Progress on cloud native solution of Machine Learning as a Service for HEP

Luca Giommi[1,*], *Daniele* Spiga[2], *Valentin* Kuznetsov[3], and *Daniele* Bonacorsi[4,5]

[1]INFN-CNAF, Viale Carlo Berti Pichat, 6/2, 40127 Bologna (ITALY)
[2]INFN Sezione di Perugia, Via Alessandro Pascoli 23c, 06123 Perugia (ITALY)
[3]Cornell University, 616 Thurston Ave., Ithaca, NY 14853 (USA)
[4]University of Bologna, Via Zamboni 33, 40126 Bologna (ITALY)
[5]INFN Sezione di Bologna, Viale Carlo Berti Pichat, 6/2, 40127 Bologna (ITALY)

Abstract. Nowadays Machine Learning (ML) techniques are successfully used in many areas of High-Energy Physics (HEP) and will play a significant role also in the upcoming High-Luminosity LHC upgrade foreseen at CERN, when a huge amount of data will be produced by LHC and collected by the experiments, facing challenges at the exascale. To favor the usage of ML in HEP analyses, it would be useful to have a service allowing to perform the entire ML pipeline (in terms of reading the data, processing data, training a ML model, and serving predictions) directly using ROOT files of arbitrary size from local or remote distributed data sources. The Machine Learning as a Service for HEP (MLaaS4HEP) solution we have already proposed aims to provide such kind of service and to be HEP experiment agnostic. To provide users with a real service and to integrate it into the INFN Cloud, we started working on MLaaS4HEP cloudification. This would allow to use cloud resources and to work in a distributed environment. In this work, we provide updates on this topic and discuss a working prototype of the service running on INFN Cloud. It includes an OAuth2 proxy server as authentication/authorization layer, a MLaaS4HEP server, an XRootD proxy server for enabling access to remote ROOT data, and the TensorFlow as a Service (TFaaS) service in charge of the inference phase. With this architecture a HEP user can submit ML pipelines, after being authenticated and authorized, using local or remote ROOT files simply using HTTP calls.

1 Introduction

The combined operations of the Large Hadron Collider (LHC) experiments yield approximately 200 PB of data annually, necessitating to be stored, processed, and analyzed. To enable physicists spread all over the world to access the required computing power and storage, CERN employs a grid-based network called Worldwide LHC Computing Grid (WLCG). The upcoming High Luminosity LHC (HL-LHC) program, scheduled to start in 2029, will amass around 1 EB of data yearly from ATLAS and CMS, to which derived and simulated

*e-mail: luca.giommi@cnaf.infn.it

data is added. This presents a significant challenge, as each experiment will enter a multi-Exabyte per year data management regime. In this scenario, the role of Machine Learning (ML) in High Energy Physics (HEP) will be critical.

ML techniques have found success across various areas of HEP, including online and offline reconstruction, detector simulation, object identification, and Monte Carlo generation, among others. However, developing and implementing ML projects for practical use is time-intensive, demanding specific skills, and HEP analysts often lack the necessary data science expertise to tackle these challenges independently. Compounding this issue is the gap between the HEP and ML communities. This is partially due to the prevalent use of the ROOT [1] data format within HEP, which remains largely unfamiliar outside this domain. HEP data relies on tree-based structures, where the event size is unpredictable (e.g. the particle count may vary in each physics event) so that careful consideration is needed when using ROOT data with ML frameworks. Therefore, offering a service to HEP physicists, particularly a Machine Learning as a Service (MLaaS), could facilitate non-expert users in harnessing the capabilities of ML. Such an approach could bridge the gap, promoting broader adoption of ML techniques in HEP analyses.

We have developed a MLaaS solution for HEP, already discussed in previous works [2–5], representing a cloud service that enables HEP users to execute ML pipelines using HTTP calls. These pipelines utilize the MLaaS for HEP (MLaaS4HEP) framework, which allows direct reading, processing, and ML model training using ROOT files of any size from local or distributed data sources. The inference aspect is managed by the Tensorflow as a Service (TFaaS) tool, which hosts pre-trained Tensor-based ML models and allows to get predictions through HTTP calls.

In this work, we provide updates on this solution and discuss a working prototype of the service running on INFN Cloud [6, 7]. Such a prototype can be created through the INFN Cloud Orchestrator Dashboard by using a proper docker-compose file to manage the deployment of the required services.

