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

Luca Giommi\textsuperscript{1,2,3} (luca.giommi@cnaf.infn.it), Daniele Spiga\textsuperscript{4}, Valentin Kuznetsov\textsuperscript{5}, Daniele Bonacorsi\textsuperscript{2,3}

\textsuperscript{1} INFN CNAF, Italy  
\textsuperscript{2} INFN Bologna, Italy  
\textsuperscript{3} University of Bologna, Italy  
\textsuperscript{4} INFN Perugia, Italy  
\textsuperscript{5} Cornell University, USA
What is Machine Learning as a Service

Machine Learning as a Service (MLaaS) is used as an umbrella definition of various cloud-based platforms that provide a web service to users interested in ML tasks.

- Leading cloud providers offer MLaaS solutions with different interfaces and APIs, designed to cover standard use cases, e.g. classification, regression, clustering, anomaly detection, performed in different sectors like natural language processing and computer vision.

- These platforms simplify and make ML accessible to even non-experts, ensuring affordability and scalability as these services inherit the strengths of the underlying cloud infrastructure. Moreover, the MLaaS solutions are well integrated with the rest of the provider’s portfolio of services which thus offers a complete solution.

<table>
<thead>
<tr>
<th>CLOUD MACHINE LEARNING SERVICES COMPARISON</th>
<th>Amazon ML and SageMaker</th>
<th>Microsoft AI Platform</th>
<th>Google AI Platform (Unified)</th>
<th>IBM Watson Machine Learning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Classification</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Regression</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Clustering</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>X</td>
</tr>
<tr>
<td>Anomaly detection</td>
<td>✓</td>
<td>✓</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Recommendation</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>X</td>
</tr>
<tr>
<td>Ranking</td>
<td>✓</td>
<td>✓</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Data Labeling</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>MLOps pipeline support</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>X</td>
</tr>
<tr>
<td>Built-in algorithms</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>
Why Machine Learning as a Service in HEP?

- ML successfully applied in many areas of HEP and it will play a significant role during the upcoming HL-LHC program at CERN

- Developing a ML project and implementing it for production use requires specific skills and is a highly time-consuming task.
  - It would be helpful to provide HEP physicists who are not experts in ML with a service that allows them to exploit the potentiality of ML easily

- MLaaS solutions offered by major service providers have many services and cover different use cases but are not directly usable in HEP.
  - ROOT data format cannot be directly used
  - Flattening of data from the dynamic size event-based tree format to the fixed-size data representation does not exist
  - Pre-processing operations may be more complex than the ones offered

- There are various R&D activities underway within HEP aimed at providing HEP analysts with tools or services to accomplish ML tasks.
  - Solutions designed only for optimization of the inference phase (e.g. hls4ml, SonicCMS)
  - Custom solutions adopted in specific CMS analyses cannot easily generalized and do not represent “as a Service” solutions
  - And others…
The MLaaS4HEP solution aims to:
- provide transparent access to HEP datasets stored in ROOT files
- use heterogeneous resources in HEP for training and inference
- use different ML frameworks of interest in HEP
- serve pre-trained HEP models and access it easily

MLaaS4HEP framework

Multi-language architecture: Python and Go

- **Data Streaming Layer**
  - developed using the Uproot library
  - allows to read ROOT data from local and remote data storage
  - use a Generator to read data in chunks

- **Data Training Layer**
  - process input data
  - provide a proper normalisation of each attribute
  - use data to train ML model chosen by user

- **Data Inference Layer**
  - implemented as Tensorflow as a Service (TFaaS)
  - provides access to pre-trained HEP models for inference purpose
Specs computing phase

1. Read all the ROOT files
2. Compute specs file
3. Load specs information

specs.json

{max:
  {key1: max_1,
   key_2: max_2,...},
  min:
  {key_1: min_1, key_2: min_2,...}, ...}

If chunk \( c_i \) is empty or fully processed, read \( N \) chunk events from the file \( f_i \).

\( \text{Did you go through all the files?} \)

\( \text{YES} \quad \text{NO} \)

\( i = i + 1 \)

\( i = 0 \)

Train the model for \( N \) epochs using batches of data with size \( N \).

Are all the files completely read?

NO

YES

 specs.json

compute specs file

load specs information

Read all the ROOT files

train the ML model
convert into numpy arrays, fix Jagged Arrays' dimension and normalise the values

Take $N_{\text{chunk}} \times \eta / N_{\text{tot}}$ events from the chunk $c_i$

Did you go through all the files?

