Machine Learning as a Service for HEP

Luca Giommi[1], Valentin Kuznetsov[2], Daniele Bonacorsi[1]

[1] University of Bologna and INFN

[2] Cornell University, USA

luca.giommi3@unibo.it, vkuznet@gmail.com, daniele.bonacorsi@unibo.it

4th IML Machine Learning Workshop - 22.10.2020

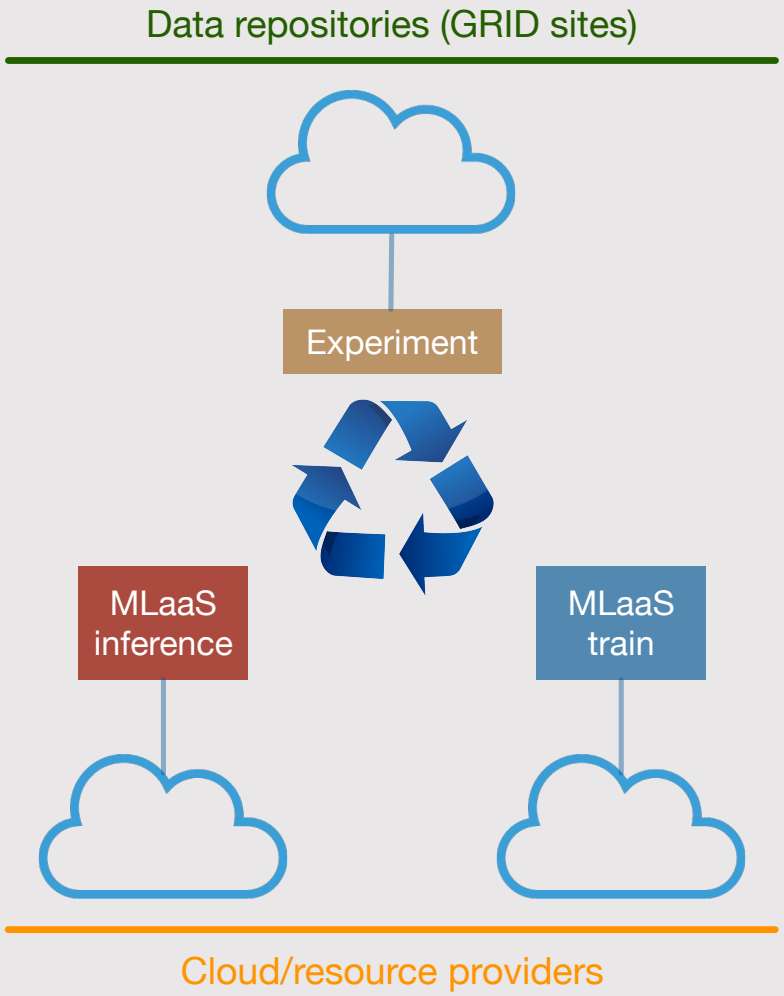
Issues on using existing solutions

- Existing MLaaS services (e.g. provided by Amazon, Google or Microsoft) can't read HEP data directly in 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 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, SonicCMS, etc.
  - These solution are designed for optimization of inference phase rather than targeting the whole ML pipeline from reading data, to training and serving predictions.

Towards MLaaS for HEP

MLaaS for HEP should provide 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**, local CPU, GPUs, farms, cloud resources, etc.
- **use different ML frameworks** (TF, PyTorch, etc.)
- **serve pre-trained HEP models**, like a model repository, and access it easily from any place, any code, any framework.



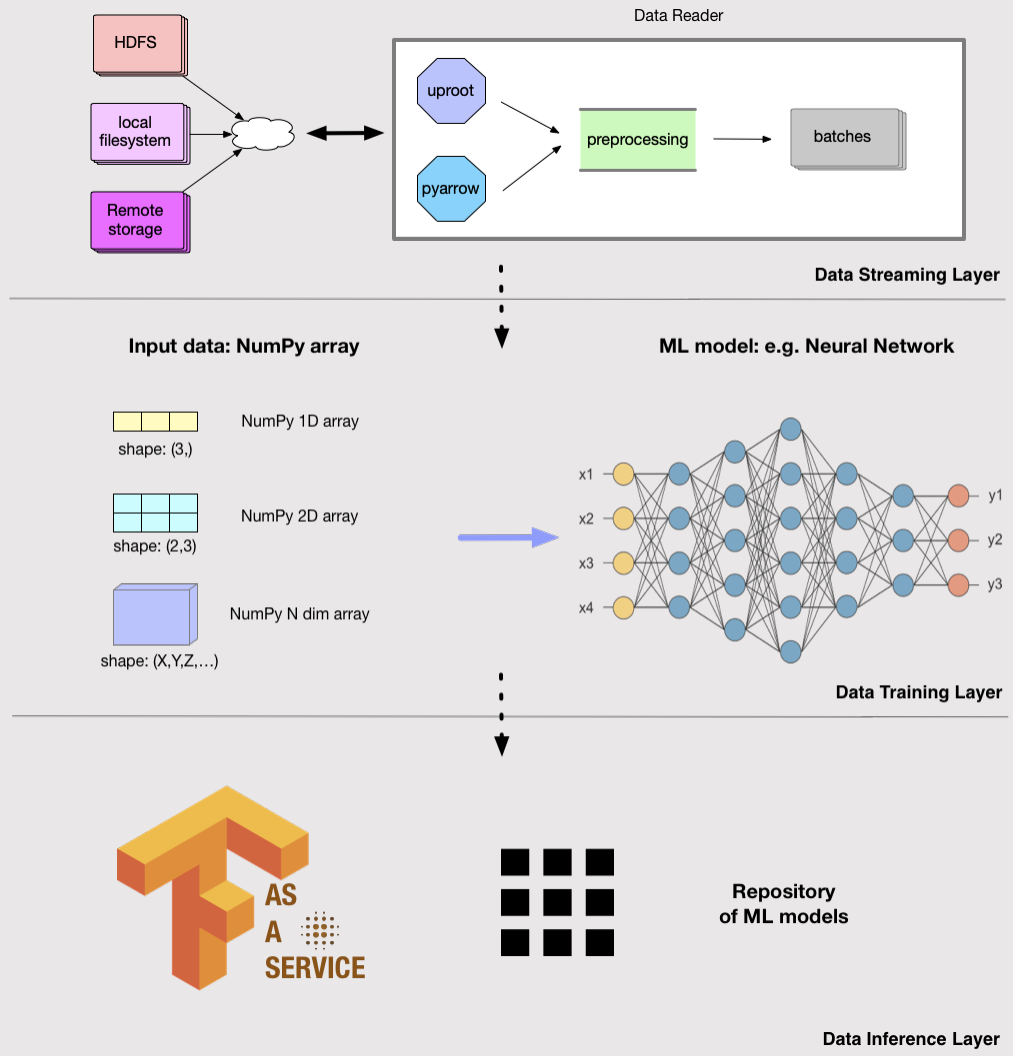
ML as a Service for HEP R&D

- **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** provides access to pre-trained HEP model for HEP users