2 The developed MLaaS solution for HEP

Initially, the MLaaS solution for HEP we developed was composed of the MLaaS4HEP framework and TFaaS, where the former is a Python software and the latter an HTTP service. Over the years the MLaaS4HEP framework has been updated and new features have been added, as well a cloud native solution has been developed for it. In the following, we provide a description of the MLaaS4HEP framework, in its original implementation, and TFaaS, then we provide details on the developments we made subsequently.

2.1 The MLaaS4HEP framework and training workflow

The MLaaS4HEP framework has been developed using the Python programming language, and its code is accessible on the GitHub repository [8]. MLaaS4HEP facilitates the real-time streaming of HEP datasets, stored in the event tree-based ROOT data format, into various Python-based ML frameworks preferred by users which typically are designed to work with row-based data structures, like NumPy [9] arrays and CSV files. MLaaS4HEP imposes a specific constraint on the input ROOT files it accepts: they must have a flat ROOT TTree structure, devoid of embedded C++ objects and without any nested arrays within branch elements.

The reading component of MLaaS4HEP was developed using the Uproot library [10] version 3 (Uproot3), which employs NumPy calls to efficiently convert data blocks from

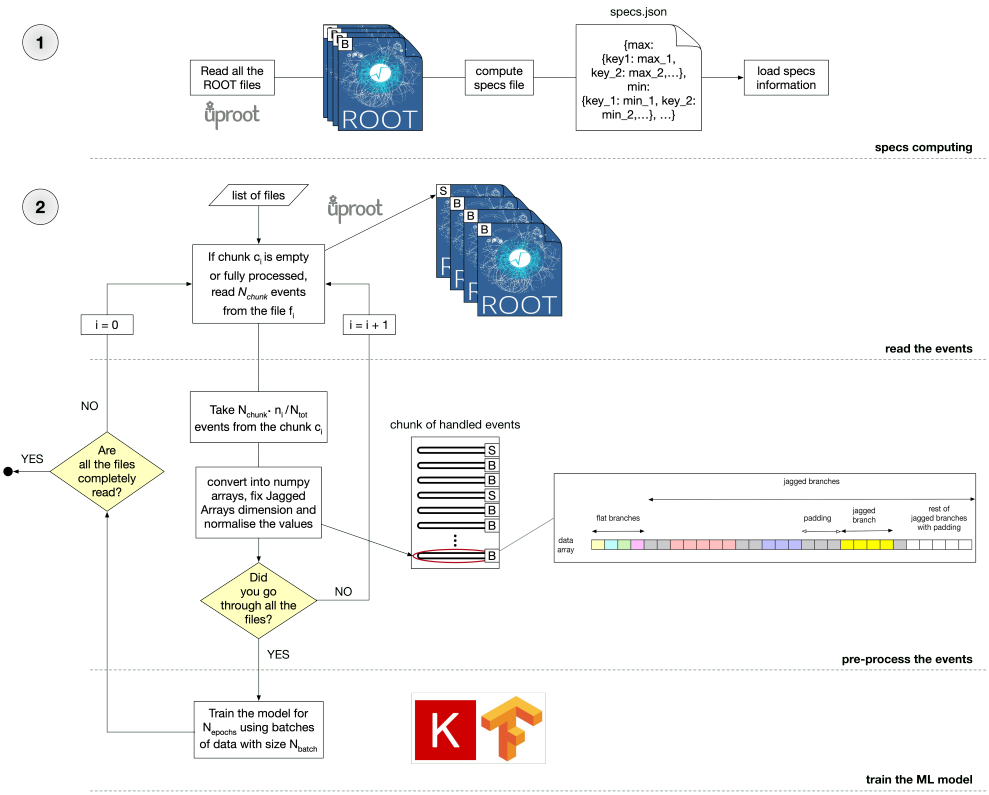

Figure 1. Schematic representation of the steps performed in the MLaaS4HEP training workflow [2].

ROOT files into NumPy arrays. This data can be sourced from local ROOT files or remotely accessed, including from WLCG sites, using the XRootD [11] protocol. MLaaS4HEP transforms the Jagged Array representation of ROOT data and feeds it into the ML framework via vector transformations applied to the I/O stream. A Jagged Array is a compact representation of variable-size event data generated in HEP experiments. In MLaaS4HEP, a Python generator that can read chunks of data and gives as output a NumPy array with flat and Jagged Array attributes is implemented. Such an implementation provides efficient access to large datasets, as the entire dataset may not be loaded into the RAM of the training node, and it can be used to parallelize the data flow into the ML workflow pipeline. Then MLaaS4HEP takes care to transform HEP ROOT data presented as Jagged Array into a flat data format used by ML frameworks. For that, a two-passes procedure has been implemented. In the first pass across all the events, the maximum dimensionality of each Jagged Array attribute and the min/max values of each attribute are determined. In the second pass, the Jagged Array attributes are mapped into a single vector representation with appropriate size (maximum dimension computed for each attribute) and padding (e.g. using NaN values or zeros). In addition, a proper normalization of each attribute is provided during this phase.

