# Machine Learning as a Service for High Energy Physics on heterogeneous computing resources

**L. Giommi,**[a,*] **V. Kuznetsov,**[b] **D. Bonacorsi**[a] **and D. Spiga**[c]

[a] *University of Bologna and INFN section of Bologna, Bologna, Italy*

[b] *Cornell University, Ithaca, USA*

[c] *INFN section of Perugia, Perugia, Italy*

*E-mail:* luca.giommi3@unibo.it, vkuznet@protonmail.com, daniele.bonacorsi@unibo.it, daniele.spiga@pg.infn.it

*Speaker

Machine Learning (ML) techniques in the High-Energy Physics (HEP) domain are ubiquitous and will play a significant role also in the upcoming High-Luminosity LHC (HL-LHC) upgrade foreseen at CERN: a huge amount of data will be produced by LHC and collected by the experiments, facing challenges at the exascale. Despite ML models are successfully applied in many use-cases (online and offline reconstruction, particle identification, detector simulation, Monte Carlo generation, just to name a few) there is a constant seek for scalable, performant, and production-quality operations of ML-enabled workflows. In addition, the scenario is complicated by the gap among HEP physicists and ML experts, caused by the specificity of some parts of the HEP typical workflows and solutions, and by the difficulty to formulate HEP problems in a way that match the skills of the Computer Science (CS) and ML community and hence its potential ability to step in and help. Among other factors, one of the technical obstacles resides in the difference of data-formats used by ML-practitioners and physicists, where the former use mostly flat-format data representations while the latter use to store data in tree-based objects via the ROOT data format. Another obstacle to further development of ML techniques in HEP resides in the difficulty to secure the adequate computing resources for training and inference of ML models, in a scalable and transparent way in terms of CPU vs GPU vs TPU vs other resources, as well as local vs cloud resources. This yields a technical barrier that prevents a relatively large portion of HEP physicists from fully accessing the potential of ML-enabled systems for scientific research.
In order to close this gap, a Machine Learning as a Service for HEP (MLaaS4HEP) solution is presented as a product of R&D activities within the CMS experiment. It offers a service that is capable to directly read ROOT-based data, use the ML solution provided by the user, and ultimately serve predictions by pre-trained ML models "as a service" accessible via HTTP protocol. This solution can be used by physicists or experts outside of HEP domain and it provides access to local or remote data storage without requiring any modification or integration with the experiment specific framework. Moreover, MLaaS4HEP is built with a modular design allowing independent resource allocation that opens up a possibility to train ML models on PB-size datasets remotely accessible from the WLCG sites without physically downloading data into local storage.
To prove the feasibility and utility of the MLaaS4HEP service with large datasets and thus be ready for the next future when an increase of data produced is expected, an exploration of different hardware resources is required. In particular, this work aims to provide the MLaaS4HEP service transparent access to heterogeneous resources, which opens up the usage of more powerful resources without requiring any effort from the user side during the access and use phase.

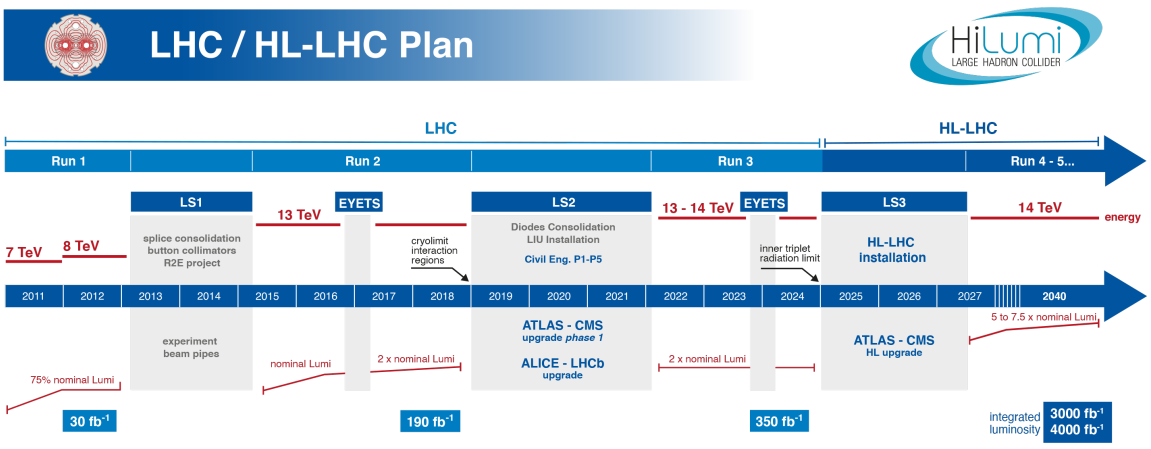

**Figure 1:** The current schedule for LHC and HL-LHC upgrade and run. Currently, the start of the HL-LHC run is foreseen at the end of 2027.

