

TensorFlow as a Service (TFaaS)

Valentin Kuznetsov, Cornell University

CMS ML Workshop, 2018

Evolution of HEP x ML Engineering

Data Layer		
ROOT Files	ROOT Files	DB / HDFS etc.
Loading Layer		
Ad hoc ROOT ETL logic	Numpy / HDF5 Converters / Loaders	Numpy / HDF5 Converter Loaders
Training Layer		
TMVA	Keras, TensorFlow, PyTorch, XGBoost, scikit-learn, ...	Keras, TensorFlow, PyTorch, XGBoost, scikit-learn, ...
Serving Layer		
Deployment Target (TMVA)	Deployment Target (lwttn, TensorFlow, TMVA wrappers)	Deployment Target (TensorFlow Serving, SageMaker, etc.)
HEP (Circa 2013)	HEP (Circa 2018)	Industry

- *ML as a Service (MLaaS)*: current cloud providers rely on a MLaaS model exploiting interactive machine learning tools in order to make efficient use of resources, however, this is not yet widely used in HEP. HEP services for interactive analysis, such as CERN's Service for Web-based Analysis, SWAN [62], may play an important role in adoption of machine learning tools in HEP workflows. In order to use these tools more efficiently, sufficient and appropriately tailored hardware and instances other than SWAN will be identified.

ML as an Application Layer

- Very common in industry - decouple runtime envs
 - Entire backend written in Go, but I need to use scikit-learn, now what?
- ATLAS / CMS / broader CERN community faces this with ML!

Every ML practitioner faces a challenges how to integrate their favorite ML model(s) into existing infrastructure

Building a ML Serving Layer

- GRPC [Google], low-latency network calls with type-safe data structures called protobufs (support for Python, C++, ...)
 - How large portions of Google do ML
- Message Queues can allow the parent application process to submit data to a queue, and have the ML application layer dequeue and perform the task

<https://bit.ly/2ssmBcG>

MLaaS on a market

CLOUD MACHINE LEARNING SERVICES COMPARISON

	Amazon ML	Amazon SageMaker*	Azure ML Studio	Google Prediction API	Google ML Engine**
Classification	✓	✓	✓	✓	✓
Regression	✓	✓	✓	✓	✓
Clustering	✗	✓	✓	✗	✓
Anomaly detection	✗	✓	✓	✗	✓
Recommendation	✗	✓	✓	✗	✓
Ranking	✗	✓	✓	✗	✓
Algorithms	unknown	10 built-in + custom available	100+ algorithms and modules	unknown	TensorFlow-based
Frameworks	✗	TensorFlow, MXNet	✗	✗	TensorFlow
Graphical interface	✗	✗	✓	✗	✗
Automation level	high	medium	low	high	low

- ❖ All major cloud providers serve MLaaS
- ❖ TF is a major framework to be used at MLaaS
- ❖ There are may be constraints on ML model, framework or data to be served / trained
- ❖ Predictions can be costly, e.g. at AWS \$0.10 / 1000 predictions using batch or \$0.0001 / prediction for real time

Integration approach

TensorFlow Interface for CMSSW

pipeline **passed**

- Main repository & issues: gitlab.cern.ch/mrieger/CMSSW-DNN
- Code mirror: github.com/riga/CMSSW-DNN

Note

The interface was merged under [PhysicsTools/TensorFlow](#) on Jan 25 2018 into [CMSSW_10_1_X](#) and backported to [CMSSW_9_4_X](#) on Feb 15 2018. For development purposes, the include paths in this repository point to [DNN/TensorFlow](#).

This interface provides simple and fast access to [TensorFlow](#) in CMSSW and lets you evaluate trained models right within your C++ modules. It **does not depend** on a converter library or custom NN implementation. In fact, it is a thin layer on top of TensorFlow's C++ API (available via externals in [/cvmfs](#)) which handles session / graph loading & cleanup, exceptions, and thread management within CMSSW. As a result, you can load and evaluate every model that was previously trained and saved in Python (or C++).