All three layers are independent from each other and allow independent resource allocation.

Data streaming and training tools: github.com/vkuznet/MLaaS4HEP
Data inference tools: github.com/vkuznet/TFaaS

https://arxiv.org/abs/2007.14781 https://arxiv.org/abs/1811.04492v2



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

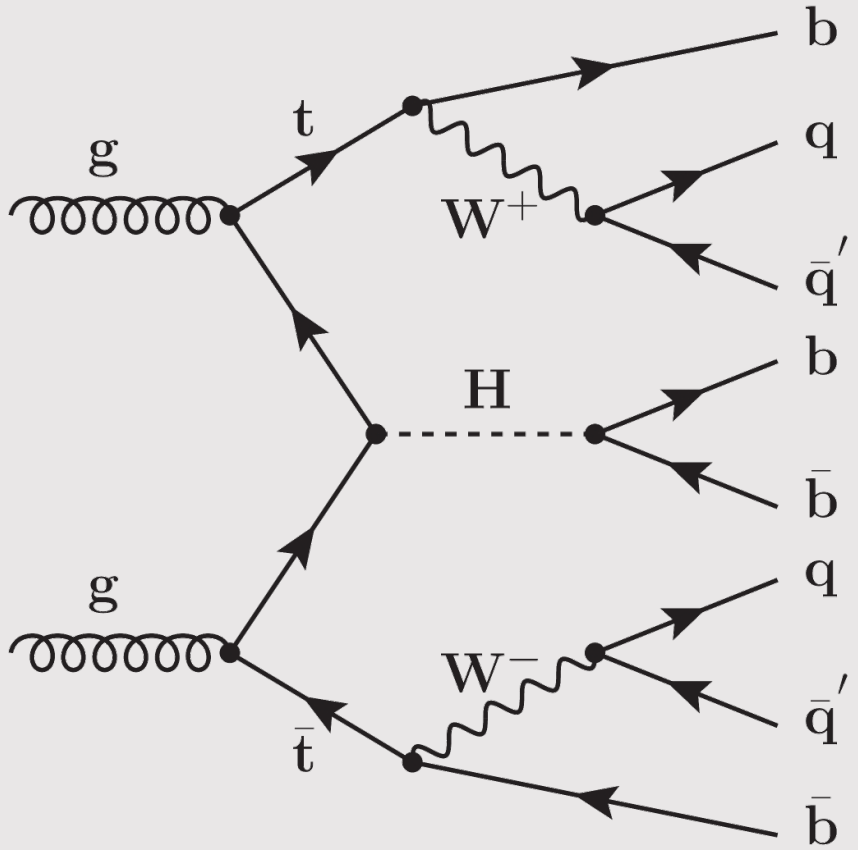
- A proof-of concept of the entire pipeline using CMS NANOAOD.

- We want to **validate** the MLaaS framework, namely we want to test the infrastructure on real physics use-case. We chose a signal vs background discrimination problem in a $t\bar{t}H$ analysis. This allows us to:

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

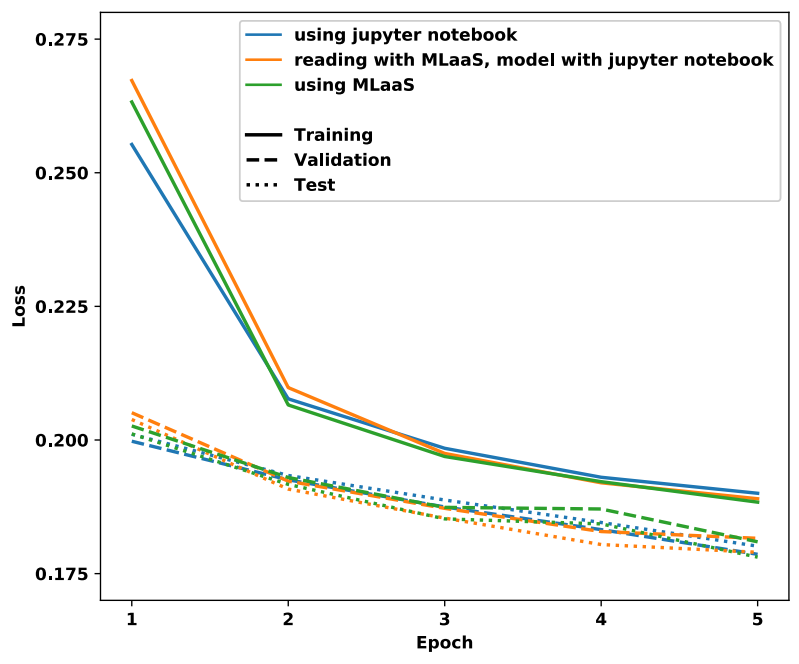
For phase 1 we used 9 ROOT files, 8 of background and 1 of signal. Each file has 27 branches, with 350 hundreds events for the whole pool of files and a total size of almost 28 MB. Ratio between signal and background is 10.8%. We need to:

- choose a generic ML model
- compare results inside and outside MLaaS

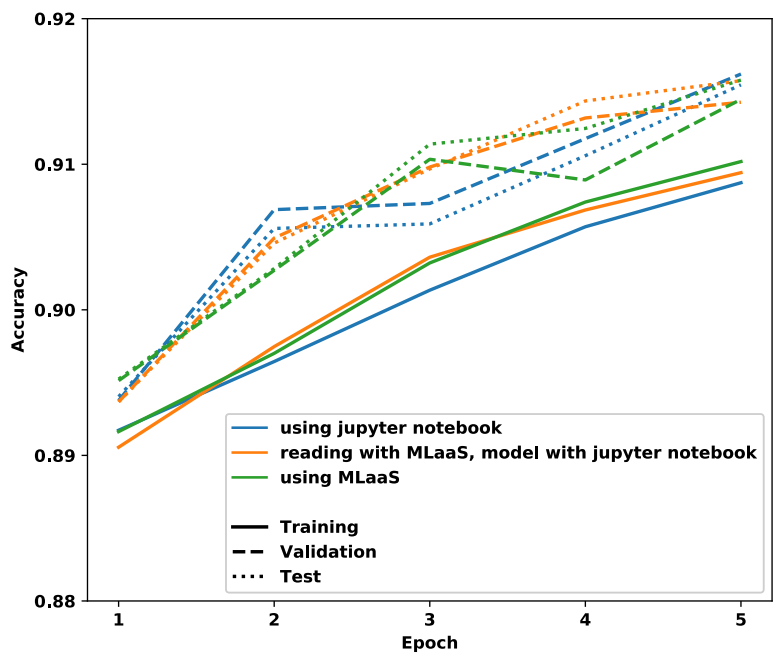


MLaaS validation

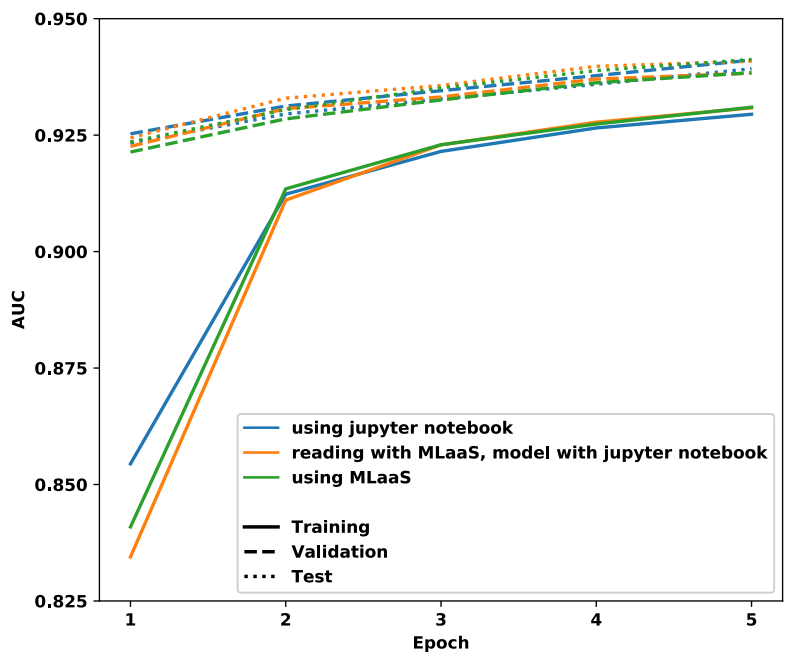
In order to verify the validity of the MLaaS framework, we decided to use a generic ML model (Keras sequential Neural Network) and to compare the results obtained inside and outside MLaaS: curves are almost indistinguishable!



Loss



Accuracy



AUC

MLaaS performance

- For this phase we used all available ROOT files without any physics cut: namely 8 ROOT files, 7 of background and 1 of signal, with a total size of almost 10.1 GB and almost 28.5 million of events.

- 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 ROOT files are read from local file-systems 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).

E.g. in macOS with local files we have a reading + specs computing frequency of 11 kHz and a frequency of creating a chunk of 1.1 kHz (for chunk size fixed to 100 thousands events).

Summary

We built a MLaaS solution for HEP where:

- local and remote ROOT files can be directly read
- complexity of data transformation from ROOT I/O to ML is hidden to the user
- resources can be used dynamically and independently for training and inference layers
- ML framework of user choice can be used (R&D works towards model transformation from one framework to another)
- customization is provided: total number of events to read; chunk size of data read; select or exclude branches to read, choice of XrootD redirector

And…

- we validated the MLaaS framework with a physics use case
- we did performance tests for the streaming and training layers

Several aspects are planned to be investigated and integrated: reading multiple files concurrently, doing distributed training, dynamically load user based pre-processing functions; possible graphical UI to build full workflow pipeline (via go-hep/groot)

For more details about MLaaS for HEP: https://arxiv.org/abs/2007.14781

Thanks for the attention

Backup slides

Machine Learning as a Service

CLOUD MACHINE LEARNING SERVICES COMPARISON

	Amazon	Microsoft	Google	IBM
Automated and semi-automated ML services				
	Amazon ML	Microsoft Azure ML Studio	Google Prediction API	IBM Watson ML Model Builder
Classification	✓	✓	deprecated	✓
Regression	✓	✓		✓
Clustering	✓	✓		✗
Anomaly detection	✗	✓		✗
Recommendation	✗	✓		✗
Ranking	✗	✓		✗
Platforms for custom modeling				
	Amazon SageMaker	Azure ML Services	Google ML Engine	IBM Watson ML Studio
Built-in algorithms	✓	✗	✗	✓
Supported frameworks	TensorFlow, MXNet, Keras, Gluon, Pytorch, Caffe2, Chainer, Torch	TensorFlow, scikit-learn, Microsoft Cognitive Toolkit, Spark ML	TensorFlow, scikit-learn, XGBoost, Keras	TensorFlow, Spark MLlib, scikit-learn, XGBoost, PyTorch, IBM SPSS, PMML