Are all the files completely read?

If chunk $c_i$ is empty or fully processed, read $N_{\text{chunk}}$ events from the file $f_i$

Did you go through all the files?

Train the model for $N_{\text{epochs}}$ using batches of data with size $N_{\text{batch}}$

pre-process the events

train the ML model

read the events
Run a MLaaS4HEP workflow

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

**MLaaS4HEP training workflow**

**Input ROOT files**

**Labels related to ROOT files content**

**Definition of the ML model**

**MLaaS4HEP parameters**

**Definition of preprocessing operations**

---

**Keras model (model.py)**

```
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(),
                     metrics=AUC(name='auc'))
```

---

**MLaaS parameters (params.json)**

```
{
    "nevts": 3000,
    "shuffle": true,
    "chunk_size": 1000,
    "epochs": 3,
    "batch_size": 100,
    "identifier": "",
    "branch": "boostedAk8/events",
    "selected_branches": "",
    "exclude_branches": "",
    "hist": "pdfs",
    "redirector": "root://xrootd.ba.infn.it",
    "verbose": 1
}
```

---

**Input ROOT files (files.txt)**

```
PATH/flatTree_ttHJetTobb_M125_13TeV_amcatnloFXFX_madspin_pythia8.root
PATH/flatTree_TT_TuneCUETP8M2T4_13TeV-powheg-pythia8.root
```

---

**Labels of ROOT files (labels.txt)**

```
1
0
```
The MLaaS4HEP framework and its developments

1. It is developed to accept flat ROOT ntuples (e.g. NANOAOD) as input for HEP classification problems

2. It is ML framework and model agnostic. Currently, it has been tested using:
   - MLP written in Keras and PyTorch
   - MLP, Gradient Boosting, AdaBoost, Random Forest, Decision Tree, kNN, SVM, and Logistic Regression written in Scikit-learn
   - Gradient Boosting written in XGBoost

3. It is experiment agnostic
   - It has been validated, and its performance tested, choosing a signal vs background discrimination problem in a $t\bar{t}H$ analysis of CMS
   - It has been used to tackle the Higgs Boson ML challenge (ATLAS open data)

Developments made

1. It has been updated to support Uproot4, enabling pre-processing operations defined by the user

2. An additional training procedure of the ML models has been introduced

3. A SaaS solution has been provided using Dynamic On Demand Analysis Service (DODAS)
SaaS solution for MLaaS4HEP

- A SaaS solution for a sharable jupyter notebook has been provided
- Token-based access to the jupyterhub, with the support for a customizable environment

**Server Options**

Select your desired image: felixfelipc/mlaas_cloud:mlaas_jupyterhub

Select your desired memory size: 4GB

GPU: NotAvailable

Start

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

```bash
# ./shared/setup_local
(base) # cd /workarea/shared/folder_test
(base) # ../../../workarea/MLaaS4HEP/src/python/MLaaS4HEP/workflow.py --files=files_test.txt --labels=labels_test.txt --model=keras_model.py --params=params_test.json
# model parameters: {"nevts": -1, "shuffle": true, "chunk_size": 10000, "epochs": 5, "batch_size": 100, "identifier": ["runNo", "evtNo", "lumi"], "branch": "events", "selected_branches": ",", "exclude_branches": ",", "hist": "pdfs", "redirector": "root://gridftp-storm-t3.cr.cnaf.infn.it:1095", "verbose": 1}
Reading ttH_signal.root
# 10000 entries, 29 branches, 1.10626220703125 MB, 0.034181833267211914 sec, 32.364039645948566 MB/sec, 292.5530623775014 kHz
# 10000 entries, 29 branches, 1.1062220703125 MB, 0.022344589233398438 sec, 49.50917626973965 MB/sec, 447.53563807084936 kHz
```
Create a cloud native solution for MLaaS4HEP

The goal is to create a cloud service that could use cloud resources and could be added into the INFN Cloud portfolio of services

- MLaaS4HEP is not yet a service and should be developed as a cloud native application. The needed steps are:
  - 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 the pillars of the entire MLaaS4HEP service:
  - a MLaaS4HEP server, which allows to submit MLaaS4HEP 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
Integrated services

