MLTF.VU

Machine **L**earning **T**raining **F**acility at **V**anderbilt **U**niversity

Jethro Gaglione, Andrew Melo, Satkar Dhakal, Lindsey Fox, Matt Heller, Prasanna Koirala, Samuel Ninchritz, Brandon Soubasis, Alan Tackett

Motivation and Facility Overview

- Machine learning/AI has been widely adapted for an array of uses in HEP and beyond.
- However, there are considerations about <u>training</u> these models -
  - *<u>Reproducibility</u> - Can someone later consistently produce the same weights?*
  - *<u>Scalability</u> - How does one scale the training from card->node->cluster scale?*
  - *<u>Efficiency</u> - What are the barriers to using high-speed transports and interconnects?*
- MLTF is a prototype facility that enables training w/these considerations in mind -
  - *<u>Hardware</u> - At this stage, capabilities were chosen over large scale*
  - *<u>Software</u> - Easy-to-use and documented hooks/wrappers for clients to enable functionality*
  - *<u>Infrastructure</u> - Services run at the site to connect the client software to hardware resources*

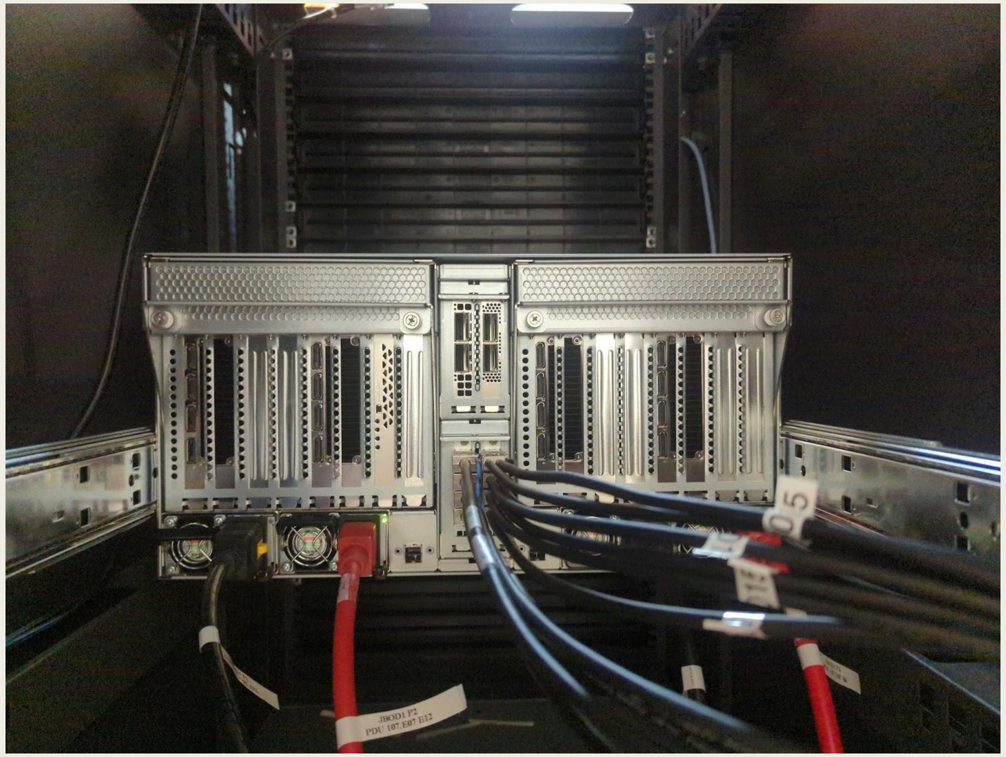
Motivation and Facility Overview

- MLTF seeks to provide a higher-level abstraction to the user
  - *Users submit training tasks, not batch jobs*
  - *Tasks describe environment, code, inputs, HW reqs*
  - *Facility takes this description, executes jobs and returns outputs*
- *Roughly* analogous to containerization
  - *Container engine needs to know little about host environment or even execution engine and vice versa*
  - *Similarly, user doesn't have to opt-in*
    - docker run --net=host
- This abstraction benefits both the user and the facility
  - *Facility can use higher-level knowledge to optimize/configure as-needed (e.g. pre-staging input data, configuring NCCL, scheduling decisions)*
  - *User doesn't need to know all implementation details*
- Helps reproducibility - Outputs tied back to task that produced them

