EPJ Web of Conferences **251**, 02067 (2021) https://doi.org/10.1051/epjconf/202125102067
CHEP 2021

Training and Serving ML workloads with Kubeflow at CERN

Dejan Golubovic[1,*] and *Ricardo* Rocha[1,**]

[1]CERN, 1 Esplanade des Particules, Geneva, Switzerland

Abstract. Machine Learning (ML) has been growing in popularity in multiple areas and groups at CERN, covering fast simulation, tracking, anomaly detection, among many others. We describe a new service available at CERN, based on Kubeflow and managing the full ML lifecycle: data preparation and interactive analysis, large scale distributed model training and model serving. We cover specific features available for hyper-parameter tuning and model metadata management, as well as infrastructure details to integrate accelerators and external resources. We also present results and a cost evaluation from scaling out a popular ML use case using public cloud resources, achieving close to linear scaling when using a large number of GPUs.

1 Introduction

In recent years there have been multiple efforts to apply machine learning (ML) techniques to solve different tasks and problems in High Energy Physics (HEP), covering fast alternatives to Monte Carlo based simulation [1], anomaly detection algorithms applied to the search of rare new physics [2], and fast inference models in 40 MHz scouting [3]. In other industries and research areas the momentum has been even larger with ML taking an important role in a variety of processes: from the more well known image pattern recognition or suggestion and profiling algorithms, to the more recent uses in the area of self driving vehicles or the controversial use of ML for deep fakes. Overall ML is now a big part of the daily life of most people, even if in many cases not easily recognizable.

The level of investment in this area has led to the spread of libraries and platforms available, popular ones including TensorFlow, PyTorch or scikit-learn. Even though they have well-established communities backing them, the overall process where they are put in use is a lot less clear. Even within the same institution it is not uncommon to have different groups following very different processes to go from the original raw data to the point of being able to serve a trained model in production. Improvements in this area can have a dramatic impact in the productivity of data scientists, and as an outcome in the quality of the results achieved.

We present a new service recently made available to the CERN community that tries to help with improving the overall process of machine learning. It is based on Kubeflow, a machine learning toolkit offering components to handle each of the required steps of data loading and pre-processing; efficient distributed model training, storage and versioning; and finally model serving. We describe an infrastructure based on Kubernetes and how that helps to ensure availability, scalability and the capability to burst out and explore external resources.

*e-mail: dejan.golubovic@cern.ch
**e-mail: ricardo.rocha@cern.ch

We present results from the first use cases deployed in our system, and how we managed to improve efficiency significantly with little or no changes to the original code base.

The service is hosted at https://ml.cern.ch. At this page, users can navigate the service via browser UI. The central dashboard provides links to Jupyter notebooks, pipelines, Katib (hyper parameter search tool) and other main features.

Documentation for using the service is provided at https://ml.docs.cern.ch.

Repository with a set of examples, covering notebooks, pipelines, distributed training, model storage, model serving, is provided at https://gitlab.cern.ch/ai-ml/examples.

2 Service Objectives

Following on the tradition of data science in our community, multiple groups at CERN have been working independently in developing solutions for machine learning. In most cases these are still rather small efforts, with a limited amount of resources available and/or handling only a few use cases.

While a central service will likely not serve all the use cases, it has the advantage of getting together experts in the ML toolkits and algorithms with those more knowledgeable in handling and scaling the infrastructure. It also serves the purpose of aggregating similar workloads in one place, potentially making a larger number of resources available to each group while improving the overall resource usage. This is especially important when the requirements include a scarce resource such as GPUs and where workloads are often spiky.

Other key features taken into account when designing the service included are:

- Ease of use, meaning that end users should be able to continue using their favorite tools and libraries. In most cases this means the ability to perform interactive analysis using Jupyter Notebooks, supporting popular libraries like TensorFlow, PyTorch or MXNET, and efficient integration with CERN identity and storage systems.
- Availability, important when offering the ability to reliably train the models, but even more important when considering that the service is also doing the serving of the models. If it is to be successful, the service must be capable of being used by critical areas for accelerator and detector operations.
- Scalability, as the problems being tackled by our community are hard and often require a large amount of resources. The service should be able to scale individual workloads as well as serve a large amount of concurrent users.
- Sustainability, meaning the tools selected must be well established and have strong communities backing them. As quickly described in the introduction, given the popularity of machine learning in a wide variety of industries and institutions it would make little sense to start a new in-house development effort that would require significant larger man power for development, operations and support.

