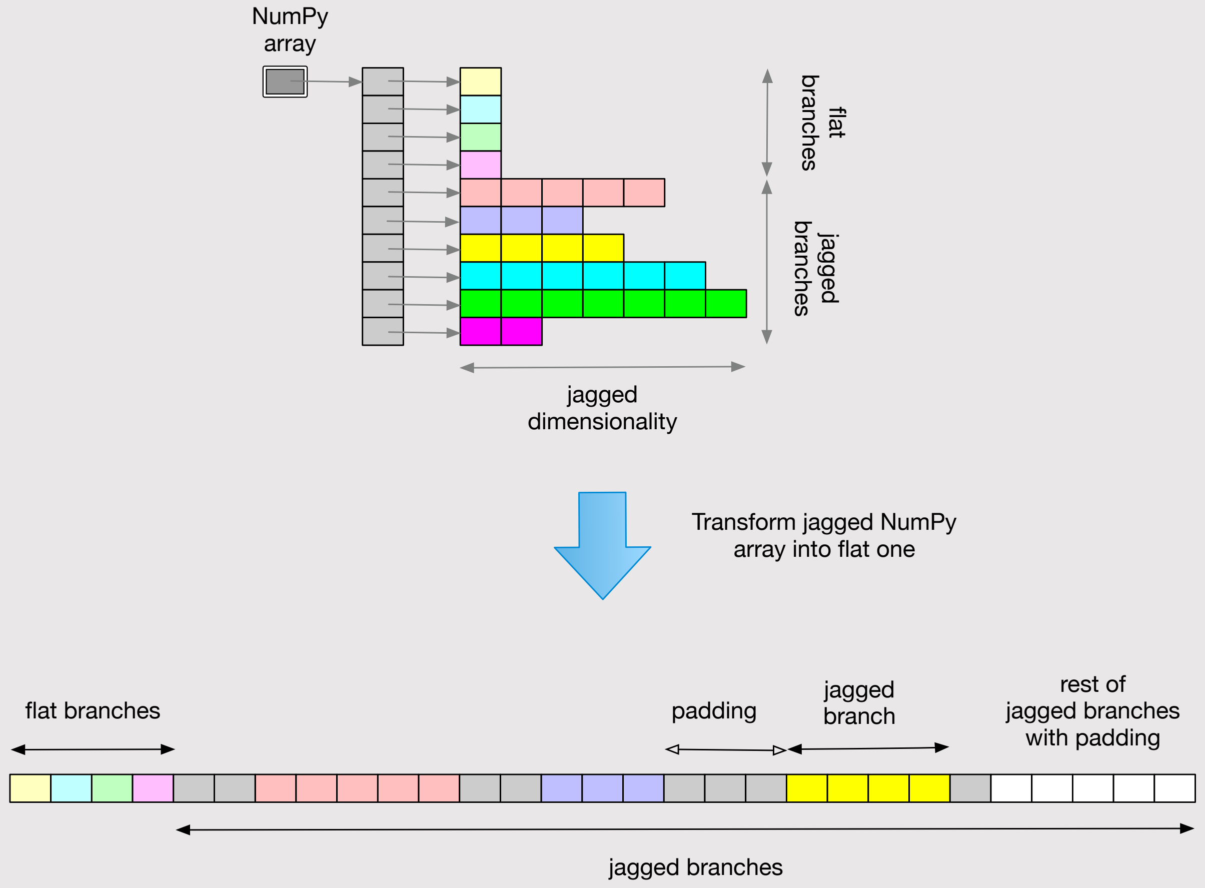
Backup slides

Issues with using existing solutions

- Existing MLaaS services can’t read HEP data directly in the **ROOT data-format**: most of the cases ML deal with either CSV or NumPy arrays representing tabular data
  - We don’t use ROOT data directly in the ML framework, we need a conversion step
  - Pre-processing operations may be more complex than offered by service providers
- R&D for specialized solutions to speed-up inference on FPGAs, e.g. HLS4ML
  - These solutions are designed for optimization of the inference phase rather than targeting the whole ML pipeline from reading data, to training and serving predictions
- Custom solutions adopted in specific CMS analysis (e.g DOI: 10.1088/2632-2153/ab9023) cannot easily generalized and do not represent ‘’as a Service’’ solutions
- Recent solution with Spark platform for data processing and ML training (DOI: 10.1007/s41781-020-00040-0). Here data are read from the CERN EOS storage system, not allowing access to data stored in WLCG sites