## 1. Introduction

The Large Hadron Collider (LHC) [1] at CERN is the the world's largest and most powerful particle accelerator, and it is the most important experimental apparatus that allows to advance the knowledge in the High Energy Physics (HEP) field. Collisions started to be produced at LHC from March 2010 and on July 2012 the discovery of the Higgs boson [2, 3], predicted by the Standard Model, was announced, representing the major milestone in the history of LHC. During the years we started seeing an acceleration of data growth produced: by the end of Run II, the CERN experiments were already operating at the Peta-Byte (PB) level, producing $O(100)$ PB of data each year. In order to provide more accurate measurements of new particles, enable the observation of rare processes and go beyond the Standard Model and its Higgs boson (e.g. in the search for dark matter and supersymmetry), a more powerful LHC is needed. The new HL-LHC program is currently scheduled to start its data taking at the end of 2027 (see Figure 1) and to operate through 2030s. It foresees an integrated luminosity that is 20 times larger than the current LHC one, bringing quantitatively and qualitatively new challenges for what concerns the datasets produced due to event size, data volume, and complexity.

In this new scenario the usage of Machine Learning (ML) will be crucial [4]. ML techniques have already been successfully used in many areas of HEP, e.g. object reconstruction, particle identification, calibration, triggering, Monte Carlo simulation and beyond [5]. As was pointed out in the ML in HEP Community White Paper [4], this scenario is complicated by a real gap among HEP physicists and ML experts, and one of the main reasons is the difference of data-format used by these two worlds: the former uses the ROOT data-format [6] while the latter represent data in a tabular form (e.g. in CSV [7] files or NumPy [8] arrays). The ROOT data-format is used to store HEP events in tree-based data-structures which cannot be fed directly to existing ML frameworks and special care should be taken in the conversion phase.

To help close this gap and ease the physicists not ML practitioners in the usage of ML techniques in their analyses, we proposed a Machine Learning as a Service for HEP solution [9][1] as a product

[1]At the time of the ISGC conference, the mentioned work has been submitted to a journal and is under peer-review.

of R&D activities within the CMS experiment [11], referred to as MLaaS4HEP in this paper. This solution allows to perform the entire ML pipeline (in terms of reading the data, process the data, training a ML model, serving predictions) directly using ROOT files of arbitrary size from local or remote distributed data-sources and using the ML model of user choice. Its modular design opens up a possibility to train ML models on PB-size datasets remotely accessible from WLCG [12] sites.

To have the compliance for a transparent access to heterogeneous (and potentially more powerful) resources managed by cloud computing providers, and deliver a real Software as a Service (SaaS) [13] solution to the user, we moved towards the cloudification of the MLaaS4HEP framework itself. In particular this work describes the steps we performed to properly use Dynamic On Demand Analysis Service (DODAS) [14] as a Platform as Service (PaaS) [13], in order to obtain an infrastructure that works with cloud resources and where MLaaS4HEP can be used without any effort by user side.

## 2. State of art

Machine Learning as a Service (MLaaS) [15] is a well known concept in industry and it is an umbrella definition for 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 offer MLaaS services, e.g. Amazon [16], Microsoft [17], Google [18], IBM [19]. But there are some difficulties in using directly these services with HEP data, which are in the ROOT data-format. Indeed in most of the cases ML frameworks deal with either CSV or NumPy arrays representing tabular data and thus, if the user want to use ROOT data, a first conversion step is needed. Moreover the HEP physicists in general use some pre-processing operations that may be more complex than the ones offered by the aforementioned service providers. Nowadays there are examples of R&D solutions (e.g. hls4ml [20], which is specialized on speeding-up the inference on FPGAs), but they are generally targeted to a specific step of the ML pipeline and not on the entire ML pipeline itself, i.e. from reading data, to processing data, training the model and serving predictions. Moreover there are existing custom solutions in specific CMS analyses (e.g. [21]), but they cannot easily generalized and do not represent "as a Service" solutions. There is also a recent solution for data processing and ML training with the Spark platform [22], but here data are read from the CERN EOS storage system not allowing access to data stored in WLCG sites.

## 3. Machine Learning as a Service for HEP

We proposed our solution of MLaaS for HEP [23] which allows to:

- natively read HEP data, that means being able to read ROOT files of arbitrary size from local or remote distributed data-sources via XrootD [24];
- use heterogeneous resources both for training and inference, like local CPU, GPUs, farms, cloud resources, etc.;
- use different ML libraries and frameworks of user choice, like Keras [25], TF [26], PyTorch [27], fast.ai [28], etc.;