3 Infrastructure and Kubeflow

Based on the objectives described earlier, we chose Kubeflow as the platform for our service. Kubeflow is an open source machine learning platform built on top of Kubernetes [4], focusing on simplicity of usage, scalability and the utilization of growing and community-supported technologies.

It can be installed on a local laptop, on on-premises cluster or on resources in a public cloud. Each component is containerized and managed as a micro-service, with all the expected declarative definitions that allow them to be efficiently deployed, handle failure

and easily scale out when required. Clusters can have thousands of nodes, split over multiple failover domains and made of heterogeneous resources. Workload isolation is guaranteed thanks to the usage of Kubernetes namespaces [5], which can serve individual users or groups of users. Each namespace is assigned a fixed resource quota, with user identity being enforced using OIDC/OAuth2 integration with the CERN application portal, and access control via Kubernetes RBAC policies and rules.

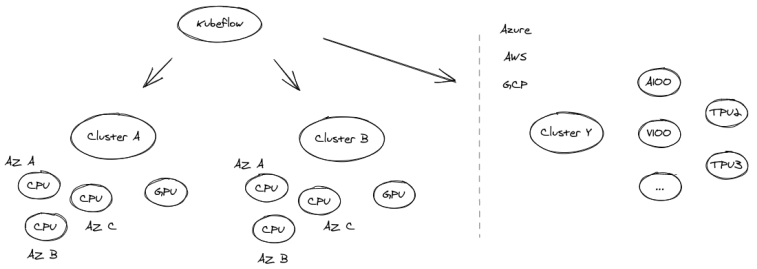

Figure 1: Kubeflow deployment splitting workloads between multiple clusters for scalability and reduced blast radius, and bursting to public clouds to access large numbers of GPUs and acceleartors such as TPUs

Figure 1 presents an overview of the CERN deployment. CERN has been running in production a large private cloud based on OpenStack, originally built to offer virtual machines and the ability to attach virtual storage appliances. It has grown significantly in the last few years and expanded to include the ability to provision baremetal resources, advanced networking features like security groups or Load Balancing as a Service (LBaaS) and more relevant to this work a managed Kubernetes service, capable of handling accelerators such as GPUs.

A few characteristics depicted in Figure 1 should be highlighted:

- The service has instances running in multiple clusters, with the goal of reducing blast radius, rolling out infrastructure changes gradually and minimizing the impact of potential failures during cluster upgrades. To ease the management of the deployment, we rely on GitOps and particularly the ArgoCD tool which is capable of managing multiple cluster deployments easily.
- Each cluster is highly available with instances in multiple availability zones. Nodes are grouped into pools according to the types of resources they offer, each being able to auto scale according to the current load on the service.
- The service is able to scale out to external cloud resources, which is important given the limited availability of GPUs and the ability to access accelerators such as TPUs or IPUs which would otherwise not be accessible.

4 Features

We cover below some of the main features offered by this new service that make it suitable to cover a large variety of machine learning use cases.

4.1 Python and Jupyter Notebooks

A common first step in a data analysis or machine learning workflow is to perform some interactive analysis over a subset of the interesting data. Python and Jupyter notebooks are

already a tool most end users are familiar with, thanks for their simplicity of use, extensive selection of libraries and frameworks, and flexibility to run on different platforms.

Kubeflow offers a central place where users can maintain multiple notebook servers. The default user interface is Jupyterlab [6], a web-based interactive development environment for Jupyter notebooks. Each server is a container and launched from a base image, with a set of curated profiles available and the ability for end users to supply their own. Users can pass the resources they need for each server, including CPU, memory and GPUs.

4.2 Pipelines

An interactive session on a notebook is a great way to quickly prototype data analysis code, but its sequential nature poses limitations when moving to training at scale. These are often managed as workflows (pipelines) with dependencies between multiple steps: a common workflow would be to perform some data retrieval followed by pre-processing, then executing a few variations of model training in parallel, and lastly aggregate the multiple results.