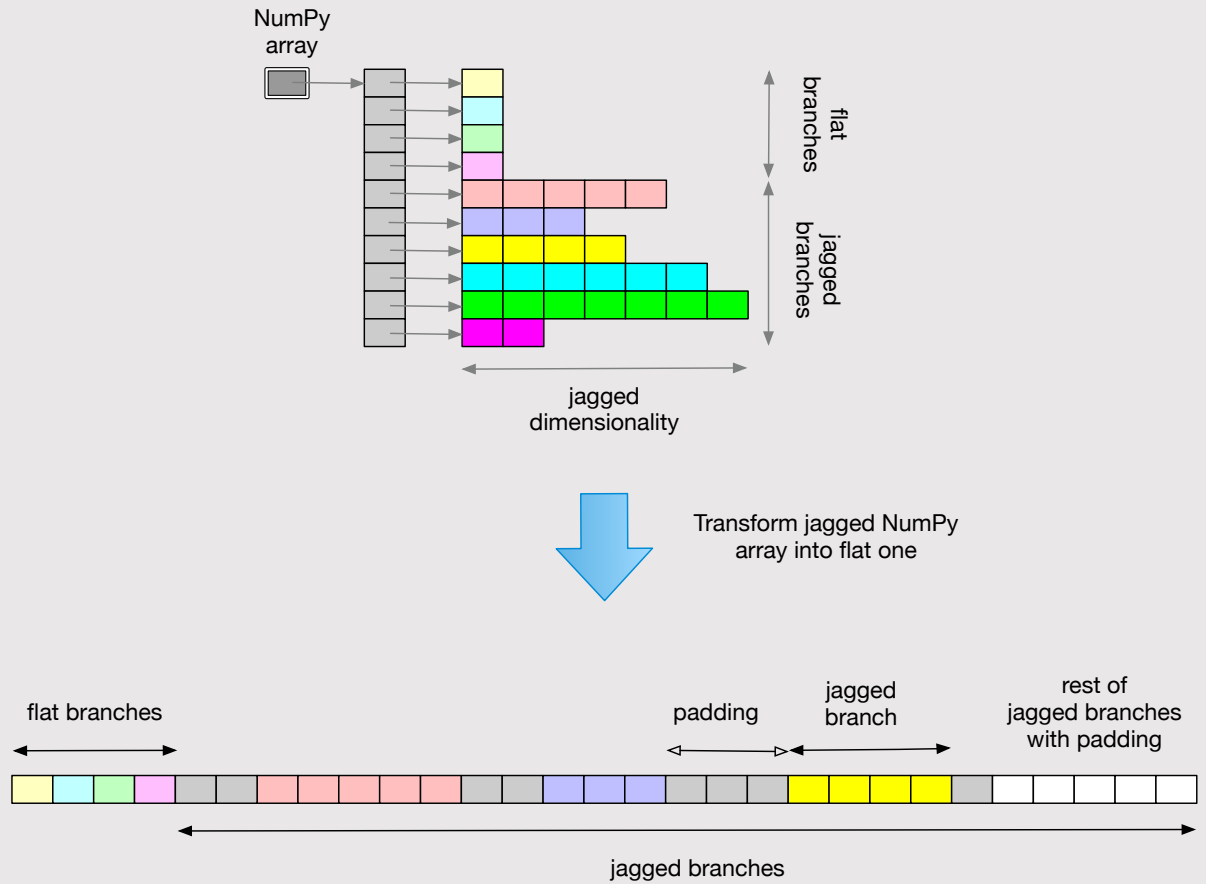
- MLaaS is a set of tools and services including: data visualization, pre-processing, model training and evaluation, serving predictions, etc.
- Many of the world's leading cloud providers provide different types of MLaaS services, including Amazon, Microsoft, Google and IBM.
- MLaaS service providers offer pre-defined models that can be used to cover standard use-cases, e.g. classification, regression, natural language processing, facial recognition, DeepLearning.

Data Streaming Layer

- The development of DIANA-HEP uproot library provides ability to read ROOT data in Python, access them as NumPy arrays, and implements XrootD access
- Now we're able to access ROOT files via XrootD protocol in C++, Python and Go
- MLaaS4HEP extends uproot library and provide APIs to read local and remote distributed ROOT files and feed them into existing ML frameworks
  - the DataReader and DataGenerator wrappers are created to read ROOT files and deliver them upstream as batches
    - random reads from multiple files are also supported (data shuffle mode)
  - the non-flat ROOT branches are read and represented as Jagged Arrays

Data Training Layer

- Each event is a composition of flat and Jagged Arrays, where usually flat arrays size is less then jagged ones
- 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 array into fixed size array with padding values through a two-step procedure:
  - know up-front dimensionality of every Jagged Array attribute (pre-processing step)
  - update 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.



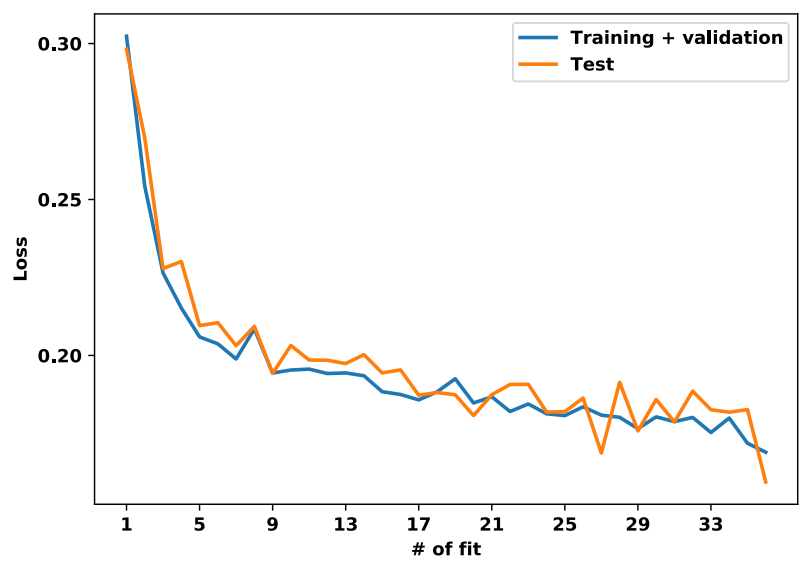
Data Inference Layer

- Data Inference Layer is implemented as TensorFlow as a Service (TFaaS), written in the Go programming language
- Capable of serving any TensorFlow model
- Can be used as global repository of pre-trained HEP models
- 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
- Can be deployed everywhere (Docker image and Kubernetes files are provided)
- TFaaS allows a rapid development or continuous training of TF models and their validation: clients can test multiple TF models at the same time