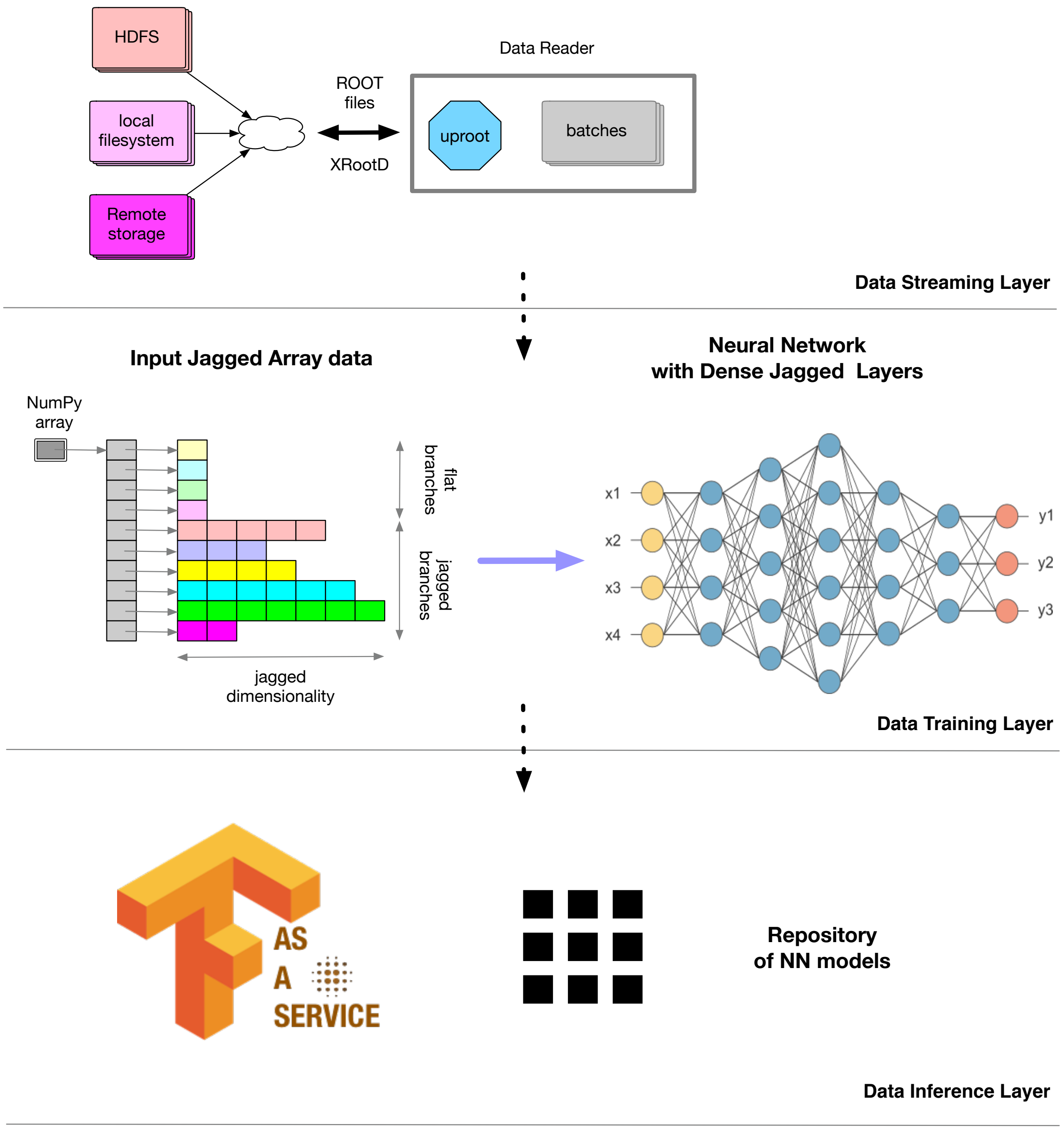

**Figure 2:** MLaaS4HEP architecture diagram representing the three independent layers: Data Streaming Layer (top), Data Training Layer (middle), and Data Inference Layer (bottom) [9].

- serve pre-trained HEP models, like a models repository, and access it easily from any place, any code, and any framework.

MLaaS4HEP is structured in three different and independent layers which allow independent resource allocation: Data Streaming, Data Training and Data Inference layers (see Fig 2). In the following subsections we provide some details on MLaaS4HEP architecture and its performance.

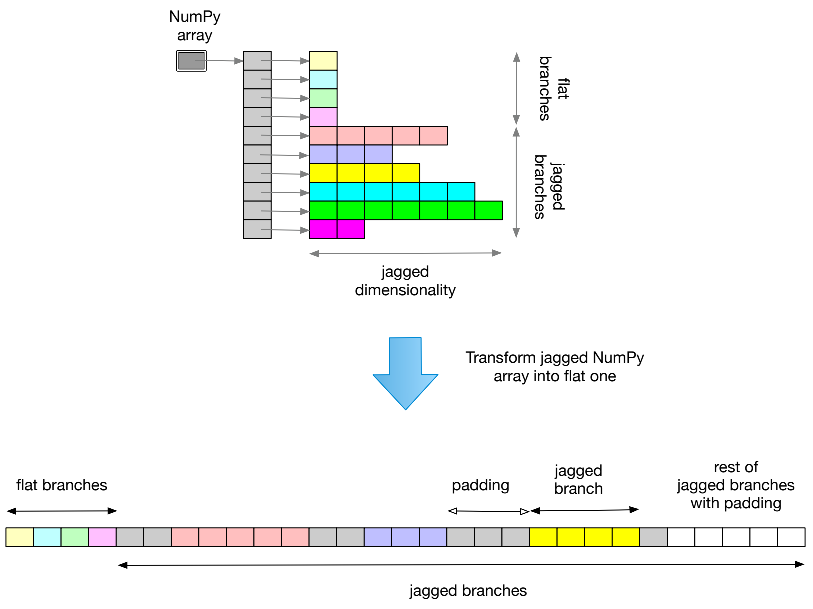

**Figure 3:** A vector representation of a Jagged Array with padded values [9].

## 3.1 Data Streaming Layer

The main development of the Data Streaming Layer was done using the DIANA-HEP [29] uproot [30] library, providing the ability to read ROOT data in Python, access them as NumPy arrays, and it implements XrootD access to read remote files. In order to properly feed data into the chosen ML framework, we wrote a Python Generator able to read ROOT files and deliver data as chunks, which size can be defined directly by the user. Such implementation provides efficient access to large datasets since it does not require loading the entire dataset into the RAM of the training node. It also allows to read the events in a proportional way from different input ROOT files. The output of the Python Generator are NumPy arrays with flat and Jagged Array attributes. To clarify, ROOT files have a tree structure where typically flat branches and not-flat branches are stored: the former stores simple values (e.g. integer or floating-point numbers) while the latter stores vectors which dimension can vary along the branch. The uproot library converts flat branches in NumPy arrays, while non-flat branches in a compact format called Jagged Array.