Kubeflow provides several key features in this area: a web interface for managing and tracking pipeline runs, an SDK for defining pipelines and components using Python, an engine for scheduling pipelines and a very handy feature to convert what is initially a sequential notebook into a pipeline using a JupyterLab extension called KALE [7]. Whatever the choice for defining and describing a pipeline, the end result is always a Kubernetes resource in YAML [8] which defines all steps and their relationships. Steps are then run as independent and isolated containers in the cluster, with the possibility of having shared storage between them. Logs and results are exposed using standard Kubernetes APIs and exposed to the users live via the web interface. Figure 2 shows pipeline creation using KALE. To create a pipeline from a running notebook, the only additional step is to annotate notebook cells and define connections between them. Clicking the *Compile and Run* button converts notebook cells to pipeline components. The UI monitoring a pipeline run is shown on the right side of Figure 2. Figure 3 shows pipelines from the infrastructural perspective. A simple pipeline example is shown on the left side. On the right side it is shown how every pipeline pipeline component is run as a Docker container. These containers are controlled by Kubernetes, and run as pods.

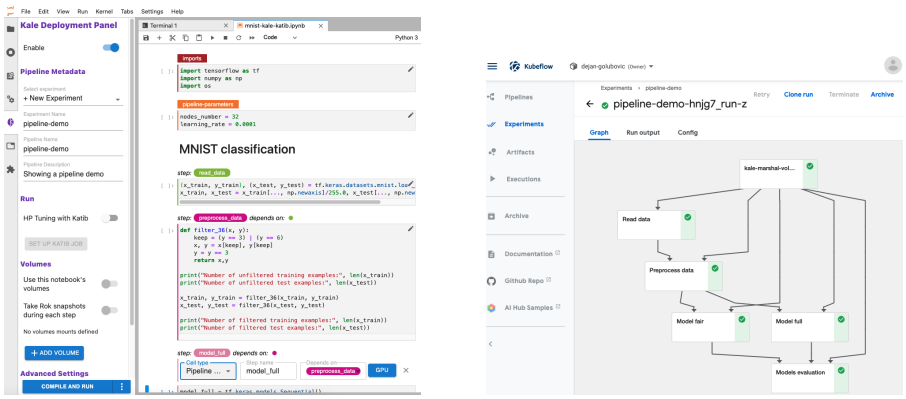

Figure 2: (left) A Jupyterlab environment. Notebook cells are annotated and connected to represent the pipeline steps. GPU can be allocated to a specific cell. Left panel shows the pipeline configuration, pipeline name, experiment name and pipeline description. *Compile and Run* button compiles the notebook and run a pipeline. (right) A Kubeflow Pipelines UI, tracking pipeline progress in real time. Clicking on pipeline components shows additional information, such as logs, visualizations and artifacts.

Some extra functionalities include the ability to express special resource requirements per pipeline step, particular important for GPUs and other accelerators making sure they are only claimed for the time they are required; recurrent (scheduled) pipelines for automated testing or report generation; grouping of pipeline runs in *experiments*, which allows tracking and comparing similar runs with varying input parameters or small changes in the code.

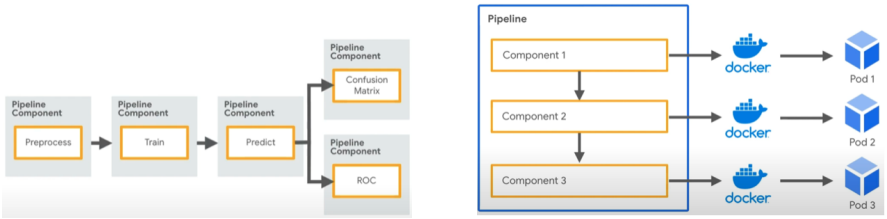

Figure 3: (left) An example of a machine learning pipeline. (right) Pipeline infrastructure in Kubeflow, packaging components as images, then running as Kubernetes pods.

4.3 Automated Hyper Parameter Optimization

Hyper parameter optimization is the process of finding the set of non trainable parameters that yield to best performance of a model. Hyper parameters can be learning rate, number of layers, regularization parameter, dropout rate, and many others. Kubeflow provides a tool called Katib which focus on automated hyper parameter search. [9]

Katib is framework agnostic, supporting any arbitrary machine learning framework. To submit a Katib experiment, it is required to provide a script for model training with hyper parameters as command line arguments, package the script as a Docker image, and provide a YAML file with the definition of hyper parameters. Katib is then scheduling the search job and running multiple trials, with each trial corresponding to one combination of hyperparameters. The submission can be done manually by implementing each step, or automatically by annotating the notebook cells using the KALE extension mentioned earlier. Further customizations include selecting the search algorithm (grid, random [10], Bayesian [11]) and its parameters, level of parallelization, and the optimization goal.