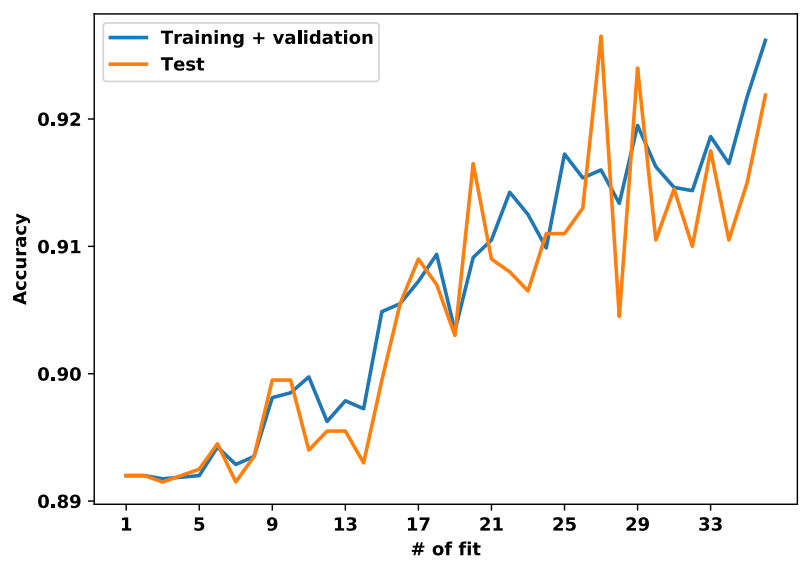
MLaaS validation (2)

In order to properly train any ML model we need the ability to read data in chunks and shuffle them accordingly in each batch of training. How?

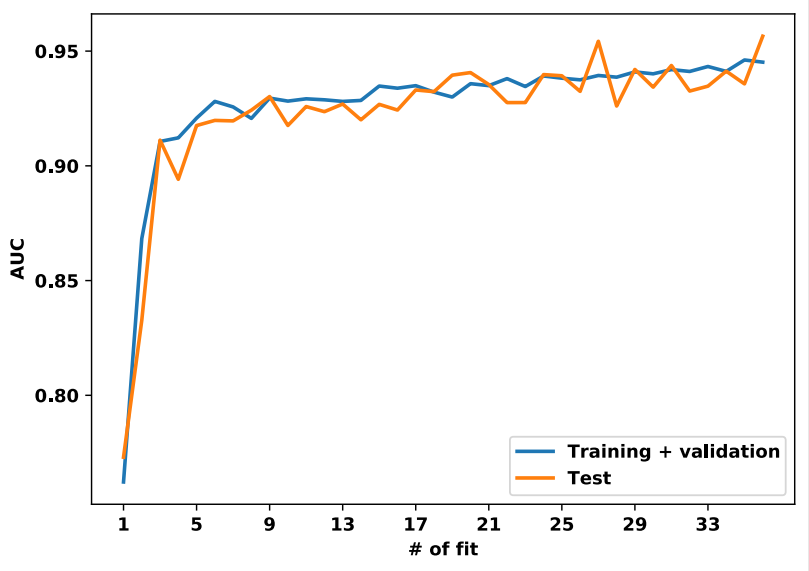
- The user specifies a chunk size (in this case 10 thousands events) and MLaaS ensures that each chunk will have the same proportion of signal and backgrounds events presented in the ROOT files.



Loss



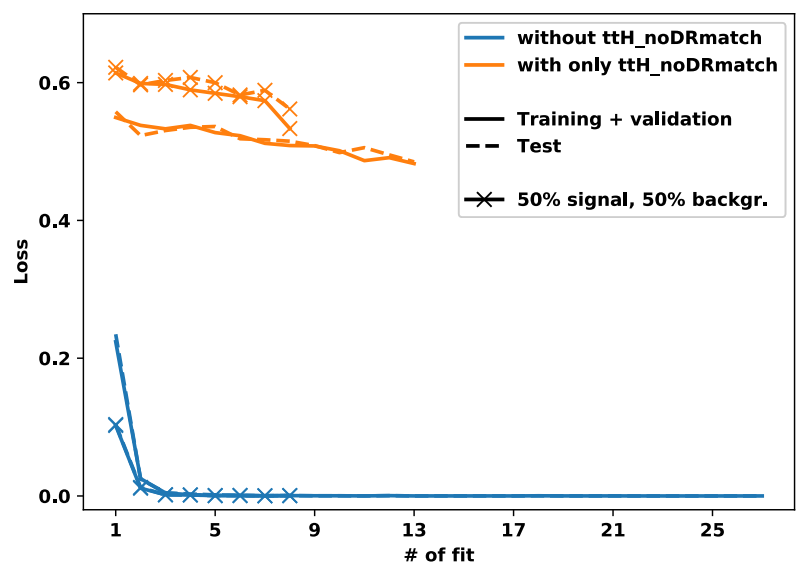
Accuracy



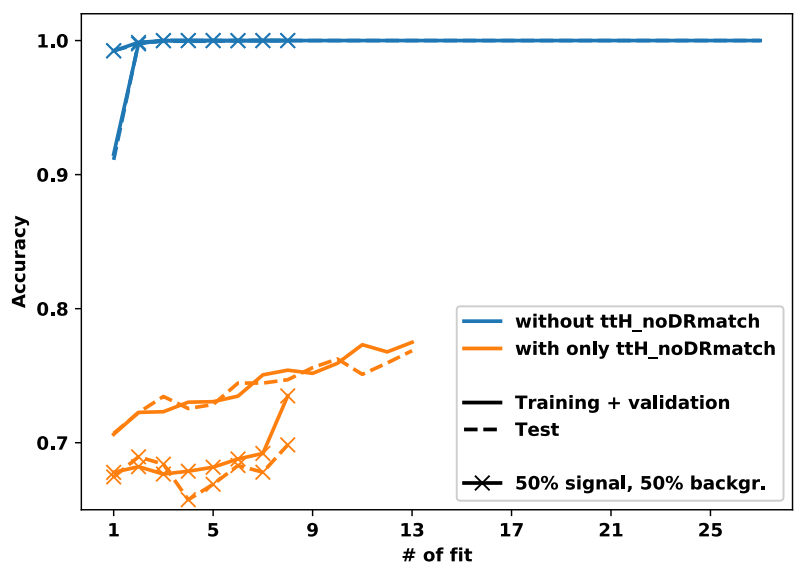
AUC

MLaaS validation (3)