Due to the development of the CMS software environment since 8_0_X, there are multiple versions of this interface. But since the C++ API was added in 9_4_X, the interface API is stable and should handle all changes within TensorFlow internally. The following table summarizes all available versions, mapped to CMSSW version and SCRAM_ARCH:

CMSSW version	SCRAM_ARCH	TF API & version (externals)	Interface branch
t.b.a.	slc6_amd64_gcc630	C++, 1.6.0	tf_cc_1.6
10_1_X	slc6_amd64_gcc630	C++, 1.5.0	tf_cc_1.5
9_4_X	slc6_amd64_gcc630	C++, 1.3.0	tf_cc_1.3
9_3_X	slc6_amd64_gcc630	C, 1.1.0	tf_c
8_0_X	slc6_amd64_gcc530	Py + CPython, 1.1.0	tf_py_cpython

Lightweight Trained Neural Network

build **passing** coverity **passed** DOI 10.5281/zenodo.597221

What is this?

ATLAS integrated solution

The code comes in two parts:

1. A set of scripts to convert saved neural networks to a standard JSON format
2. A set of classes which reconstruct the neural network for application in a C++ production environment

The main design principles are:

- **Minimal dependencies:** The C++ code depends on C++11, [Eigen](#), and boost [PropertyTree](#). The converters have additional requirements (Python3 and h5py) but these can be run outside the C++ production environment.
- **Easy to extend:** Should cover 95% of deep network architectures we would realistically consider.
- **Hard to break:** The NN constructor checks the input NN for consistency and fails loudly if anything goes wrong.

We also include converters from several popular formats to the `lwttn` JSON format. Currently the following formats are supported:

- [AGILEPack](#)
- [Keras](#) (most popular, see below)

Why are we doing this?

Our underlying assumption is that *training* and *inference* happen in very different environments: we assume that the training environment is flexible enough to support modern and frequently-changing libraries, and that the inference environment is much less flexible.

If you have the flexibility to run any framework in your production environment, this package is *not* for you. If you want to apply a network you've trained with Keras in a 6M line C++ production framework that's only updated twice a year, you'll find this package very useful.

MLaaS vs integration approach

MLaaS

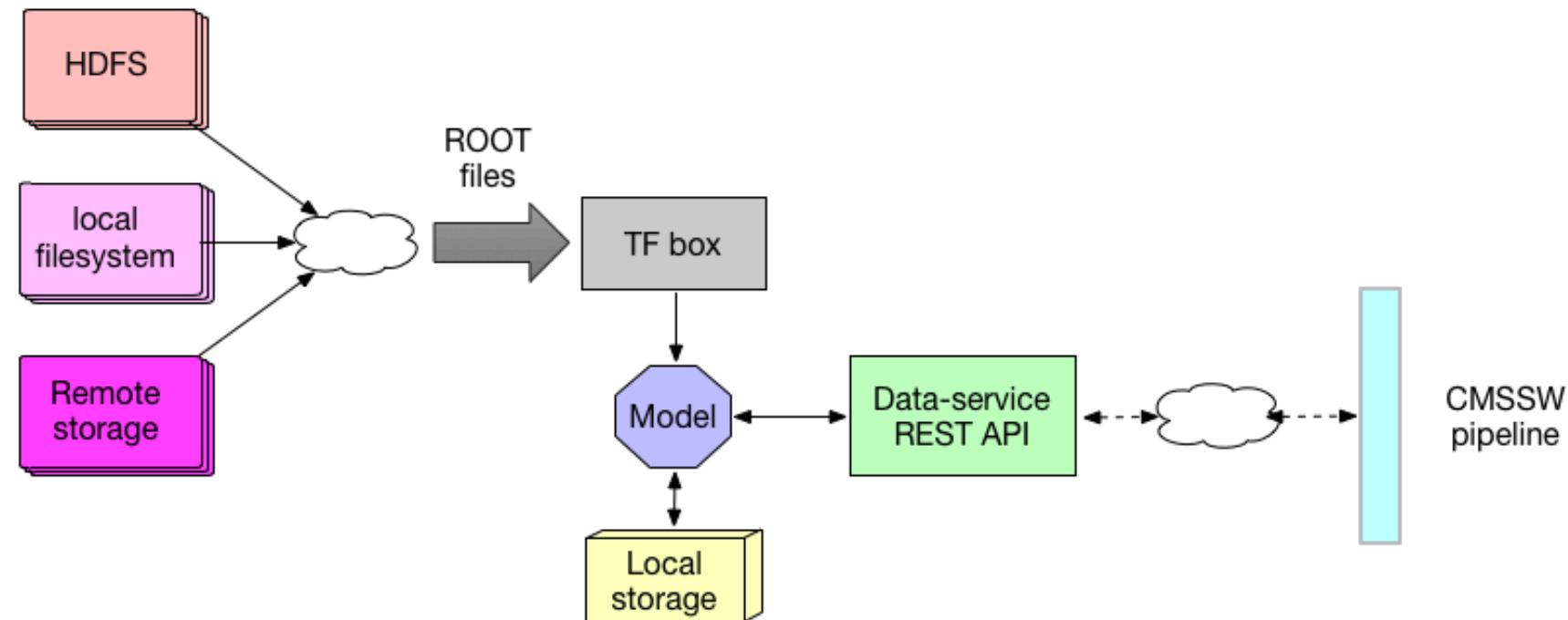
- ❖ Separation of layers, code maintenance and framework independent
- ❖ Decoupling of hardware resources, i.e. your run-time resources are not constrained by inference
- ❖ ML models can be separated from run time environment
- ❖ Centrally managed ML models
- ❖ Easy to maintain multiple or frequently changed ML models
- ❖ Works well in distributed environment, e.g. clients talk to central service
- ❖ High throughput achievable via horizontal scaling