Finally, the MLaaS4HEP framework uses data chunks, with the proportion of events presented in the input ROOT files, to train the ML algorithms which definition is provided by the user code. The data flow schema used in the MLaaS4HEP training workflow when a Neural Network (NN) is chosen as ML algorithm is reported in Fig. 1.

2.2 TFaaS architecture

The inference layer, which complements the MLaaS4HEP framework, is represented by the TFaaS service based on the HTTP protocol, and its code is accessible on the GitHub repository[12]. TensorFlow graphs were selected because of their versatility across different programming languages, optimization tailored for GPUs and TPUs, and the backing they receive from the TensorFlow library. TFaaS was written using the Go programming language, leveraging its inherent concurrency support, seamless integration with TensorFlow, and streamlined deployment process. Clients can upload TensorFlow models and access them via REST APIs, making the framework versatile and usable for a wide range of applications, including those outside of HEP.

2.3 Further developments on the MLaaS4HEP framework

In the following, the main updates made on the original version of the MLaaS4HEP framework are reported.

- The MLaaS4HEP framework has been updated to support version 4 of Uproot (Uproot4) in addition to Uproot3. This was necessary because there has been a breaking point for the MLaaS4HEP compatibility with the Uproot library with the transition from Uproot3 to Uproot4 in the second half of 2020. Thanks to this update, it was also possible to introduce pre-processing operations to data in MLaaS4HEP. Now the user can provide a proper configuration file (in JSON format) with information to create new branches and apply cuts both on existing and new branches.
- MLaaS4HEP has been originally used only with Multi Layer Perceptrons (MLPs) written in Keras [13]. Subsequently, we have abstracted it to support any kind of Python-based ML algorithm and framework. MLaaS4HEP has been successfully tested using: MLP written in Keras and PyTorch [14]; MLP, Gradient Boosting, AdaBoost, Random Forest, Decision Tree, kNN, SVM, and Logistic Regression written in Scikit-learn [15]; Gradient Boosting written in XGBoost [16].
- We have added in MLaaS4HEP the possibility to print some of the most common metrics in ML, i.e. Confusion Matrix, AUC, Precision, Recall, and F1.
- We have added a new training procedure for ML models. Considering the scenario of a NN, the original MLaaS4HEP training procedure is performed chunk by chunk, where each chunk is used one at a time to train the model for N_{epochs} epochs, whereas in the new training procedure, all the chunks are used in each epoch. The original MLaaS4HEP training approach is useful when the dataset is large and exceeds the amount of RAM of the training node since only a chunk is stored at a time in the memory and the chunk size can be adjusted to fit in the hardware resources. However, the user should carefully evaluate the ML model convergence and validate it after each chunk. Conversely, the new MLaaS4HEP training approach, which uses the entire dataset for each epoch, can guarantee the ML model convergence, but the dataset should fit into RAM.

3 Cloud native solution of MLaaS4HEP

To provide a real MLaaS solution for MLaaS4HEP, a cloud native application has been developed, and to achieve this goal, the following guidelines had to be followed:

- provide APIs through which a user can interact with it;
- develop interconnected microservices, each of them in charge of different tasks;

- containerize each microservice.

The following microservices have been identified as pillars of the entire MLaaS4HEP service:

- a MLaaS4HEP server, which allows users to submit MLaaS4HEP training workflow requests and manage all the actions related to it;
- an authentication/authorization layer, which allows to authenticate the users and authorize their requests to the MLaaS4HEP server;
- an XRootD Proxy server, which allows to use X.509 proxies for the remote access of data.

The MLaaS4HEP server has been developed using the Flask [17] framework and it has been equipped with APIs that allow the user to submit and manage a MLaaS4HEP training workflow by making HTTP calls via curl. The authentication/authorization layer has been implemented using an OAuth2 Proxy server [18] and it has been configured in a way that a user of the CMS experiment can use the MLaaS4HEP server. In particular, he/she can register a client choosing *https://cms-auth.web.cern.ch/* as authorization server and use the oidc-agent tool [19] to get back an access token. Then is the proxy's task to validate the user-supplied token, authorizing him/her to exploit the MLaaS4HEP functionalities. The XRootD Proxy server has been developed using the compose-xrootd solution [20] that allows to create and renew X.509 proxies to access remote ROOT files located on WLCG sites.