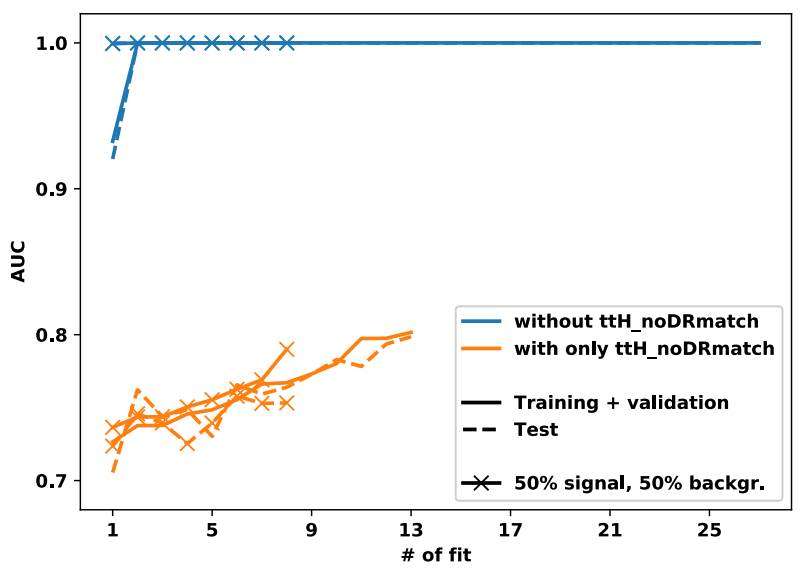
The trend of the metrics in the previous slide is not smooth, namely we see sawtooth shape patterns.
To investigate this behaviour we dropped one by one ROOT files from the pool, and we found that a particular ROOT file with *ttH_noDRmatch* background is responsible for this effect.



Loss



Accuracy



AUC

MLaaS validation (4)

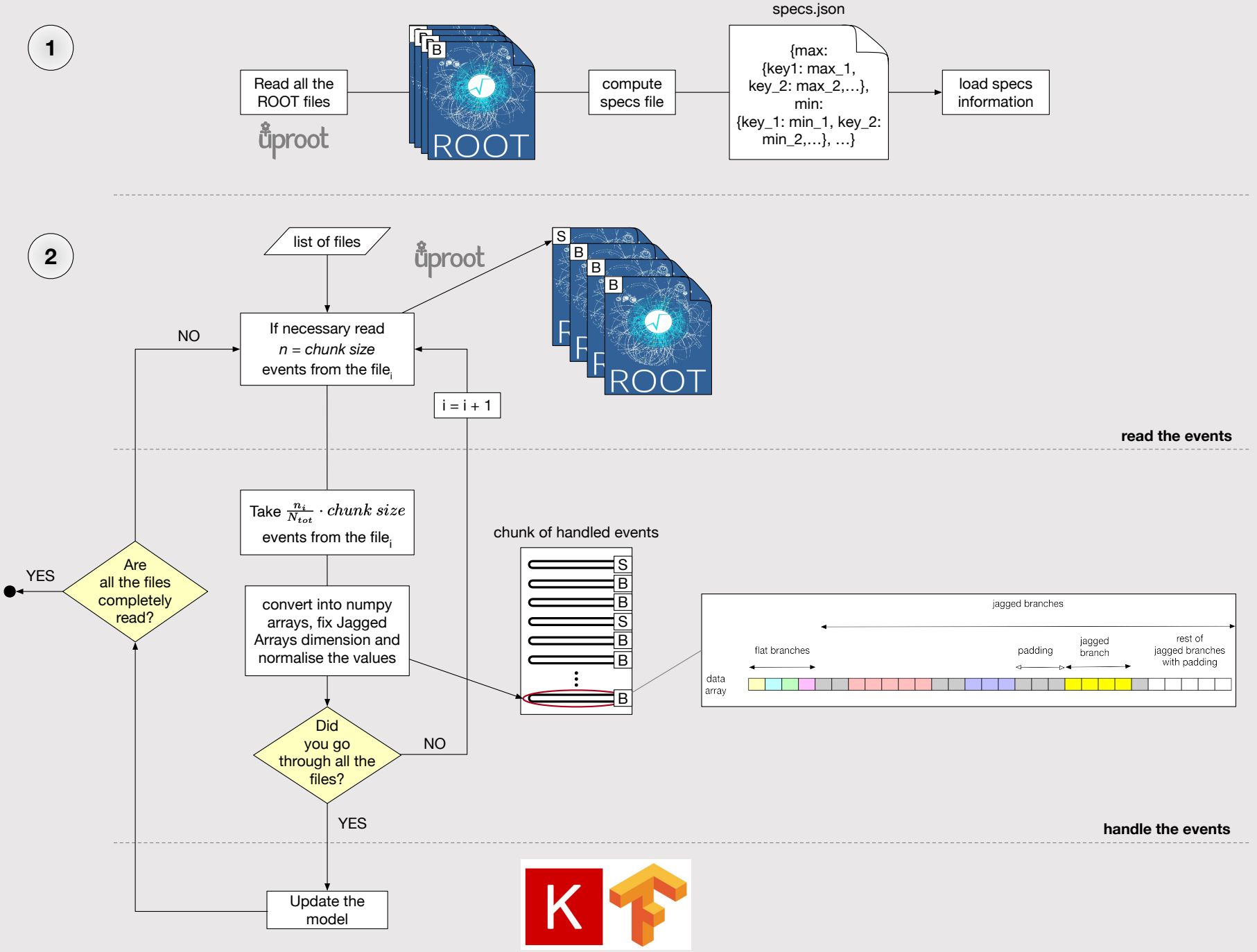
Actually, we found that physics analysis drop certain features due to their high correlations and used set of custom engineered features in their final ML training.

	loss	accuracy	AUC
27 features model	0.187	0.911	0.934
14 features model	0.298	0.892	0.766
analysis reference			0.886

The reason of the discrepancy between different models is two-fold.