- **MLaaS4HEP server**
  - Written using the (Python-based) Flask framework

- **OAuth2 Proxy server**
  - Register the client with the authorization server: https://cms-auth.web.cern.ch/
  - Use a proper configuration file for the proxy
  - Obtain a token for the registered client using oidc-agent

- **XRootD Proxy server**
  - It creates an X.509 proxy and renews it when it is expired

- **TFaaS**

A working prototype of the service is running on a VM of INFN Cloud. Once the user obtains an access token from the authorization server, he/she can contact the MLaaS4HEP server or TFaaS 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://90.147.174.27:4433/submit
```

```
```
MLaaS4HEP service in INFN Cloud

Docker-compose

Description: Deploy a virtual machine with docker engine and docker-compose pre-installed. Optionally run a docker-compose file fetched from the specified URL.

Deployment description
MLaaS4HEP service

Environment variables

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>docker_compose_file_url</td>
<td><a href="https://raw.githubusercontent.com/giandom/MLaaS4HEP_server/master/docker-compose.yaml">https://raw.githubusercontent.com/giandom/MLaaS4HEP_server/master/docker-compose.yaml</a></td>
</tr>
</tbody>
</table>

URL of the docker-compose file to deploy

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>project_name</td>
<td>myProject</td>
</tr>
</tbody>
</table>

Name of the project. This name will be used to create a folder under /opt to store the docker-compose file.
Conclusions

- The MLaaS4HEP framework allows to perform ML pipelines (read data, pre-process data, train ML models) in HEP while TFaaS can be used for the inference phase
- A working prototype of the MLaaS4HEP service has been deployed on a VM of INFN Cloud
- A `docker compose` file has been developed to deploy the various microservices
- The MLaaS4HEP service deployment can be fully automated using the docker compose deployment of INFN Cloud
- Code available in GitHub ([MLaaS4HEP framework](https://github.com/MLaaS4HEPframework), [MLaaS4HEP service](https://github.com/MLaaS4HEPservice)), MLaaS4HEP service [demo](https://github.com/MLaaS4HEPservice) available

Outlook

- Involve parallelization of I/O, distributed ML training, etc.
- Make MLaaS4HEP usable also for other tasks, e.g. regression problems, image classifications, as well as accept other data formats as input
- Provide a general inference service
Thanks for the attention
Questions?
Collectively, LHC experiments operations produce ~200 PB of data each year that must be stored, processed, and analyzed (the entire amount is ~1.5 EB). To allow physicists to have access to computing power and storage needed to conduct research activities, CERN exploits the Worldwide LHC Computing Grid (WLCG).

Challenges towards High Luminosity LHC (HL–LHC)
- Fitting within the limited budget for computing
- Managing Exabyte scale data
- Heterogeneous computing and portability

The **ROOT** framework provides the data format commonly used to store HEP data, as well as tools to access and analyze such data.
Jagged/Awkward Arrays

- Each event is a composition of flat and Jagged/Awkward branches.
  - Jagged Array is a compact representation of variable size event data produced in HEP experiments
  - Such a data representation is not directly suitable for ML

- To feed these data into ML frameworks, the Jagged Arrays are flattened into fixed-size arrays with padding values through a two-step procedure:
  - compute the dimensionality of every Jagged Array attribute
  - update the dimension of the Jagged branches using padding values

- The mask array with padding values location is stored.
Real case scenario: $tt\bar{H}(bb)$ analysis in the boosted, all–hadronic final states

The MLaaS4HEP framework was tested on a real physics use-case: a signal vs background discrimination problem in a $t\bar{t}H$ (CMS) analysis. This allowed to:

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

For the validation phase 9 ROOT files were used, 8 of background and 1 of signal. Each file has 27 branches, with ~350 thousand events for the whole pool of files and a total size of ~28 MB. The ratio between signal and background is ~10.8%.
MLaaS4HEP validation

- Validate the MLaaS4HEP approach by comparing it with alternative methods on the reference use-case.
  - A simple NN with Keras in all methods has been chosen

- **Validation successful**: physics results are not impacted.

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

---

**Graphs and Figures**

- **Loss vs. # of chunks**: Shows the loss decreasing with the number of chunks.
- **AUC vs. Epoch**: Illustrates the AUC score increasing with epochs.

---

**Notes**

- **Chunk size**: Set to the total number of events.
- **Chunk size set to 10k events**: Demonstrates the impact on the AUC score.

---

**References**

- Use-case analysis by a CMS subgroup.
- Comparison with BDT-based TMVA framework.

**Conclusion**

- Validation successful with minimal impact on physics results and comparable AUC scores.
The MLaaS4HEP code has been updated to support Uproot4 and to allow users to perform pre-processing operations on the input ROOT data.

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