Jagged Arrays

- Each event is a composition of flat and Jagged Arrays
  - Such data representation is not directly suitable for ML (dynamic dimension of Jagged Arrays across events)
- To feed these data into ML we need to resolve how to treat Jagged Arrays. We opted to flatten Jagged Arrays into fixed-size array with padding values through a two-step procedure:
  - know the dimensionality of every Jagged Array attribute;
  - update the dimension of jagged branches using padding values, which should be assigned as NANs since all other numerical values can represent attribute spectrum.
- Keep the mask array with padding values location



MLaaS parameters

Read remote root files

Write and load the specs

```
./workflow.py --files=files.txt --labels=labels.txt —model=model.py --params=params.json
DataGenerator: <MLaaS4HEP.generator.RootDataGenerator object at 0x7f0cb58d7fd0> [29/Jun/2020:17:53:14] 1593445994.0
model parameters: {"nevts": 30000, "shuffle": true, "chunk_size": 10000, "epochs": 2, "batch_size": 500, "identifier": ["runNo", "evtNo", "lumi"],
"branch": "boostedAk8/events", "selected_branches": "", "exclude_branches": "", "hist": "pdfs", "redirector": "root://xrootd.ba.infn.it", "verbose": 1}


Reading root://xrootd.ba.infn.it//PATH_FILES/flatTree_ttHJetTobb_M125_13TeV_amcatnloFXFX_madspin_pythia8.root
# 10000 entries, 77 branches, 9.52220344543457 MB, 1.0169336795806885 sec, 9.36364252323795 MB/sec, 9.833482950553169 kHz
# 10000 entries, 77 branches, 9.533915519714355 MB, 1.2977769374847412 sec, 7.346343770133804 MB/sec, 7.705484441248654 kHz
# 10000 entries, 77 branches, 9.538667678833008 MB, 1.4104814529418945 sec, 6.7627033726234735 MB/sec, 7.089777734505208 kHz
--- first pass: 948348 events, (22-flat, 51-jagged) branches, 328 attrs