- we did not performed any features engineering.
- the results of physics analysis was based on TMVA tool, which we treated as black-box. Moreover, we knew that in TMVA a weight for each ROOT file was applied according to the inverse of the luminosity.

But this is not our goal: in fact we have a clear demonstration of a working system, we validate it by applying it to an HEP use case, and we are able to build ML models using directly ROOT files.


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
ROOT
If necessary read
n = chunk size
events from the file_i
NO
i = i + 1
read the events
Take $\frac{n_i}{N_{tot}} \cdot chunk\ size$
events from the file_i
chunk of handled events
Are
all the files
completely
read?
YES
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
handle the events
Update the
model

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

MLaaS workflow | Input ROOT files | Labels of ROOT files | User model | MLaaS parameters

Keras model

```
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

```
{
    "nevts": 30000,
    "shuffle": true,
    "chunk_size": 10000,
    "epochs": 5,
    "batch_size": 100,
    "identifier":["runNo", "evtNo", "lumi"],
    "branch": "boostedAk8/events",
    "selected_branches":"",
    "exclude_branches": "",
    "hist": "pdfs",
    "redirector": "root://xrootd.ba.infn.it",
    "verbose": 1
  }
```

MLaaS parameters

Read remote root files

Write and load the specs

```
./workflow.p  --files=files.txt --la ls=lab s.txt -model=model.py --params=params.json
DataGenerato : <MLaaS4HEP.generator ootData enerator object at 0x7f0cb58d7fd0> [29/Jun/2020:17:53:  ] 1593445994.0
model parameters: {"nevts": 30000 "shuffle  true, "chunk_size": 10000, "epochs": 2, "batch_size"  500, "identifier": ["runNo", "evtNo", "lumi"],
"branch": "boostedAk8/events", "  elected_b nches": "", "exclude_branches": "", "hist": "pdfs", "r director": "root://xrootd.ba.infn.it", "verbose": 1}


Reading root://xrootd.ba.infn.it//$PATH_F LES/flatTree_ttHJetTobb_M125_13TeV_amcatnloFXFX_madspi pythia8.root
# 10000 entries, 77 branches, 9.52220344 3457 MB, 1.0169336795806885 sec, 9.36364252323795 MB/s , 9.833482950553169 kHz
# 10000 entries, 77 branches, 9.53391551 714355 MB, 1.2977769374847412 sec, 7.346343770133804 M sec, 7.705484441248654 kHz
# 10000 entries, 77 branches, 9.53866761 833008 MB, 1.4104814529418945 sec, 6.7627033726234735  /sec, 7.089777734505208 kHz
--- first pass: 948348 events, (22-flat 52-jagged) branches, 328 attrs
<MLaaS4HEP.reader.RootDataReader objec at 0x7f840dbf4d50> init is complete in 4.8529925346374 1 sec


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

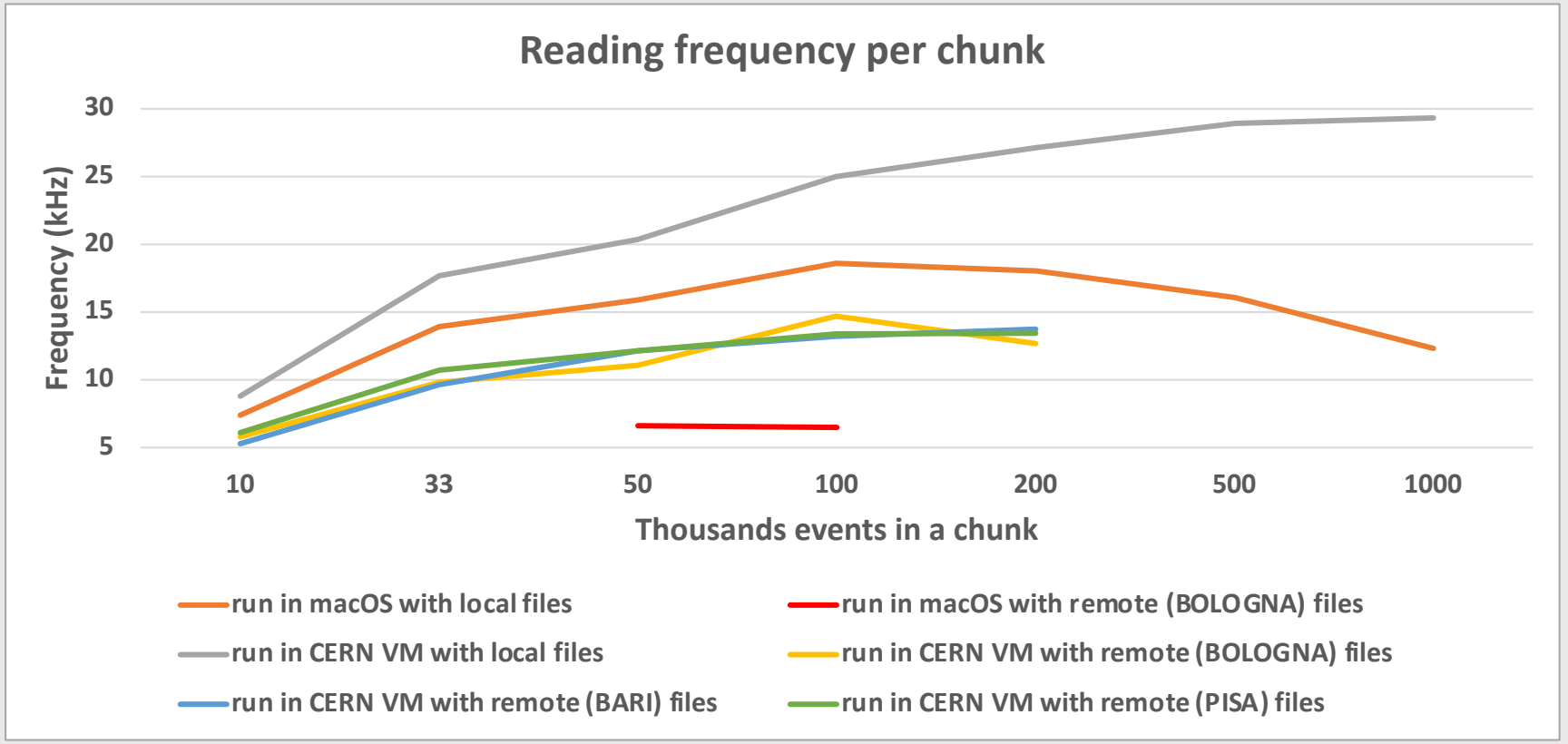
```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 128)               49152
_________________________________________________________________
dropout (Dropout)            (None, 128)               0
_________________________________________________________________
dense_1 (Dense)              (None, 64)                8256
_________________________________________________________________
dropout_1 (Dropout)          (None, 64)                0
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 65
=================================================================
Total params: 57,473
Trainable params: 57,473
Non-trainable params: 0
_________________________________________________________________
```