Figure 4 shows Katib usage from KALE. The left side shows parameters selection. Parameters, optimization algorithm and its settings can be specified. This is translated to a set of trials, each trail being a pipeline with the specified parameters. The results of four trials are shown on the right side. Katib UI provides visual and numerical representation of results.

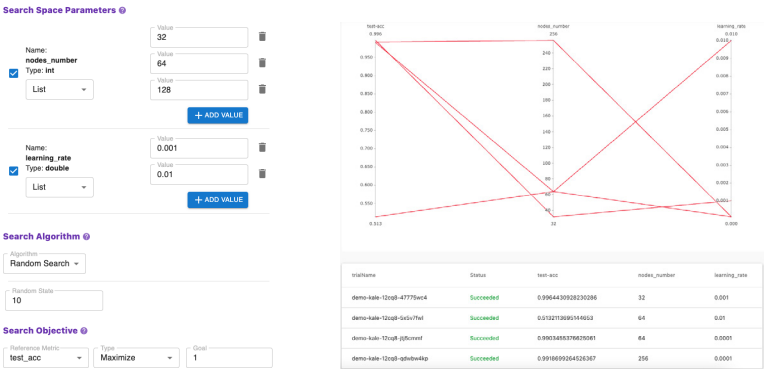

Figure 4: (left) Configuring a hyper parameter optimization job. Defining and selecting parameters and their values, selecting search algorithm, defining search objective. (right) After the completed search job, results are shown in the Katib UI, visually and numerically.

4.4 Distributed Training

Computationally intensive machine learning models can benefit from distributed strategies in terms of the time required to complete the training job. Machine learning frameworks such as TensorFlow [12] or PyTorch [13] offer features to run a training job across multiple GPUs. Kubeflow provides a connection between the frameworks' distributing features and the Kubernetes containerized environment via dedicated operators, with existing support for TensorFlow, PyTorch, MXNET, MPI, among others.

These operators take care of all the required orchestration and setup of the multiple jobs in the training, required communication and credentials, as well as data exchange and aggregation after completion. This is a huge step in automating what is otherwise a rather complex and error prone operation. Users are only required to specify the number of chiefs and workers they require, and the type of resources they should have access to.

In a later section we present results from a real use case where we show that the system is extremely efficient when executing this sort of task, scaling almost linearly when handling requests with a high number of parallel jobs.

4.5 Storage Integration

One of the key aspects of the service is access to storage. Requirements include access to input data, software, user home directories, as well as a reliable way to store the outputs. The service as deployed at CERN provides access to multiple storage backends:

- EOS [14] as this is the main source for input datasets, and also the location of the user's home directories. Access to EOS from the CERN Kubeflow instance is done using OAuth2 tokens retrieved from the CERN IdP, which are kept and refreshed as required for long lasting jobs.
- S3, as the backend for the image registry which is the base for all notebooks and jobs running on the system. We recommend users to keep these images up to date to ensure reproducibility and ease sharing of code and documentation. S3 is also the backend for the Kubeflow metadata component, which allows storing and versioning of machine learning artifacts such as models, datasets or pipeline runs.
- CVMFS as the host of a significant amount of end user software.

4.6 Model Serving

Once a model is trained it needs to be made available to other components - a process usually called serving. The goal is to provide a way to query the model, getting outputs for a certain set of inputs. Kubeflow provides a component that exposes a REST API for model serving, with a dedicated endpoint for querying each model. These are serverless endpoints backed by a tool called Knative [15], and are able to auto scale according to the number of requests being made at each point in time. Figure 5 shows the workflow of model serving. A load balancer is receiving requests and redirects traffic to one of the pods that have access to the model, for obtaining predictions.

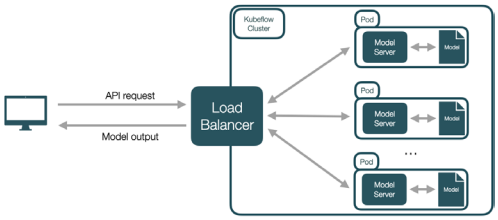

Figure 5: Model serving example. Users are querying a load balancer via the network. The API request is pointing to the model's API endpoint, sending the data input. An example can be a curl command: curl -v "MODEL_IP:PORT/v1/models/model:predict" -d @./input.json. The API request is received by a load balancer, which directs the traffic to one of the model servers. Servers pass the data input for inference through the model and return the model output.