<MLaaS4HEP.reader.RootDataReader object at 0x7f840dbf4d50> init is complete in 4.8529925346374[illegible]1 sec


Reading root://xrootd.ba.infn.it//PATH_FILES/flatTree_TT_TuneCUETP8M2T4_13TeV-powheg-pythia8.root
# 10000 entries, 77 branches, 8.875920295715332 MB, 0.9596493244171143 sec, 9.24912889518941[illegible] MB/sec, 10.42047313071777 kHz
# 10000 entries, 77 branches, 8.868906021118164 MB, 1.2938923835754395 sec, 6.85443869497906[illegible] MB/sec, 7.728618026459661 kHz
# 10000 entries, 77 branches, 8.869449615478516 MB, 1.1267895698547363 sec, 7.87143389747730[illegible]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[illegible]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

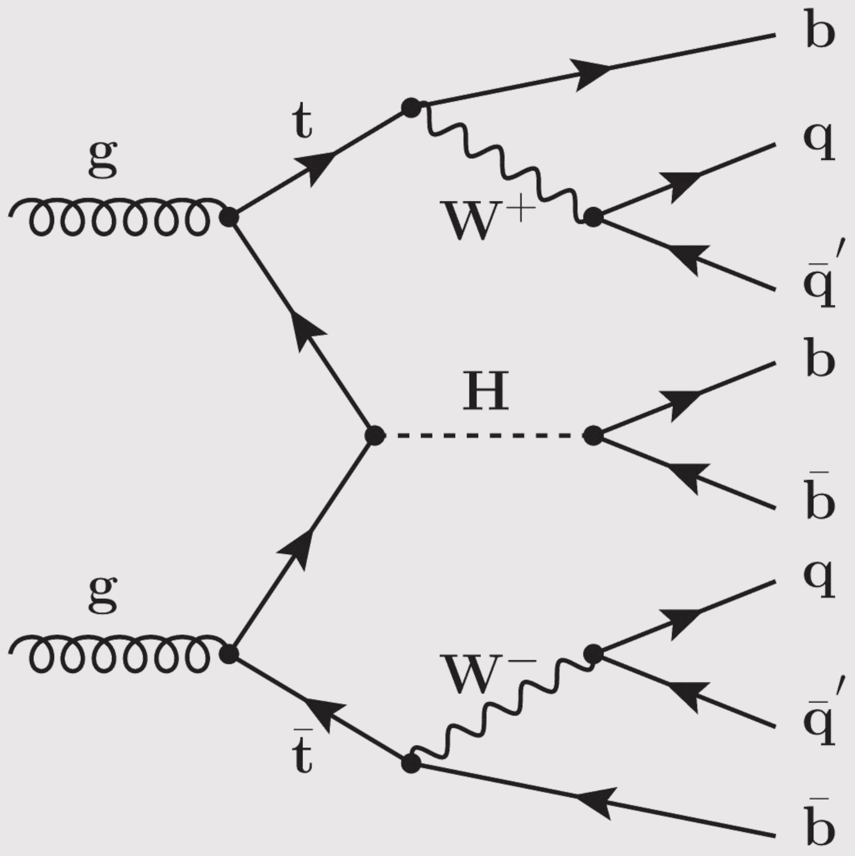
Real case scenario: $t\bar{t}H(bb)$ analysis in the boosted, all-hadronic final states

- We chose a signal vs background discrimination problem in a $t\bar{t}H$ analysis as a real physics use-case. This allowed us to:

  1. validate MLaaS results from the physics point of view
  2. test performances of MLaaS4HEP framework

- In the validation phase we used:
  - 9 ROOT files with a total of 350k events and a total size of ~ 28 MB
  - a simple Neural Network in Keras

- we obtained an AUC score comparable with the BDT-based analysis, performed within the TMVA framework by a subgroup of the CMS HIG PAG

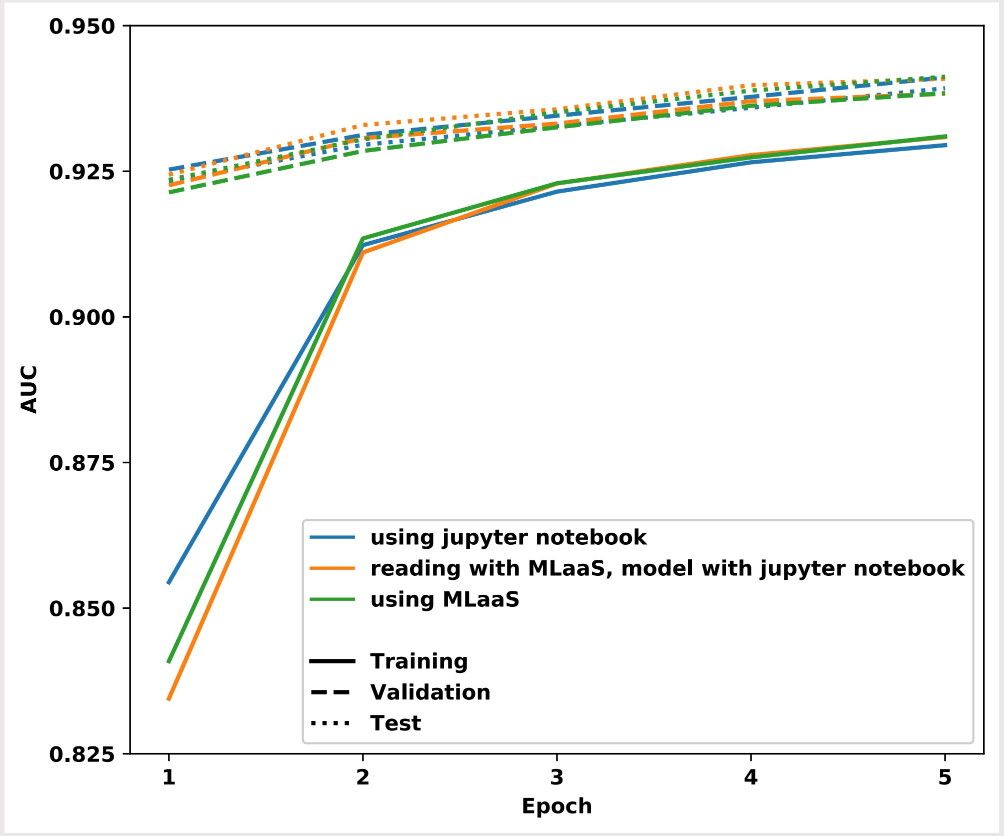


MLaaS4HEP validation

- ➢ Validate the MLaaS4HEP approach by comparing it with alternative methods on the reference use-case
  - We used a simple NN with Keras in all methods

- ➢ Validation successful: physics results are not impacted

- ➢ The AUC score is also comparable with the BDT-based analysis, performed within the TMVA framework by a subgroup of the CMS HIG PAG