Init the ML model

Perform train cycle

```
Train on 7000 samples, validate on 3000 samples
Epoch 1/2
7000/7000 [==============================] - 2s 220us/sample - loss: 1.5275 - auc: 0.7845 - accuracy: 0.7307 - val_loss: 2.5731e-04 -
val_auc: 1.0000 - val_accuracy: 1.0000
Epoch 2/2
7000/7000 [==============================] - 0s 20us/sample - loss: 0.1406 - auc: 0.9883 - accuracy: 0.9543 - val_loss: 8.8477e-06 - val_auc:
1.0000 - val_accuracy: 1.0000
```

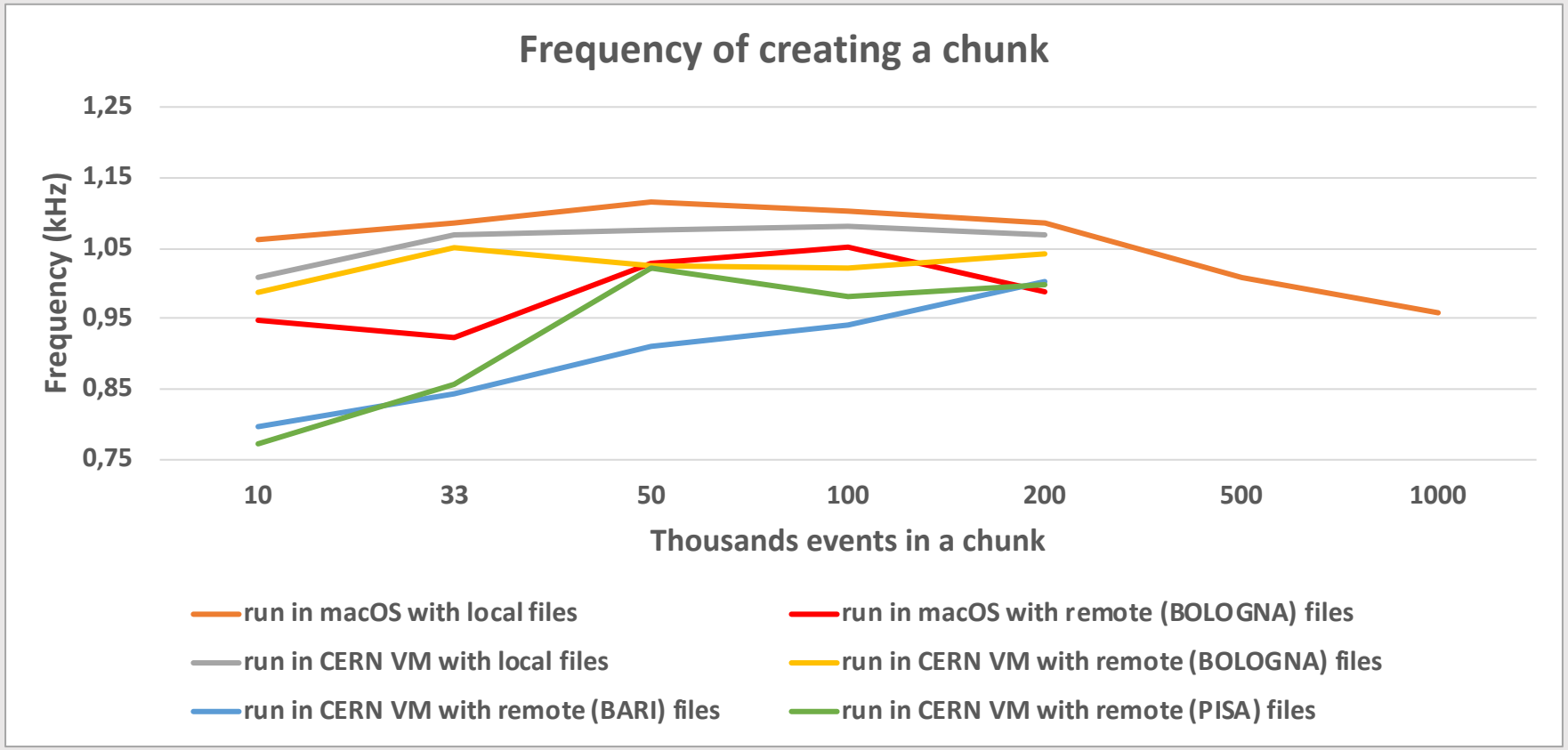
1



	time to go through all the files (s)	reading + specs comp. frequency (kHz)	reading freq. (kHz)	specs comp. freq. (kHz)
macOS with local files	2608	11.2	18.6	28
macOS with remote files (BO)	5453	5.3	6.5	28
VM with local files	2153	13.4	24.9	28.8
VM with remote files (BO)	2994	9.6	14.7	27.7
VM with remote files (BA)	3193	9	13.2	28.5
VM with remote files (PI)	3171	9.1	13.3	28.3

Values for chunk size fixed to 100 thousands events

2



	Frequency for creating a chunk (kHz)	Frequency for handling a chunk (kHz)
macOS with local files	1.1	1.16
macOS with remote files (BO)	1.05	1.19
VM with local files	1.08	1.12
VM with remote files (BO)	1.02	1.08
VM with remote files (BA)	0.94	1.19
VM with remote files (PI)	0.98	1.06

Values for chunk size fixed to 100 thousands events