```json
{
    "new_branch": {
        "log_partonE": {
            "def": "log(partonE)",
            "type": "jagged",
            "cut_1": ["log_partonE<6.31", "any"],
            "cut_2": ["log_partonE>5.85", "all"],
            "remove": "False",
            "keys_to_remove": ["partonE"]},
        "nJets_square": {
            "def": "nJets**2",
            "type": "flat",
            "cut": "1<=nJets_square<=16",
            "remove": "False",
            "keys_to_remove": ["nJets"]},
        "flat_cut": {
            "nLeptons": {
                "cut": "0<=nLeptons<=2",
                "remove": "False"}}
    },
    "jagged_cut": {
        "partonPt": {
            "cut": ["partonPt>200", "all"],
            "remove": "False"}}
}
```
In the phase of **testing the MLaaS4HEP performance**, all available ROOT files without any physics cut were used. This gave a dataset with ~28.5M events with 74 branches (22 flat and 52 Jagged), and a total size of ~10.1 GB.

All the tests were performed running the MLaaS4HEP 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 (SSD storages) and remotely from the Grid sites, stored in three different data-centers located at Bologna (BO), Pisa (PI), Bari (BA).

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

In the reading phase there is a worse performance using Uproot4 than using Uproot3 but in the chunk creation phase, there is better performance with Uproot4.
  - Strong performance degradation when cuts on existing Jagged branches and on new branches are applied
Towards MLaaS4HEP cloudification

- The MLaaS4HEP performance strictly depends on the available hardware resources. How to improve it?
  - Adopt new solutions in the code
  - Invest in better and more expensive on-premise resources
  - Move to the cloud

- The operation of cloudification has two benefits.
  - Opens to potentially more performing resources
  - Opens to the creation of an “as a Service” solution

- Work towards the MLaaS4HEP cloudification using DODAS

Dynamic On Demand Analysis Service (DODAS) is a Platform as a Service tool for generating over cloud resources and on-demand, container based solution.
Creation of a **docker** image able to run the `workflow.py` script

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

Convert the Ansible playbook into an Ansible role

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

Run `workflow.py` interactively or with `jupyterhub`

Create the deployment from command line

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

Read remote ROOT files and compute specs

Write and load the specs

!/workflow.p --files=files.txt --labels=labels.txt --model=model.py --params=param.json

Datagenator: ./MLaaSHEP_generator.ROOTDatagenator object at 0x7f8b3587f140 [29/Jan/2020:17:53:14] 1539345994.4
model parameters: 
- "nevts": 30000,
- "ifile": true,
- "chunk_size": 10000,
- "epochs": 2,
- "batch_size": 100,
- "identifier": 
  - "runNo",
  - "eventNo",
  - "lumi",
  - "branch",
  - "boosted_1b/events",
  - "detected_branches": "",
  - "exclude_branches": "",
  - "hist": "pdfs",
  - "redirector": "root://xrootd.ba.infn.it/", "verbose": 1}]

Reading root://xrootd.ba.infn.it/store/user/lgiomi/TT_TuneCUETPMPT4_13TeV-powheg-pythia8.root
# 10000 entries, 77 branches, 8.785928295715332 MB, 0.959649344171143 sec, 9.249128951894 GB/sec, 10.4247313071777 kHZ
# 10000 entries, 77 branches, 8.80696062118164 MB, 1.2399239835754905 sec, 6.8544386497968 GB/sec, 7.2786180265961 kHZ
# 10000 entries, 77 branches, 8.8094496154780516 MB, 1.126785968547363 sec, 8.77143397477039 GB/sec, 8.874771534572969 kHZ
--- first pass: 1003920 events, (22-flat, 52-jagged) branches, 312 attrs