AUC score

MLaaS4HEP performance

- For this phase we used all available ROOT files without any physics cut. This gave us a dataset with 28.5M events with 74 branches (22 flat and 52 Jagged), and a total size of about 10.1 GB.

- We performed all the tests running MLaaS framework on
  - macOS, 2.2 GHz Intel Core i7 dual-core, 8 GB of RAM
  - CentOS 7 Linux, 4 VCPU Intel Core Processor Haswell 2.4 GHz, 7.3 GB of RAM CERN Virtual Machine

- The average available bandwidth was approximately 129 Mbit/s and 639 Mbit/s using macOS and CERN VM, respectively.

- The ROOT files are read from local file-systems (SSD storages) and remotely from the Grid sites. In particular, we read files remotely from three different data-centers located at
  - Bologna (BO)
  - Pisa (PI)
  - Bari (BA)

MLaaS4HEP performance results

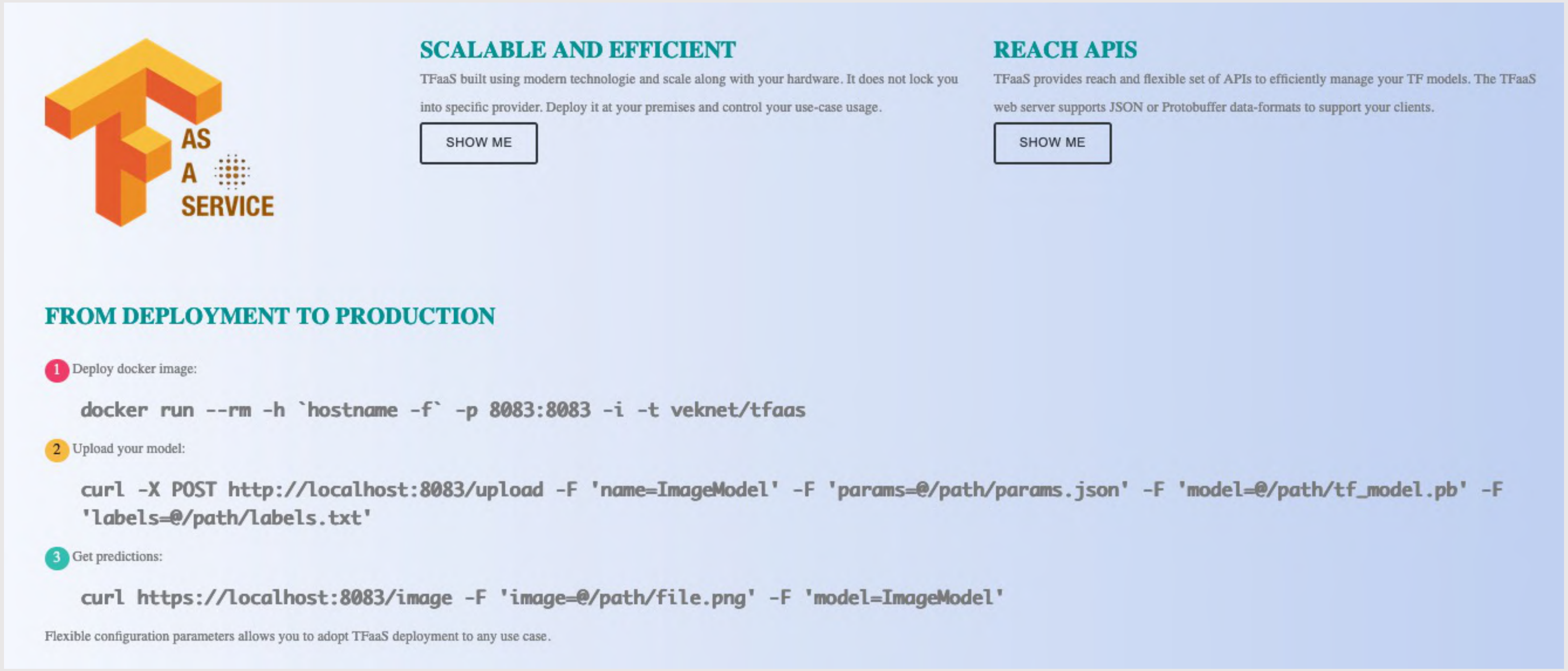
- Based on the resource we used and if the ROOT files were local or remote, we obtained:
  - **specs computing phase (chunk size = 100k events)**
    - Event throughput: 8.4k – 13.7k evts/s
    - Total time using all the 28.5M events: 35 – 57 min
  - **chunks creation in the training phase (chunk size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

- 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.

How to use TFaaS

DEMO



SCALABLE AND EFFICIENT

TFaaS built using modern technologie and scale along with your hardware. It does not lock you into specific provider. Deploy it at your premises and control your use-case usage.

REACH APIS

TFaaS provides reach and flexible set of APIs to efficiently manage your TF models. The TFaaS web server supports JSON or Protobuffer data-formats to support your clients.

FROM DEPLOYMENT TO PRODUCTION

1 Deploy docker image:

```
docker run --rm -h `hostname -f` -p 8083:8083 -i -t veknet/tfaas
```

2 Upload your model:

```
curl -X POST http://localhost:8083/upload -F 'name=ImageModel' -F 'params=@/path/params.json' -F 'model=@/path/tf_model.pb' -F
'labels=@/path/labels.txt'
```

3 Get predictions:

```
curl https://localhost:8083/image -F 'image=@/path/file.png' -F 'model=ImageModel'
```

Flexible configuration parameters allows you to adopt TFaaS deployment to any use case.

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

TFaaS performance

- We did TFaaS benchmarks on CentOS 7 Linux, 16 cores, 30 GB of RAM in two modes:
    - using 1k calls with 100 concurrent clients,
    - using 5k calls with 200 concurrent clients.

- We tested both JSON and ProtoBuffer data formats while sending and fetching the data to/from the TFaaS server.

- In both cases, we achieved a throughput of **500 req/sec**. These numbers were obtained by serving a mid-size pre-trained NN model with 27 features and 1024x1024 hidden layers used in the physics analysis discussed. Similar performance was found for image classification datasets (MNIST).

- The actual performance of TFaaS will depend on the complexity of served ML model and available hardware resources.

- Even though a single TFaaS server may not be as efficient as an integrated solution, it can be horizontally scaled, e.g. using Kubernetes or other cluster orchestrated solutions, and may provide the desired throughput for concurrent clients.

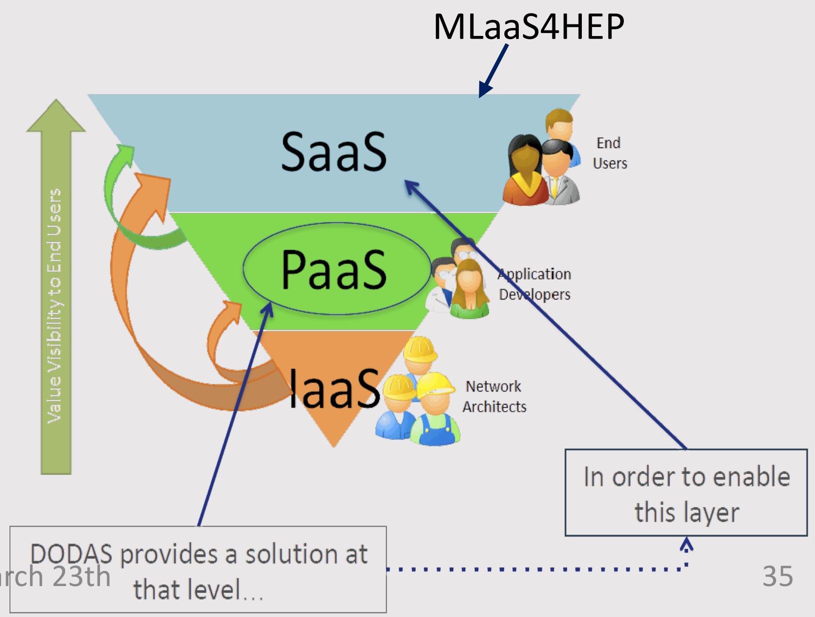
Towards MLaaS4HEP cloudification

- ➢ We created a service performing the ML pipeline using local and remote ROOT files
  - The performance strictly depends on the available hardware resources

- ➢ How to improve the performance?
  - Adopt new solutions in the code
  - Invest in better and more expensive on-premise resources
  - Move to the cloud

- ➢ The operation of cloudification has two benefits:
  - opens us to potentially more performing resources
  - provides a real ''as a Service'' solution for the user

- ➢ We started to work for a MLaaS4HEP cloudification using DODAS
  - implements services composition model based on templates


MLaaS4HEP
Value Visibility to End Users
SaaS
PaaS
IaaS
End Users
Application Developers
Network Architects
In order to enable this layer
DODAS provides a solution at that level…

MLaaS4HEP cloudification with DODAS

Creation of a docker image able to run the workflow.py script

Create an Ansible playbook to automatize the configuration and deployment of the container with dependencies

Convert the Ansible playbook into an Ansible role

Creation of a Tosca template to define the resource requirements and the input parameters for the creation of the docker container

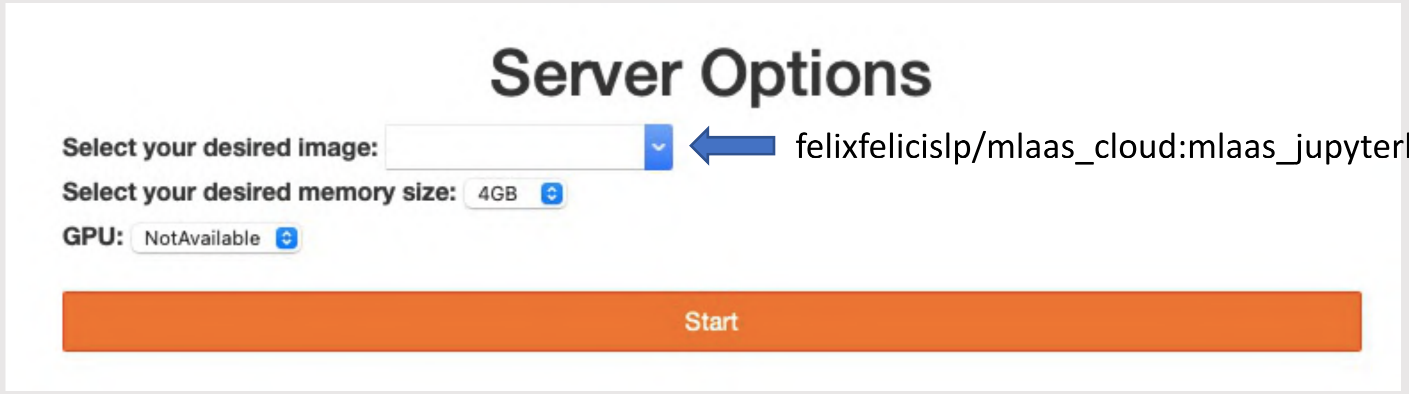
Create the deployment from command line

Run workflow.py interactively or with jupyterhub