## 3.2 Data Training Layer

The aim of the Data Training Layer is to process the read data and use them to train the ML model chosen by the user. Firstly the Jagged Array attributes need to be properly converted and we opted to flatten them into fixed-size arrays with padding values (see Figure 3) through a two step procedure. We scan all the events of the input ROOT files to compute the maximum dimension of the Jagged Array attributes for each Jagged branch; then we update the dimension of these attributes to the maximum one computed previously, filling the additional dimensions with padding values, which we assigned as NANs since all other numerical values can represent attribute spectrum. We also keep a separate masking vector to save the location of padding values. In addition we provide a proper normalization procedure of each attribute, that can be decoded directly by the user. Finally

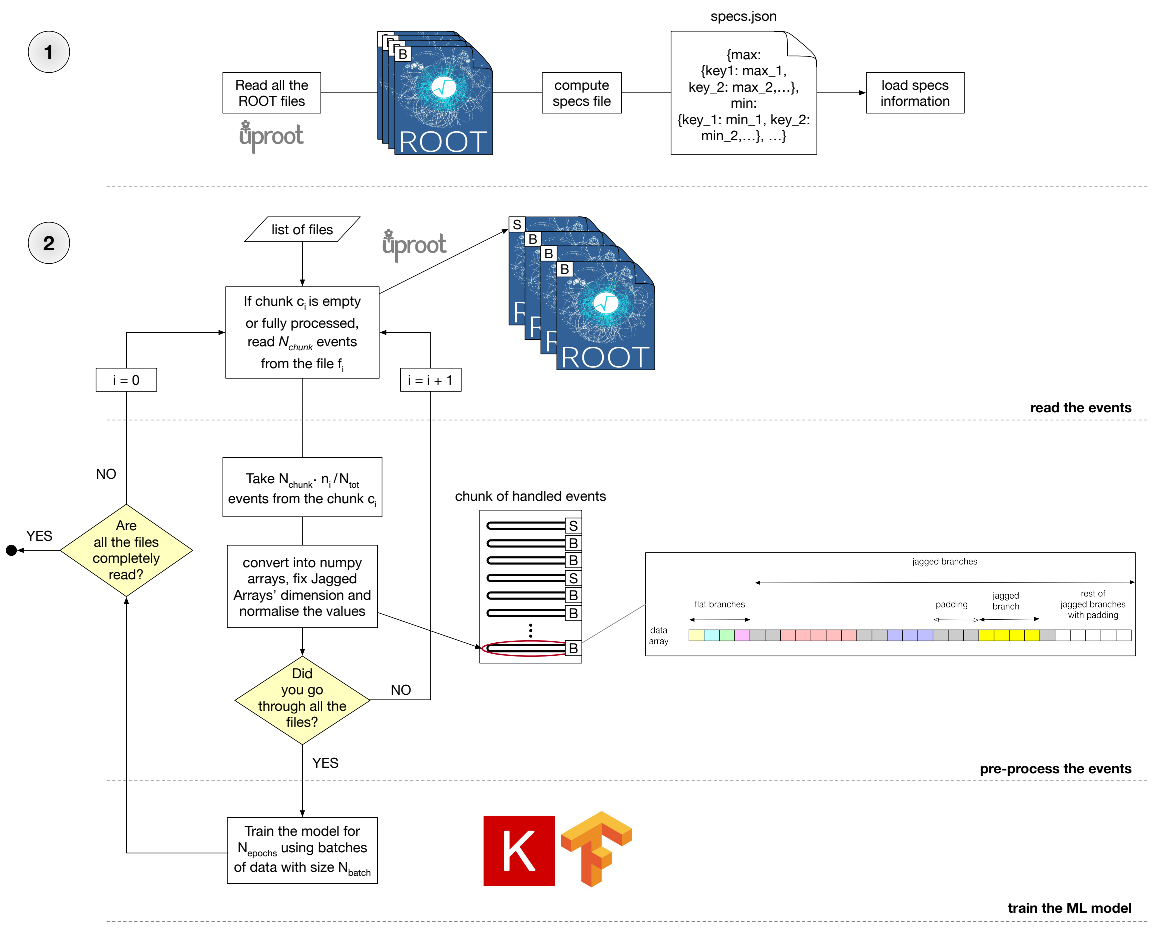
1
Read all the ROOT files
uproot
ROOT
compute specs file
specs.json
{max: {key1: max_1, key_2: max_2,...}, min: {key_1: min_1, key_2: min_2,...}, ...}
load specs information
2
list of files
uproot
If chunk $c_i$ is empty or fully processed, read $N_{chunk}$ events from the file $f_i$
S
B
ROOT
i = 0
i = 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
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

**Figure 4:** Schematic representation of the steps performed in the MLaaS4HEP pipeline, in particular those inside the Data Streaming and Data Training layers (see text for details) [9]. In the picture $N_{chunk}$ stands for the chunk size, $n_i$ for the number of events in the file $f_i$, $N_{tot}$ for the whole amount of events stored in the ROOT files.

the ML model chosen by the user is trained. In the next subsection we provide details of the MLaaS4HEP workflow that combines the Data Streaming and Data Training layers.