Integrated approach

- ❖ Close integration of ML within existing framework: [CMSSW-DNN](#) or [LWTNN](#) (ATLAS)
- ❖ Either work within the same language or extra wrapper / translation are required
- ❖ Job resources are limited or should be controlled
- ❖ ML models should be reachable at run time environment
- ❖ Locally managed ML models and *assume* that ML models will not change often
- ❖ Harder to maintain and work with distributed environment
- ❖ High throughput within a job since ML is integrated within framework

TensorFlow as a Service (TFaaS) for CMS experiment

CMS experiment at CERN use various Machine Learning (ML) techniques, including DNN, in various physics and computing related projects. The popular TensorFlow Google framework is an excellent choice to apply ML algorithms for using in CMS workflow pipeline. The project intends to build end-to-end data-service to serve TF trained model for CMSSW framework. The overall architecture is shown below:




The projects will explore and implement the following topics

- build data-service which will read ROOT file(s) to train ML model
 - read ROOT files from python via [uproot](#)
- serve ML model via REST API, aka *Machine Learning as a Service*
- read ML model either via [CMSSW-DNN](#) framework or from external data-service (MLaaS or TFaaS) and demonstrate its usage for HEP
- explore Cloud and custom solution for TF back-end as well as [distributed Keras](#) framework on Spark cluster
- port [fast.ai/PyTorch](#) models into [Keras/TensorFlow](#). The PyTorch framework provides dynamic models while TF are static one (compiled). We need to find a way to port saved PyTorch models (in Protobuffer format) into TF one in order to serve them in TFaaS. This has been discussed in [fast.ai course](#) as well as there is preliminary ideas [here](#) and [here](#). See also discussion about [PyTorch vs TensorFlow](#).

The TFaaS demonstrator instruction is available [here](#)

TensorFlow as a Service (TFaaS)

[Home](#) [Download](#) [Models](#) [FAQ](#) [Contact](#)



SCALABLE AND EFFICIENT

TFaaS built using modern technologie and scale along with your hardware. It does not lock you into specific provider. Deploy it at your premises and control your use-case usage.

[SHOW ME](#)

REACH APIS

TFaaS provides reach and flexible set of APIs to efficiently manage your TF models. The TFaaS web server supports JSON or Protobuffer data-formats to support your clients.

[SHOW ME](#)

FROM DEPLOYMENT TO PRODUCTION

- 1 Deploy docker image:

```
docker run --rm -h `hostname -f` -p 8083:8083 -i -t veknet/tfaas
```
- 2 Upload your model:

```
curl -X POST http://localhost:8083/upload -F 'name=ImageModel' -F 'params=@/path/params.json' -F 'model=@/path/tf_model.pb' -F 'labels=@/path/labels.txt'
```
- 3 Get predictions:

```
curl https://localhost:8083/image -F 'image=@/path/file.png' -F 'model=ImageModel'
```

Flexible configuration parameters allows you to adopt TFaaS deployment to any use case.