https://www.evergreen-line.com/container/images/CNTR_Overive.jpg

Hardware

- The proof-of-concept facility will have 16xA6000s (w/NVLink), RoCE networking, and 16 large NVME disks:
  - RoCE allows RDMA access over an ethernet network for low-latency, high-performance transmission.
  - NVMe allows for direct GPU access to data with high throughput and fast response times.
- The components are housed in PCIe enclosures, which connect to compute nodes via PCIe expander cards
- We would like to acquire PCIe switches, which will allow us to scale out the PCIe fabric to an arbitrary number of enclosures and compute hardware.
- Delays in receiving hardware means a lot yet to explore in terms of functionality 🙁
- There is a possibility for “opportunistic” access to other hardware @ Vanderbilt, including O(100) GPU, DGX hardware, Grace Hopper GH200 machines

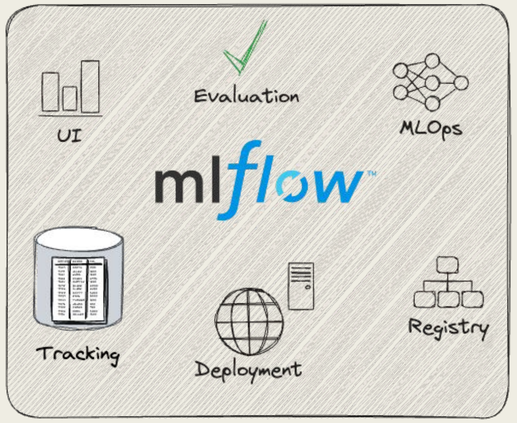

PCIe enclosure. Thick black cables in the middle route to compute hardware.

Training Framework Considerations

- To begin this work, we need to select a training framework.
  - *<u>Important</u>: this is not a decision about ML library (e.g. Tensorflow/Keras/etc..)*
- With the ongoing gold rush, there's more than a few frameworks available
  - Nearly all of them are "semi-open source"
    - Lock-in / price become a risk if you want to use "interesting" features
  - Many are additionally exclusively Kubernetes-based, meaning the GPUs are provisioned via Kubernetes.
    - This is traditionally not how we deploy resources in HPC centers and would imply segregating (expensive) hardware in many places.
    - K8S scheduling not as robust as Condor/SLURM – the assumption is that if you have more jobs than resources, you just rent more cloud resources. Certainly not usually the case for academia!
- After evaluating several options, <u>MLFlow</u> was chosen as the preferred training framework

MLflow

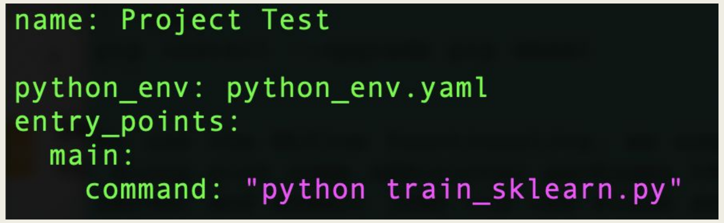
- MLflow is a solution for the entire lifecycle of ML:
  - A job submission interface
  - Tracking server which can store:
    - Inputs - Python code, environment, input files/datasets
    - Outputs - Trained weights
    - Diagnostics - CPU/GPU utilization, loss metrics, time per epoch, etc..
  - Inference server - effectively a REST wrapper around a model
- Ticks a lot of necessary boxes
  - *Open-source, with public Helm charts for deployment*
  - *Right mental model: An MLflow "project" is a training "task"*
  - *Minimal changes to existing code required to integrate*
    - File describing Project metadata, two lines of python in user code
  - *Robust REST API*
  - *Defined & Documented extension API*
- Not the first in academia to come to this same conclusion. NCSA also uses MLflow as their preferred training framework(NCSA docs)
  - *SLURM & Token authentication plugins developed by NCSA*