## 3.3 MLaaS4HEP training workflow

We implemented the Data Streaming and Data Training layers using the Python programming language. Together contribute to the MLaaS4HEP training workflow, which consists in two different steps schematized in Figure 4. In the first step (denoted by ① in Figure 4) the input ROOT files are read in chunk (which size is fixed a priori by the user) and a specs file is computed. This file contains useful information about the ROOT files, e.g. the maximum dimension of the attributes in Jagged branches, the minimum and maximum values for each branch. In the second step, shown as ②, the ML training phase is performed. It consists in a loop where each time the right proportion of events (the same presented in the ROOT files) is read from each file. Then the events are processed, that means converted into NumPy arrays, with the necessary fix of the Jagged Arrays dimensions and normalization of the values. This part is performed using the information contained in the specs file computed previously. Finally the chunk is used to train the ML model chosen by the user, for a number of epochs and a batch size fixed a priory by the user itself. The loop continue, creating new chunks of events to use for the training, until all the ROOT files are completely read. At the end of the cycle, the training process of the model is completed and the model produced can be used in physics analysis.

An important aspect of MLaaS4HEP is its being ML model and framework agnostic, that means the user can choose the desired framework and provide directly the definition of the model to use in the training phase.

The MLaaS4HEP training workflow is performed running the *workflow.py* script:

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

The *workflow.py* script takes several arguments:

- *files.txt* stores the path of the input ROOT files;
- *labels.txt* stores the labels of the input ROOT files in case of classification problems;
- *model.py* stores the definition of the custom ML model to use in the training phase, in the user's favorite ML framework;
- *params.json* stores other parameters, e.g. number of events to use, chunk size, batch size, redirector path for files located in remote storage.

### 3.4 Data Inference Layer

The Data Inference Layer is implemented as TensorFlow as a Service (TFaaS) [31], written in the Go programming language [32] and based on HTTP protocol. We chose the Go language because it natively supports concurrency and it is very well integrated with the TF library. A TFaaS server (e.g. [33]) can be used as a global repository of pre-trained TF models and serve predictions using the loaded TF models. It is experiment agnostic and can work with any HTTP based client, in particular both Python and C++ clients have been developed on top of REST APIs (end-points). With TFaaS the clients can test multiple TF models at the same time allowing a rapid development or continuous training of TF models, and their validation.

### 3.5 MLaaS4HEP and TFaaS performance

As discussed in [9], we tested MLaaS4HEP performance using a dataset of a signal vs background discrimination problem in a $t\bar{t}H$ analysis, containing roughly 28.5M events and with a total size of roughly 10 GB. We used two different platforms, a macOS, 2.2 GHz Intel Core i7 dual-core, 8 GB of RAM laptop and a 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 (SSD storages) and remotely from Grid sites. In particular, we read files remotely from three different WLCG data-centers located in Italy (in Bologna, Pisa and Bari). The performance results are shown in Table 1, where the chunk size was fixed to 100k events. The time to train the ML model is not included in the performance: it is independent from the MLaaS4HEP framework while depends on the underlying ML framework, the complexity of the used ML model, and the available hardware

| | event throughput | total time using all the 28.5M events |
|---|---|---|
| specs computing phase | 8.4k - 13.7k evts/s | 35 - 57 min |
| chunks creation in the training phase | 1.1k - 1.2k evts/s | 6.5 - 7.5 hrs |

**Table 1:** Mean values of event throughput and total time spent using all the 28.5 events for the specs computing phase and for the chunks creation in the training phase. Intervals of values are reported in table, where the upper limit for the event throughput and the lower limit for the total time are reached using local ROOT files, while the opposite limits using remote files. No relevant changes were found using different data-centers where remote files were stored, or changing the platform where run MLaaS4HEP.

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 $O(1\text{k})$ hours (plus the time required to train the ML model). Further optimization of the MLaaS4HEP pipeline will be required to process TB or PB-size datasets and it may involve parallelization of I/O, distributed ML training, etc.

In order to test the TFaaS performance we did benchmarks on CentOS 7 Linux, 16 cores, 30 GB of RAM in two modes: using 1k calls with 100 concurrent clients and using 5k calls with 200 concurrent clients. In both cases, we achieved a throughput of 500 req/sec. These numbers were obtained by serving a mid-size pretrained Keras NN model with 27 features and 1024x1024 hidden layers used in the aforementioned physics analysis. Similar performance was found for the MNIST [34] classification problem. The actual performance of TFaaS will depend on the complexity of served ML model and available hardware resources, and to provide the desired throughput for concurrent clients TFaaS can be horizontally scaled, e.g. using Kubernetes [35].