TFaaS features

- ❖ HTTP server (written in Go) to serve **any** TF models
 - ❖ users may upload **any number** of TF models, models are stored on local filesystem and cached within TFaaS server
 - ❖ TFaaS provides instructions/tools to convert Keras models to TF
 - ❖ TFaaS supports JSON and ProtoBuffer data-formats
- ❖ Separate model training from inference, train your model on your GPUs locally or remotely, upload them to TFaaS server and serve the predictions via TFaaS APIs
- ❖ Clients only required to support HTTP protocol, e.g. curl, C++ (via curl library or TFaaS C++ client), Python can talk to TFaaS via HTTP APIs
 - ❖ C++ client library talks to TFaaS via ProtoBuffer data-format, all other clients uses JSON
- ❖ Benchmarked with 200 concurrent calls, observed throughput **500 requests/second** for tested TF model
 - ❖ performance are similar to JSON and ProtoBuffer clients

Python client: gihub repository

-
- ❖ **upload** API lets you upload your model and model parameter files to TFaaS

```
tfaas_client.py --url=url --upload=upload.json
```

- ❖ **models** API lets you list existing models on TFaaS server

```
tfaas_client.py --url=url --models
```

- ❖ **delete** API lets you delete given model on TFaaS server

```
tfaas_client.py --url=url --delete=ModelName
```

- ❖ **predict** API lets you get prediction from your model and your given set of input parameters

```
# input.json: {"keys":["attr1", "attr2", ...], "values": [1,2,...]}
```

```
tfaas_client.py --url=url --predict=input.json
```

- ❖ **image** API provides predictions for image classification

```
tfaas_client.py --url=url --image=/path/file.png --model=HEPImageModel
```


C++ client

CMS BuildFile.xml

```
<use name="protobuf"/>
<lib name="curl"/>
<lib name="protobuf"/>
```

```
#include <iostream>
#include <vector>
#include <sstream>
#include "TFClient.h"

// main function
int main() {
    std::vector<std::string> attrs;
    std::vector<float> values;
    auto url = "http://localhost:8083/proto";
    auto model = "MyModel";

    // fill out our data
    for(int i=0; i<42; i++) {
        values.push_back(i);
        std::ostringstream oss;
        oss << i;
        attrs.push_back(oss.str());
    }

    // make prediction call
    auto res = predict(url, model, attrs, values); // get predictions from TFaaS
    for(int i=0; i<res.prediction_size(); i++) {
        auto p = res.prediction(i); // fetch and print model predictions
        std::cout << "class: " << p.label() << " probability: " << p.probability() << std::endl;
    }
}
```

```
// include TFClient header

// define vector of attributes
// define vector of values
// define your TFaaS URL
// name your model

// the model I tested had 42 parameters
// create your vector values

// create your vector headers
```


TFaaS APIs: available end-points

- ❖ **json**: handles data send to TFaaS in JSON data-format, e.g. you'll use this API to fetch predictions for your vector of parameters presented in JSON data-format (used by Python client)
- ❖ **proto**: handlers data send to TFaaS in ProtoBuffer data-format (user by C++ client)
- ❖ **image**: handles images (png and jpg) and yields predictions for given image and model name
- ❖ **upload**: upload your TF model to TFaaS
- ❖ **delete**: delete TF model on TFaaS server
- ❖ **models**: return list of existing TF models on TFaaS
- ❖ **params**: return list of parameters of TF model
- ❖ **status**: return status of TFaaS server

Use cases

- ❖ TFaaS provides framework independent access to your TF models
 - ❖ easy to integrate into your workflow, either Python or C++
 - ❖ C++ client library can be used to integrate within CMSSW as well
- ❖ Rapid development of TF models and their verifications is ideal use-case for TFaaS
 - ❖ clients can test multiple TF models at the same time
- ❖ Integration of ML into existing infrastructure without extra development and maintenance effort to support ML infrastructure
- ❖ TFaaS deployment is trivial (via docker) and you can setup your TFaaS server at your premises, e.g. on your local hardware or at a cloud provider
- ❖ TFaaS fits well in distributed environment where clients can connect to central TFaaS server(s) via HTTP APIs