5 Sample Use Case: Training 3DGANs

We present one of the first use cases to be run in the new service: fast calorimeter simulation with 3D GANs, a popular use case for deep learning in HEP [1]. The code is based on TensorFlow 2 and the *mirrored strategy* of the *distributed.strategy* API, which allowed to run it on GPUs and other accelerators like TPUs. Below we focus on the training part, specifically on taking an existing code base and attempting to scale the training with minimal changes.

The first run was done at CERN and allowed for the validation of the setup and code with a low number of GPUs in a multi node environment, due to limited availability of resources. We then moved to trying the same setup in the Google Cloud Platform (GCP) where we had access to a larger number of GPUs as well as TPUs.

5.1 Public Cloud

ML workloads, which are often spiky when doing training, can greatly benefit from accessing a large amount of resources that are often scarce in on-premises deployments (GPUs and FPGAs), as well as access to specialized accelerators otherwise not available such as TPUs (GCP) or IPUs (Azure).

As part of the deployment of the new service we experimented with the integration of external resources from GCP, attempting to scale out the use case described above. The results presented below were achieved relying on a Google Kubernetes Engine (GKE) cluster running Kubernetes 1.17.15, Nvidia driver version 450.51.06 and Kubeflow 1.1. Nodes are of type *n1-standard-32* with 8 V100 GPUs each.

We would like to highlight a few best practices when using the public cloud that made a big difference in both performance and cost efficiency, without going too much in detail as this is not the main focus of this report.

- Preemptibles are low Service Level Agreement (SLA) instances which can be reclaimed at any moment with short notice to the user and have a maximum lifetime of 24 hours, but are several times cheaper and can make a big difference in terms of final cost. Model training is a suitable use case for this type of resources - or the similar spot instance model in AWS or Azure.
- Splitting the cluster into multiple node pools allows appropriate matching between workloads and the best suited and most cost effective type of resource. For this use case single

nodes with 8 GPUs each are the best option but these would clearly be too large for simpler workloads and would lead to waste, so we also have pools with smaller nodes.

- Enabling cluster auto scaling is essential to ensure appropriate resource usage. Relying on Kubeflow and Kubernetes greatly simplifies this task, but it is still important to make sure all workloads are properly labeled to match the node group that fits them best.

An often relevant point when relying on the public cloud concerns egress costs, but given the generated data (the models) is rather small this was not something we had to deal with. For other types of workloads we expect to have in place the required agreements for cost effective egress usage.

5.2 Results and Cost Analysis

Prior to the runs that lead to these results, we evaluated the best batch size settling at 96, as well as the best number of workers for the number of GPUs - which turned out to be 1 worker per node, we believe due to the reduction in competition for shared resources in a single node.

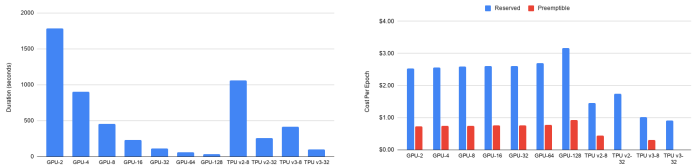

Figure 6: (left) The impact of increasing number of total GPUs assigned to workers on time to process one epoch of training. (right) Cost of running one epoch on the Google Cloud Platform. The cost per epoch remains similar for all GPU combinations, though the training time improves almost linearly.

The performance improvement is close to linear with a slight drop when moving to 128 GPUs, which are positive results regarding both TensorFlow's capabilities to parallelize and distribute the training, as well as Kubeflow's capability to deploy the workload using a TFJob [16], particularly considering the additional ease of use and automation offered to end users. TPUs are also very efficient, significant gains are achieved by increasing the number of cores.

Another important point when using public cloud resources is to understand the costs involved, and specifically to choose the best type and adequate amount of resources for each particular workload - with the goal of achieving a good balance between investment and speed and quality of results. For this particular test the most relevant cost comes from the GPUs, with virtual machines accounting for less than 5% of the total. Our results also include a calculation of the training time of one epoch against the hourly GPU costs in region *europe-west4* and include both reserved and preemptible options. It should be noted that when using reserved instances it is possible to apply for sustained or committed-use discounts which reduce the difference to preemptible instances costs. Two important points worth noting (figure 6, right panel): the cost per epoch remains similar when increasing the number of GPUs, while the training time is reduced up to 52 times for 128 GPUs (almost linear up to 64 GPUs); the best results are achieved using preemptible TPU v3-8, which are 2.4 times cheaper than their GPU equivalent considering the training time. This is an interesting result that positions TPUs as a cost-effective option combined with their potential to reduce the overall training time. Preemptible TPU are only available up to 8 cores, with the reserved instance costs being less attractive. As an example, the overall training cost with 128 GPUs is the same as TPU v3-32, but it takes half the time.