## 4. MLaaS4HEP cloudification

As pointed out in the previous subsection, MLaaS4HEP and TFaaS performance strictly depend on the available hardware resources. These tests give us the idea of MLaaS4HEP performance but, even if it can be sufficient for datasets of a size like that of our case, we have to improve it if we plan to use our solution with larger datasets (key point in the HL-LHC phase). This improvement can be reached exploring different ways, like adopting new solutions in the code, investing in better and more expensive on-premise resources, or moving to the cloud. The operation of cloudification has two benefits: it opens us to potentially more performing resources but also provides a real “as a Service” solution to the user. For these reasons, we decided to move towards the MLaaS4HEP cloudification and we opted to use the DODAS service. If DODAS allow us to provide a platform and we properly integrate our MLaaS4HEP framework, we deliver to users a SaaS solution (see Figure 5).

In the following we provide a description of DODAS with its services and of all the steps we followed to create our SaaS.

### 4.1 DODAS

DODAS is a PaaS tool for generating container based solutions over cloud and on-demand resources. It is designed for scientists seeking to easily exploit distributed and heterogeneous cloud

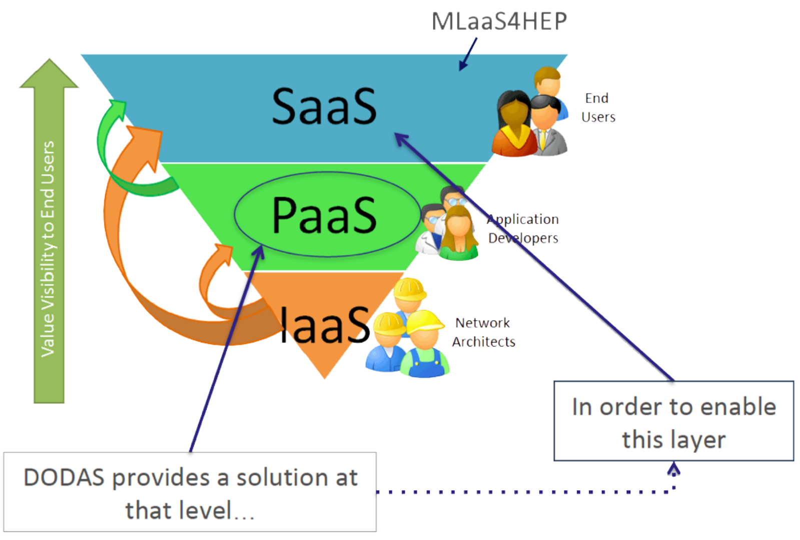
MLaaS4HEP
SaaS
PaaS
IaaS
End Users
Application Developers
Network Architects
Value Visibility to End Users
In order to enable this layer
DODAS provides a solution at that level…

**Figure 5:** Schematic representation of the levels of service. DODAS provides a solution at the PaaS level, and integrating the MLaaS4HEP framework we obtain a SaaS solution.

resources, automatizing the process of provisioning, creating, managing and accessing resources. Its development started in the context of INDIGO-DataCloud [36] project and its architecture was realized mostly using INDIGO building blocks. It implements a services composition model based on templates, where services can be chosen depending on the use case thus providing a customizable platform. The DODAS architectural pillars are listed below.

- The PaaS Orchestrator [37] is the DODAS entrypoint where user's requests for application or service deployments, typically expressed in a TOSCA [2] [38] template, are taken in charge. It has the role of providing the best infrastructure for the deployment considering the requests of the users, the availability and the health status of the IaaS services. The interaction with the infrastructure is delegated to the Infrastructure Manager.

- The Infrastructure Manager (IM) [39] is in charge to deploy customized virtual infrastructures on different IaaS Cloud deployments, allowing the support for several cloud providers (e.g OpenStack [40], Amazon AWS [41], Microsoft Azure [42]). It eases the access and the usability of IaaS clouds by automating the VMI (Virtual Machine Image) selection, deployment, configuration, software installation, monitoring and update of the virtual infrastructure.

- The Identity and Access Management service (IAM) [43] manages identities, enrollment, group membership, attributes and policies to access distributed services and resources. It supports the federated authentication mechanisms behind the INDIGO Authentication and Authorization Infrastructure (AAI). Users can authenticate with any of the supported mechanisms (SAML [44], OpenID Connect [45], X.509 certificates [46], local username/password), also using the credentials provided by existing Identity Federations (e.g. IDEM [47], eduGAIN [48]).

[2]Topology and Orchestration Specification for Cloud Applications (TOSCA) is an OASIS open standard to define the topology of services provisioned in IT infrastructures.

The resource abstraction and the full automation are implemented combining together the PaaS Orchestrator and the IM. In this way the end users are able to exploit computational resources without knowing the IaaS details. The cluster setup and the services configuration are automated using Ansible [49] recipes. The TOSCA and Ansible combination allows an easy procedure to describe complex computing infrastructures.

### 4.2 Using DODAS for MLaaS4HEP claudification

For the MLaaS4HEP cloudification with DODAS we followed a series of step [50]:

- create a Docker [51] image able to run the MLaaS4HEP pipeline (i.e. run the *workflow.py* script);
- create an Ansible role to automatize the configuration and deployment of the container with dependencies (e.g. download Docker, pull the Docker image, run a container);
- create a Tosca template to define the resource requirements (e.g. number of CPUs and memory size of the Virtual Machine we want to use) and the input parameters for the creation of the docker container (e.g. define a folder of the host file system to connect with the container);
- create the deployment from command line.