All these services have been deployed as Docker [21] containers and can be connected with TFaaS (and an addittional OAuth2 Proxy server) in order to use the trained ML models for inference. A schematic representation of this solution can be found in Fig. 2.

4 The MLaaS4HEP service on INFN Cloud

INFN offers to users a comprehensive and integrated set of Cloud services through its dedicated INFN Cloud infrastructure. The INFN Cloud portfolio of services can be accessed via an easy-to-use web interface (the INFN Cloud Orchestrator Dashboard) but also via command-line interface. It is based on composable, scalable, open-source solutions and can be easily extended directly by the end users themselves. The INFN Cloud services are based on modular components and span the IaaS, PaaS, and SaaS models for both computing and data. All these services are described by TOSCA templates [22], which can refer internally to other components, such as Ansible [23] playbooks and HELM [24] charts. One of the services available in INFN Cloud is docker-compose which deploys a Virtual Machine (VM) with docker engine and docker-compose pre-installed and optionally runs a docker-compose file fetched from the URL specified by the user.

Thus, we took advantage of the docker-compose service, already available in INFN Cloud, to deploy a prototype of the MLaaS4HEP service. Once the related button is selected in the Orchestrator Dashboard (see Fig. 3), the user is redirected to a configuration page where he/she is asked to provide some information aimed at setting up:

- the resource used to host the application (e.g. the RAM size, the number of CPUs, and the ports to open), see Fig. 4;
- the URL of the docker-compose file to be used, see Fig. 5;
- variable keys and the corresponding values to be used by the application during the deployment process, see Fig. 5;
- the cloud provider where to schedule the service deployment, e.g Backbone-Bari, Backbone-CNAF, and Cloud-CNAF, see Fig. 6.

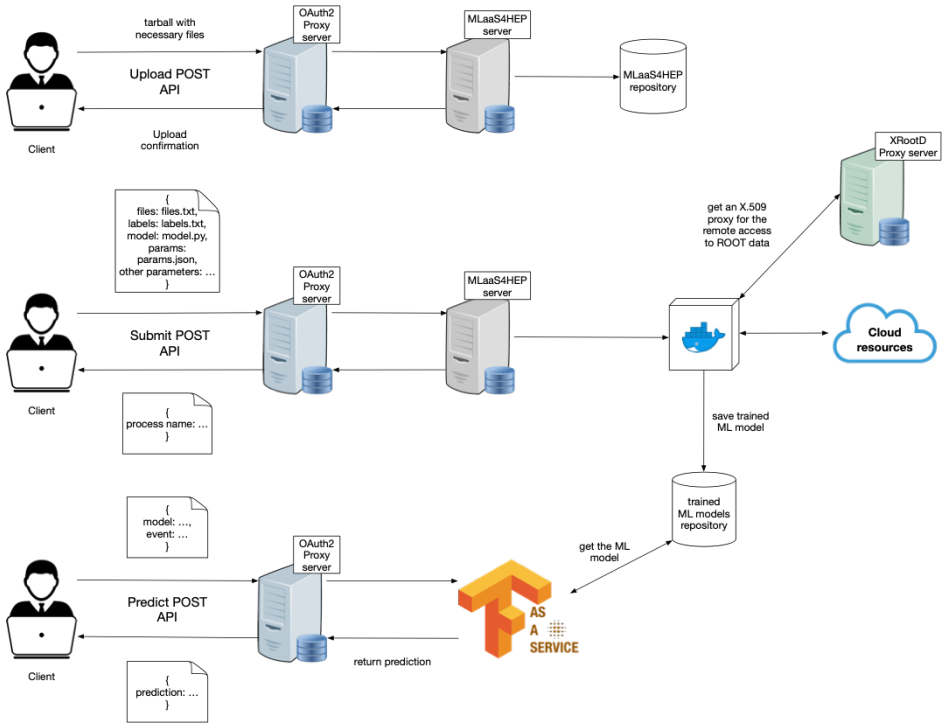

Figure 2. Solution connecting two OAuth2-Proxy servers, MLaaS4HEP server, XRootD Proxy server, and TFaaS. Firstly, a client uploads a tarball with the necessary files to the MLaaS4HEP server. Then he/she submits a MLaaS4HEP training workflow (which consists in running a Docker container in the server) where remote ROOT files can be read using valid X.509 proxies, and the ML model is trained and saved. This model is accessed by the TFaaS service that uses it to make inference. In front of the MLaaS4HEP and TFaaS servers, two OAuth2-Proxy servers are used to handle user authentication and request authorization.