```
dodas create lgiommi-template.yml
    dodas login <infID> <vmID>
```

MLaaS4HEP using Jupyterhub

- We provide a SaaS solution for a sharable jupyter notebook
- Token-based access to the jupyterhub, with the support for a customizable environment



Select your desired image:
Select your desired memory size: 4GB
GPU: NotAvailable

felixfelicislp/mlaas_cloud:mlaas_jupyterhub

- Integrate cloud storage for managing the required files (ROOT files, ML model, etc.)

```
# . ./shared/setup_local
(base) # cd /workarea/shared/folder_test
(base) # ../../workarea/MLaaS4HEP/src/python/MLaaS4HEP/workflow.py --files=files_test.txt --labels=labels_test.txt --
model=keras_model.py --params=params_test.json
model parameters: {"nevts": -1, "shuffle": true, "chunk_size": 10000, "epochs": 5, "batch_size": 100, "identifier": ["runNo", "evtNo",
"lumi"], "branch": "events", "selected_branches": "", "exclude_branches": "", "hist": "pdfs", "redirector": "root://gridftp-storm-
t3.cr.cnaf.infn.it:1095", "verbose": 1}
Reading ttH_signal.root
# 10000 entries, 29 branches, 1.10626220703125 MB, 0.034181833267211914 sec, 32.364039645948566 MB/sec, 292.5530623775014 kHz
# 10000 entries, 29 branches, 1.10626220703125 MB, 0.022344589233398438 sec, 49.50917626973965 MB/sec, 447.53563807084936 kHz
```