Thus we used DODAS to automatize the deployment via command line interface and in this way the user is equipped with a platform where run the *workflow.py* script. Moreover we used DODAS to manage the system via JupyterHub [52] interface which can be very useful in research groups because it provides a user-friendly solution with a shareable Jupyter Notebook [53], i.e. the user can work in a shared environment with the colleagues using the same cloud interface. It integrates a cloud storage for managing the required files (e.g. ROOT files, ML models) which can be shared with other users. It implements a token-based access using the IAM service and it has the support for a customizable environment, i.e. we can personalize and use the proper Docker image as needed, see Figure 6. Using this interface the user can simply open a terminal and run the *workflow.py* script without any effort[3].

## 5. Conclusions

The need of HEP to fully exploit the potential of ML techniques will be crucial in the next years, in particular during the HL-LHC phase when a significant increase of data production is expected. To ease the rise of ML in HEP and reducing the gap between these two worlds, we proposed the MLaaS4HEP solution [9] where ML models of user's choice can be trained using directly local or remote ROOT files without the need of a data-format conversion phase from user side. MLaaS4HEP performance strictly depends on the available hardware resources and its usage with TB or PB-size datasets needs additional improvements, both at code level integrating new solutions and exploring cloud solutions that can guarantee the usage of more performing resources.

[3]In the [50] repository the reader can find both all the material used for the MLaaS4HEP cloudification with DODAS and a Demo that shows how the user can access and use the JupyterHub interface to run the *workflow.py* script.

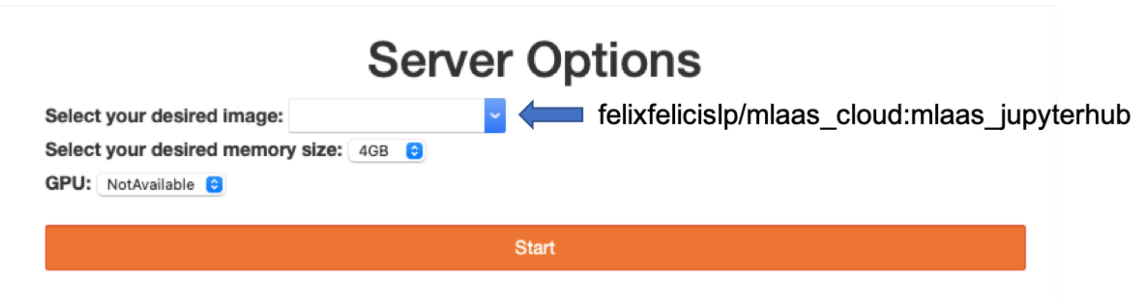
Server Options
Select your desired image:
felixfelicislp/mlaas_cloud:mlaas_jupyterhub
Select your desired memory size: 4GB
GPU: NotAvailable
Start

**Figure 6:** JupyterHub interface where the user can choose the desired image to run the server. In particular we created and used the *felixfelicislp/mlaas_cloud:mlaas_jupyterhub* image stored in Dockerhub [54].

The work presented in this paper is focused on the MLaaS4HEP cloudification using DODAS, that allowed us to have a real "as a Service" solution, where the user is equipped on-demand of an infrastructure in which the MLaaS4HEP workflow can be run without any effort. In this way the user, in addition to use the framework with his/her local resources that can be limited, has also the possibility to use the heterogeneous resources that cloud providers can offer. At the same time we also provided a jupyterhub interface to have an user-friendly access to the platform where run the code and share the material with other users.

We performed some functional tests of this solution using an Ubuntu 18 Linux, 8 AMD Opteron 62xx class CPU 2.6 GHz, 16 GB RAM VM, proving its functionality. Nevertheless additional improvements are planned to be integrated to fully exploit the potentiality of cloud resources, in particular through the distribution of MLaaS4HEP workflow on a cluster of resources. This will entail to parallelize as much as possible parts of MLaaS4HEP, e.g. the reading of input ROOT files and the ML model training phase.

The choice of using DODAS as PaaS was mostly driven by our closeness with the DODAS working group and our easiness to use its resources. At the same time the work done so far provides compliance with the INFN-Cloud [55] portfolio of services, which opens up the possibility of letting use our framework "as a Service" by users of INFN-Cloud.

## References

[1] O. Bruning, H. Burkhardt and S. Myers, *Large Hadron Collider*, Prog. Part. Nucl. Phys. **67** (2012) 705-734

[2] ATLAS Collaboration, *Observation of a New Particle in the Search for the Standard Model Higgs Boson with the ATLAS Detector at the LHC*, Phys. Lett. B **716** (2012) 1-29 [hep-ex/1207.7214]

[3] CMS Collaboration, *Observation of a New Boson at a Mass of 125 GeV with the CMS Experiment at the LHC*, Phys. Lett. B **716** (2012) 30-61 [hep-ex/1207.7235]

[4] K. Albertsson et al, *Machine Learning in High Energy Physics Community White Paper*, J. Phys. Conf. Ser. **1085** (2018) no.2 022008 [physics.comp-ph/1807.02876]

[5] *A Living Review of Machine Learning for Particle Physics*. https://iml-wg.github.io/HEPML-LivingReview/