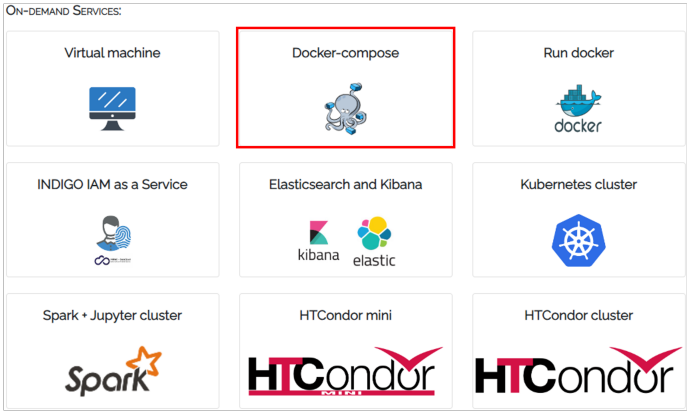

Figure 3. Overview of some services available in the INFN Cloud Orchestrator Dashboard.

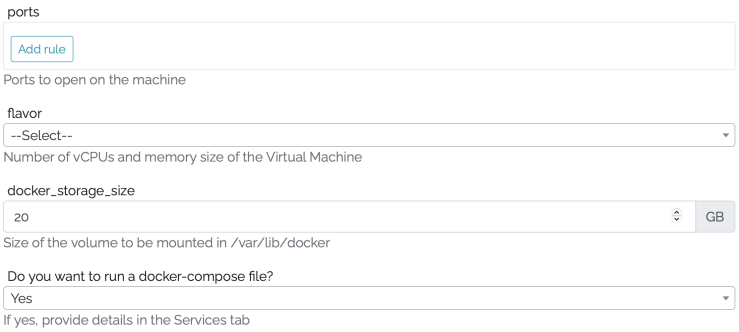

Figure 4. Parameters for the docker-compose service available in the INFN Cloud Orchestrator Dashboard, e.g. the RAM size, the number of CPUs, and the ports to open of the resource used to host the application to be deployed.

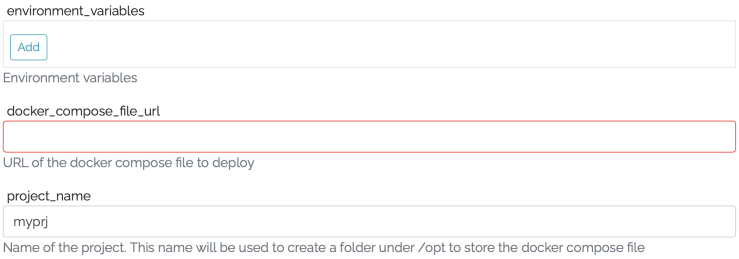

Figure 5. Parameters for the docker-compose service available in the INFN Cloud Orchestrator Dashboard, in particular the URL of the docker-compose file to be used, and the variable keys with the corresponding values to be used by the application during the deployment process.

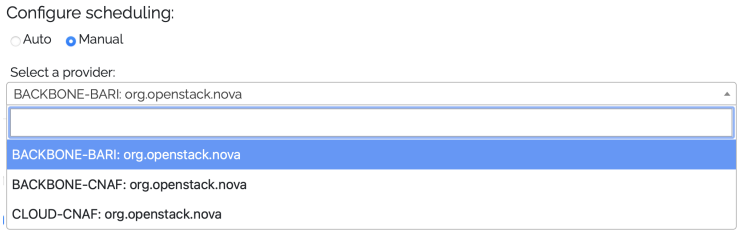

Figure 6. Parameters for the docker-compose service available in the INFN Cloud Orchestrator Dashboard, in particular the cloud provider where to schedule the service deployment.

This information is used to populate a TOSCA template, which is then submitted to the PaaS Orchestrator [6] as a service deployment request. As an outcome, the system will provide the IP address of the deployed VM. Since the MLaaS4HEP service requires an SSL certificate (for the two OAuth2-Proxies) and a Grid certificate (for the XRootD Proxy), we used the docker-compose service in the INFN Cloud Orchestrator Dashboard to deploy the VM with the docker engine and docker-compose pre-installed. Then, once logged into the

VM we copied our certificates to the proper locations, and subsequently the docker-compose file [25] we prepared has be used to deploy the entire service.