6 Conclusions and Future Plans

Machine learning usage has been growing significantly in the HEP community, and is one of the most promising areas where progress can help tackle the upcoming challenges for future experiments.

We present a new service available at CERN focusing on providing a scalable and sustainable solution covering the ML requirements of our community, and offering the features required to minimize the entry barrier for end users by allowing them to continue using the libraries and platforms they are already familiar with.

Our service covers the full ML process from loading and pre-processing the data, quick interactive analysis, large scale distributed model training and finally model serving. It provides access to both on-premises and external cloud resources, including both CPUs, GPUs and dedicated accelerators such as TPUs and IPUs. We presented a real use case where linear scaling was achieved with a large number of GPUs, as well as the equivalent results when using TPUs. We also presented a cost analysis of public cloud usage, showing that it can be a very good solution for spiky workloads that can scale out to large amounts of resources.

As the service is made generally available we expect new challenges ahead, but we have already identified some areas where future work is required: integrating additional data sources, in particular the systems hosting the log data for CERN systems - a requirement from multiple anomaly detection efforts; improving the multi cluster and multi cloud experience, which is currently still rather complex on the infrastructure side - we expect to do this together with the Kubernetes community, where there's a strong push from people with similar interests; integrating and evaluating new types of resources, particular FPGAs which promise a significant improvement in model service.

References

[1] Vallecorsa, Sofia, Carminati, Federico, Khattak, Gulrukh, EPJ Web Conf. **214**, 02010 (2019)

[2] A.A. Pol, V. Berger, G. Cerminara, C. Germain, M. Pierini, Tech. Rep. arXiv:2010.05531 (2020), presented at ICMLA 2019, https://cds.cern.ch/record/2742923

[3] D. Golubovic, T.O. James, E. Meschi, E. Puljak, D. Rabady, A.Z. Rashid, H. Sakulin, E. Vourliotis, P. Zejdl, *40 MHz Scouting with Deep Learning in CMS*, in *Connecting the Dots 2020* (2020)

[4] *Kubeflow*, https://www.kubeflow.org/ (2021), accessed: 2021-02-28

[5] *Kubernetes namespaces*, https://kubernetes.io/docs/concepts/overview/working-with-objects/namespaces/ (2021), accessed: 2021-02-28

[6] *Project jupyter*, https://jupyter.org (2021), accessed: 2021-02-28

[7] *Kale*, https://github.com/kubeflow-kale/kale (2021), accessed: 2021-02-28

[8] *Yaml*, https://www.yaml.org/ (2021), accessed: 2021-02-28

[9] J. George, C. Gao, R. Liu, H.G. Liu, Y. Tang, R. Pydipaty, A.K. Saha, *A scalable and cloud-native hyperparameter tuning system* (2020), 2006.02085

[10] P. Liashchynskyi, P. Liashchynskyi, *Grid search, random search, genetic algorithm: A big comparison for nas* (2019), 1912.06059

[11] J. Snoek, H. Larochelle, R.P. Adams, *Practical Bayesian Optimization of Machine Learning Algorithms*, in *Advances in Neural Information Processing Systems*, edited by F. Pereira, C.J.C. Burges, L. Bottou, K.Q. Weinberger (Curran Associates, Inc., 2012), Vol. 25, https://proceedings.neurips.cc/paper/2012/file/05311655a15b75fab86956663e1819cd-Paper.pdf

[12] *Distributed tensorflow*, https://www.tensorflow.org/guide/distributed_training (2021), accessed: 2021-02-28

[13] *Distributed pytorch*, https://pytorch.org/tutorials/intermediate/ddp_tutorial.html (2021), accessed: 2021-02-28

[14] *Eos open storage*, https://eos-web.web.cern.ch/eos-web/ (2021), accessed: 2021-02-28

[15] *Knative*, https://knative.dev/ (2021), accessed: 2021-02-28

[16] *Kubeflow tfjob*, https://www.kubeflow.org/docs/components/training/tftraining (2021), accessed: 2021-02-28