Summary

- ❖ TFaaS is a general purpose, framework independent, HTTP based service to serve any TF models
- ❖ TFaaS server natively supports concurrency, it organizes TF models in hierarchical structure on local file system, and it uses cache to serve TF models to end-users
 - ❖ no integration is required to include TFaaS into your infrastructure, i.e. clients talk to TFaaS server via HTTP protocol (python and C++ clients are available)
 - ❖ allow separation TF models from CMSSW framework, do not use CMSSW run-time resources, dedicated resources can be used to serve any number of TF models we may need at run-time
 - ❖ can be used as model repository, TFaaS architecture allows to implement model versioning, tagging, ...
- ❖ TFaaS server was tested with heavy loaded concurrent clients and delivery 500 req/sec throughput (subject of TF model complexity)
- ❖ TFaaS is available on <https://cmsweb-testbed.cern.ch/tfaas/> for CMS end-users and we plan to put it to production (cmsweb). Feel free to upload your model and test it (open GitHub issue ticket if necessary).

R&D topics

- ❖ Model conversion: PyTorch / fast.ai to TensorFlow
- ❖ Model repository: implement persistent model storage, versioning, tagging, etc.
- ❖ TFaaS clustering: explore kubernetes and auto-scaling
- ❖ Training model with distributed data: read CMS ROOT files and feed them into TFaaS for continuous training

Demo

How to build and run from source code

<https://drive.google.com/file/d/1Ipwt9dOJCCb9EN3lmiYKhExel6dd4baO/view>

Client-server interaction

curl client: <https://www.youtube.com/watch?v=ZGjnM8wk8eA>

python client: <https://youtu.be/ZhD2jEqc0Fw>

Back-up slides

Example of model parameters

upload.json: describe input model

```
{  
  "model": "/opt/cms/models/vk/model.pb",  
  "labels": "/opt/cms/models/vk/labels.txt",  
  "name": "vk",  
  "params": "/opt/cms/models/vk/params.json"  
}
```

params.json: describe model parameters

```
{  
  "name": "vk",  
  "model": "model.pb",  
  "description": "vk test model",  
  "labels": "labels.txt",  
  "inputNode": "dense_1_input",  
  "outputNode": "output_node0"  
}
```


Example of input.json

```
{  
  
  "keys": ["nJets", "nLeptons", "jetEta_0", "jetEta_1", "jetEta_2",  
    "jetEta_3", "jetEta_4", "jetMass_0", "jetMass_1", "jetMass_2",  
    "jetMass_3", "jetMass_4", "jetMassSoftDrop_0",  
    "jetMassSoftDrop_1", "jetMassSoftDrop_2", "jetMassSoftDrop_3",  
    "jetMassSoftDrop_4", "jetPhi_0", "jetPhi_1", "jetPhi_2",  
    "jetPhi_3", "jetPhi_4", "jetPt_0", "jetPt_1", "jetPt_2", "jetPt_3",  
    "jetPt_4", "jetTau1_0", "jetTau1_1", "jetTau1_2", "jetTau1_3",  
    "jetTau1_4", "jetTau2_0", "jetTau2_1", "jetTau2_2", "jetTau2_3",  
    "jetTau2_4", "jetTau3_0", "jetTau3_1", "jetTau3_2", "jetTau3_3",  
    "jetTau3_4"],  
  
  "values": [2.0, 0.0, 0.9228423833849999, -1.1428750753399999, 0.0, 0.0,  
    0.0, 155.239425659, 142.709609985, 0.0, 0.0, 0.0, 83.5365982056,  
    120.549507141, 0.0, 0.0, 0.0, 1.9305502176299998, -1.17742347717, 0.0,  
    0.0, 0.0, 481.419799805, 449.04394531199995, 0.0, 0.0, 0.0,  
    0.296700358391, 0.286615312099, 0.0, 0.0, 0.0, 0.164555206895,  
    0.19625715911400002, 0.0, 0.0, 0.0, 0.117722302675, 0.155229091644, 0.0,  
    0.0, 0.0],  
  
  "model": "vk"}  
}
```