At this point, once we get an access token from the authorization server with the oidc-agent tool, we can contact the MLaaS4HEP or TFaaS servers using curl, e.g. in the following ways:

```
curl -L -k -H "Authorization: Bearer ${TOKEN_MLAAS}" -H "Content-Type: application/json"
-d @submit.json https://VM_ip:MLaaS4HEP_port/submit

curl -L -k -H "Authorization: Bearer ${TOKEN_TFAAS}" -X POST
-H "Content-type: application/json" -d @predict_bkg.json https://VM_ip:TFaaS_port/json
```

The former command is used to submit a MLaaS4HEP training workflow exploiting the information stored in the *submit.json* file to train the ML model which definition is provided by the user. The latter command allows to get the prediction on a given event (defined in the *predict_bkg.json* file) using the trained ML model. A demonstration showing the commands that can be used to interact with the MLaaS4HEP and TFaaS servers can be found in [26].

5 Conclusions

In this paper, we presented a working prototype of the MLaaS4HEP service, a cloud native solution that allows to perform ML pipelines using HEP data, deployed using the INFN Cloud Orchestrator Dashboard. The entire service is composed of several components which are containerized and deployed using the docker-compose service. With this solution a HEP user can submit ML pipelines, after being authenticated and authorized, using local or remote ROOT files simply using HTTP calls.

Currently, there are some working directions under investigation and development for the MLaaS4HEP project, e.g. involving parallelization of I/O and distributed ML training, making MLaaS4HEP usable also for other tasks (e.g. regression problems and image classification) as well as accepting other data formats as input, and providing a general inference service.

References

[1] *About ROOT*, https://root.cern/about/

[2] V. Kuznetsov, L. Giommi, D. Bonacorsi, Comput. Softw. Big Sci. **5**, 17 (2021), 2007.14781

[3] L. Giommi, V. Kuznetsov, D. Bonacorsi, D. Spiga, PoS **ISGC2021**, 019 (2021)

[4] L. Giommi, D. Spiga, V. Kuznetsov, D. Bonacorsi, M. Paladino, PoS **ISGC2022**, 012 (2022)

[5] L. Giommi, D. Spiga, V. Kuznetsov, D. Bonacorsi, PoS **ICHEP2022**, 968 (2022)

[6] D. Salomoni, I. Campos, L. Gaido, G. Donvito, M. Antonacci et al., *INDIGO-Datacloud: foundations and architectural description of a Platform as a Service oriented to scientific computing* (2016), https://arxiv.org/abs/1603.09536

[7] *INFN Cloud*, https://www.cloud.infn.it

[8] *MLaaS4HEP*, https://github.com/lgiommi/MLaaS4HEP

[9] *NumPy: the fundamental package for scientific computing with Python*, https://numpy.org

[10] *Uproot*, https://uproot.readthedocs.io/en/latest/index.html

[11] *XRootD*, https://xrootd.slac.stanford.edu

[12] *TFaaS*, https://github.com/vkuznet/TFaaS
[13] *Keras: Simple. Flexible. Powerful.*, https://keras.io
[14] *PyTorch*, https://pytorch.org
[15] *scikit-learn: Machine Learning in Python*, https://scikit-learn.org/stable/
[16] *XGBoost Documentation*, https://xgboost.readthedocs.io/en/stable/
[17] *Flask*, https://flask.palletsprojects.com/en/3.0.x/
[18] *OAuth2-Proxy server*, https://oauth2-proxy.github.io/oauth2-proxy/
[19] *oidc-agent*, https://indigo-dc.gitbook.io/oidc-agent/
[20] *compose-xrootd*, https://github.com/comp-dev-cms-ita/compose-xrootd
[21] *Docker: Accelerated Container Application Development*, https://www.docker.com
[22] *INFN Cloud customized Tosca templates*, https://baltig.infn.it/infn-cloud/tosca-templates
[23] *Ansible is Simple IT Automation*, https://www.ansible.com
[24] *HELM: The package manager for Kubernetes*, https://helm.sh
[25] *Docker-compose file to deploy the MLaaS4HEP service*, https://github.com/lgiommi/MLaaS4HEP_server/blob/master/docker-compose.yaml
[26] *MLaaS4HEP service demo*, https://www.youtube.com/watch?v=_JHg4oTeVbc