MLaaSHEP.reader.RootDataReader object at 0x7f848d13350d init is complete in 4.535124771745759 sec

write global-specs.json
load specs from global-specs.json for xrootd.ba.infn.it/store/user/lgiomi/TT_TuneCUETPMPT4_13TeV-powheg-pythia8.root
load specs from global-specs.json for xrootd.ba.infn.it/store/user/lgiomi/TT_TuneCUETPMPT4_13TeV-powheg-pythia8.root
init RootDatagenator in 11.186564683914185 sec

label 1, file <TT_TuneCUETPM2T4_13TeV-powheg-pythia8_root>, going to read 4858 events
read chunk [0:4857] from store/user/lgiomi/TT_TuneCUETPM2T4_13TeV-powheg-pythia8.root
# 10000 entries, 77 branches, 9.5222034453457 MB, 1.3816642761238469 sec, 6.89138589507834 GB/sec, 7.237648228164387 kHZ
total read 4858 evts from store/user/lgiomi/TT_TuneCUETPM2T4_13TeV-powheg-pythia8.root

label 0, file <TT_TuneCUETPM2T4_13TeV-powheg-pythia8_root>, going to read 5142 events
read chunk [4858:9999] from store/user/lgiomi/TT_TuneCUETPM2T4_13TeV-powheg-pythia8.root
# 10000 entries, 77 branches, 8.875929295715332 MB, 1.717011233026123 sec, 5.169401473297779 GB/sec, 7.824037873606205 kHZ
total read 5142 evts from store/user/lgiomi/TT_TuneCUETPM2T4_13TeV-powheg-pythia8.root

Read events from remote ROOT files, pre-process them and create the chunk
### MLaaS4HEP performance: Uproot3 vs Uproot4

<table>
<thead>
<tr>
<th></th>
<th>Uproot3</th>
<th>Uproot4</th>
</tr>
</thead>
<tbody>
<tr>
<td>reading time (s)</td>
<td>1136 (2)</td>
<td>1301 (4)</td>
</tr>
<tr>
<td>specs comp. time (s)</td>
<td>653 (1)</td>
<td>607 (1)</td>
</tr>
<tr>
<td>time to complete step 1 (s)</td>
<td>1796 (3)</td>
<td>1914 (5)</td>
</tr>
<tr>
<td>mean event throughput for reading (evts/s)</td>
<td>25304 (44)</td>
<td>21995 (73)</td>
</tr>
<tr>
<td>mean event throughput for specs comp. (evts/s)</td>
<td>43604 (50)</td>
<td>46968 (50)</td>
</tr>
<tr>
<td>mean event throughput for reading + specs comp. (evts/s)</td>
<td>16012 (24)</td>
<td>14980 (38)</td>
</tr>
<tr>
<td>event throughput for creating a chunk (evts/s)</td>
<td>1197 (5)</td>
<td>1406 (14)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>no cut</th>
<th>flat cut</th>
<th>Jagged cut</th>
<th>new branch cut</th>
<th>mixed cuts</th>
</tr>
</thead>
<tbody>
<tr>
<td>mean event throughput for reading (evts/s)</td>
<td>15157 (71)</td>
<td>15325 (56)</td>
<td>22505 (64)</td>
<td>19718 (51)</td>
<td>19375 (20)</td>
</tr>
<tr>
<td>mean event throughput for specs comp. (evts/s)</td>
<td>44004 (52)</td>
<td>43600 (136)</td>
<td>947 (4)</td>
<td>878 (11)</td>
<td>944 (5)</td>
</tr>
<tr>
<td>mean event throughput for reading + specs comp. (evts/s)</td>
<td>11273 (37)</td>
<td>11339 (29)</td>
<td>908 (3)</td>
<td>841 (10)</td>
<td>900 (5)</td>
</tr>
<tr>
<td>event throughput for creating a chunk (evts/s)</td>
<td>1363 (3)</td>
<td>1395 (8)</td>
<td>125 (1)</td>
<td>124 (1)</td>
<td>124 (1)</td>
</tr>
</tbody>
</table>
Comparison of the two MLaaS4HEP training procedures

Chunk size 100 events

Chunk size 100k events

(a)  
(b)  
(c)  
(d)  

CHEP, 8-12 May 2023
What is MLaaS4HEP?

[Diagram of the MLaaS4HEP process]

Client

Submit POST API

{ files: files.txt, labels: labels.txt, model: model.py, params: params.json, other parameters: ... }

OAuth2 Proxy server

MLaaS4HEP server

get an X.509 proxy for the remote access to ROOT data

Cloud resources

save trained ML model

get the ML model

return prediction

[Diagram of the MLaaS4HEP process]

Client

Predict POST API

{ process name: ... }

{ model: ... , event: ... }

OAuth2 Proxy server

trained ML models repository