[6] *A modular scientific toolkit used in HEP for analysis and as a data-storage format*. https://root.cern.ch

[7] *Comma Separated Values (CSV) data-format*. https://www.wikiwand.com/en/Comma-separated_values

[8] *Scientific package to represent data as multi-dimensional arrays*. http://www.numpy.org

[9] V. Kuznetsov, L. Giommi, D. Bonacorsi, *Machine Learning as a Service for HEP* hep-ex/2007.14781

[10] CMS Collaboration, *The CMS Experiment at the CERN LHC*, JINST **3** (2008) S08004

[11] *WLCG Project*. http://www.cern.ch/lcg

[12] P. Mell and T. Grance, *The NIST Definition of Cloud Computing*, National Institute of Standards & Technology (2011)

[13] D. Spiga et al., *DODAS: How to effectively exploit heterogeneous clouds for scientific computations*, in proceedings of *International Symposium on Grids and Clouds (ISGC) 2018 in conjunction with Frontiers in Computational Drug Discovery*

[14] *What's machine learning as a service or MLaaS?* https://www.topbots.com/comprehensive-guide-to-mlaas/

[15] *Machine Learning on AWS*. https://aws.amazon.com/machine-learning/

[16] *Azure Machine Learning*. https://azure.microsoft.com/en-in/services/machine-learning/

[17] *AI and machine learning products*. https://cloud.google.com/products/ai/

[18] *IBM Watson Machine Learning*. https://www.ibm.com/cloud/machine-learning

[19] *A package for machine learning inference in FPGAs*. https://fastmachinelearning.org/hls4ml/

[20] CMS Collaboration, *A deep neural network to search for new long-lived particles decaying to jets*, Mach. Learn.: Sci. Technol. **1** (2020) 035012 [hep-ex/1912.12238]

[21] M. Migliorini, R. Castellotti, L. Canali and M. Zanetti, *Machine Learning Pipelines with Modern Big Data Tools for High Energy Physics*, Comput. Softw. Big Sci. **4** (2020) no.1, 8 [cs.DC/1909.10389]

[22] V. Kuznetsov, L. Giommi, *MLaaS4HEP is set of MLaaS components for HEP*. https://zenodo.org/badge/latestdoi/156857396, https://github.com/vkuznet/MLaaS4HEP

[23] *A high performance, scalable fault tolerant access to data repositories of many kinds*. http://xrootd.org

[24] *Keras AI library*. https://keras.io

[25] *Tensor Flow AI library*. http://www.tensorflow.org

[26] *PyTorch AI library*. https://www.pytorch.org

[27] *Fast AI library*. https://www.fast.ai

[28] *An umbrella organization for bringing state-of-the art for HEP experiments*. http://diana-hep.org

[29] *DIANA-HEP Scikit-hep uproot library. Minimalist ROOT I/O in pure Python and NumPy*. https://github.com/scikit-hep/uproot

[30] V. Kuznetsov, *TensorFlow as a Service*. https://zenodo.org/badge/latestdoi/109141847, http://github.com/vkuznet/TFaaS

[31] *Go programming language*. http://www.golang.org

[32] *A TFaaS demo server hosted by CERN*. https://cms-tfaas.cern.ch/

[33] *The MNIST database of handwritten digits*. http://yann.lecun.com/exdb/mnist/

[34] *Kubernetes: Production-Grade Container Orchestration*. https://kubernetes.io

[35] D. Salomoni et al., *INDIGO-Datacloud: foundations and architectural description of a Platform as a Service oriented to scientific computing*, [cs.SE/1603.09536]

[36] *Orchestrator of the PaaS layer*. https://indigo-dc.gitbook.io/indigo-datacloud-releases/chapter2/all/orchestrator2

[37] *OASIS Topology and Orchestration Specification for Cloud Applications (TOSCA) TC*. https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=tosca

[38] *Infrastructure Manager*. https://indigo-dc.gitbook.io/im/

[39] *OpenStack*. https://www.openstack.org/

[40] *Amazon AWS*. https://aws.amazon.com

[41] *Microsoft Azure*. https://azure.microsoft.com/

[42] *INDIGO Identity and Access Management (IAM) service*. https://indigo-iam.github.io/docs/v/current/

[43] *Security Assertion Markup Language (SAML)*. https://docs.oasis-open.org/security/saml/Post2.0/sstc-saml-tech-overview-2.0.html

[44] *OpenID Connect*. https://openid.net/connect/

[45] *Internet X.509 Public Key Infrastructure Certificate*. https://www.ietf.org/rfc/rfc5280.txt

[46] *IDEM GARR AAI*. https://www.idem.garr.it

[47] *eduGAIN*. https://edugain.org

[48] *Ansible*. https://www.ansible.com/

[49] *Material for MLaaS4HEP cloudification with DODAS*. https://github.com/lgiommi/mlaas_cloud

[50] *Docker*. https://www.docker.com

[51] *JupyterHub*. https://jupyterhub.readthedocs.io/en/stable/

[52] *Jupyter Notebook*. https://jupyter.org

[53] *felixfelicislp/mlaas_cloud:mlaas_jupyterhub Docker image*. http://bit.ly/mlaas_jupyterhub

[54] *INFN-Cloud* https://www.cloud.infn.it