https://mlflow.org/docs/latest/index.html

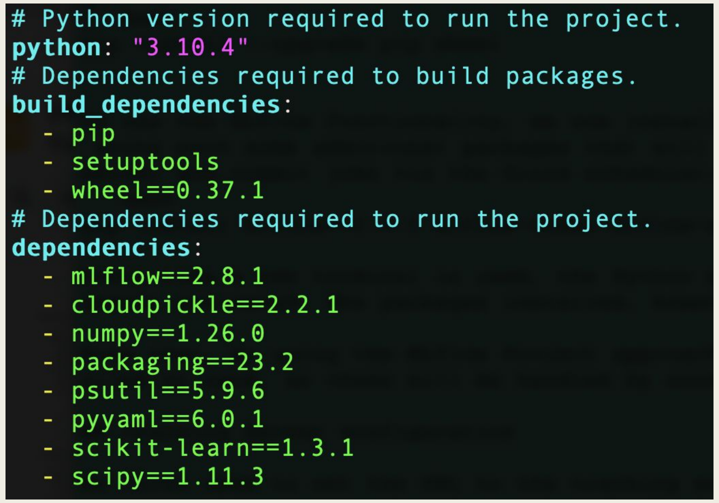
MLflow Projects Example

- Minimal MLflow Project is simple
  - Directory containing four files: “*MLproject*” entry file, “*my_env.yaml”* environment req. specs, “slurm_config.json” for job req., and your training script.
  - Can be local or in VCS (e.g. Git)
- Multi-step workflows are supported natively - e.g. feature extraction followed by training
- Have adapted an in-production CMS ML production workflow to use MLflow
- Future work:
  - *Support rucio:// URIs for input files*
  - *Support parallelism via Dask/Luigi*

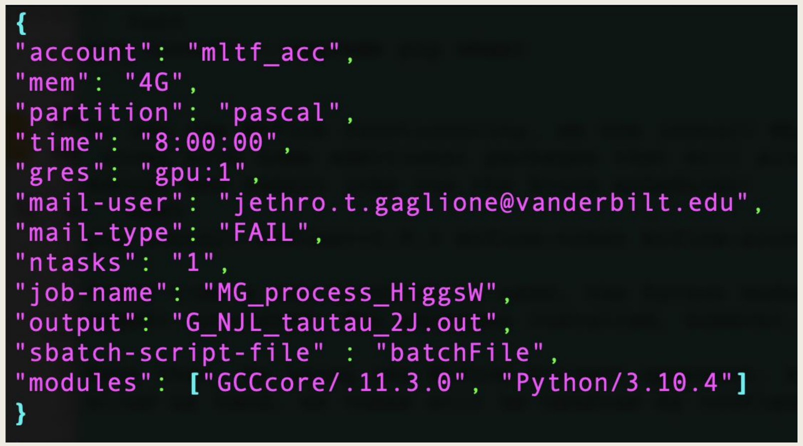


```yaml
name: Project Test

python_env: python_env.yaml
entry_points:
  main:
    command: "python train_sklearn.py"
```

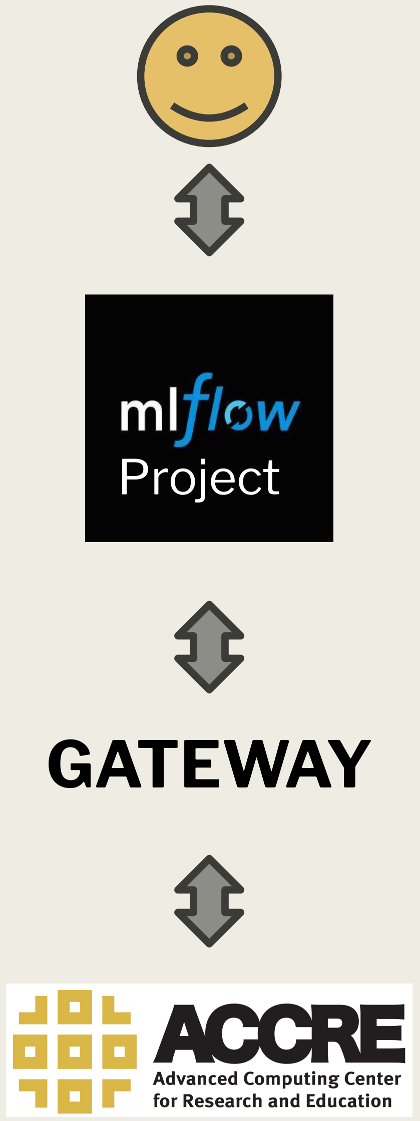


```yaml
# Python version required to run the project.
python: "3.10.4"
# Dependencies required to build packages.
build_dependencies:
  - pip
  - setuptools
  - wheel==0.37.1
# Dependencies required to run the project.
dependencies:
  - mlflow==2.8.1
  - cloudpickle==2.2.1
  - numpy==1.26.0
  - packaging==23.2
  - psutil==5.9.6
  - pyyaml==6.0.1
  - scikit-learn==1.3.1
  - scipy==1.11.3
```



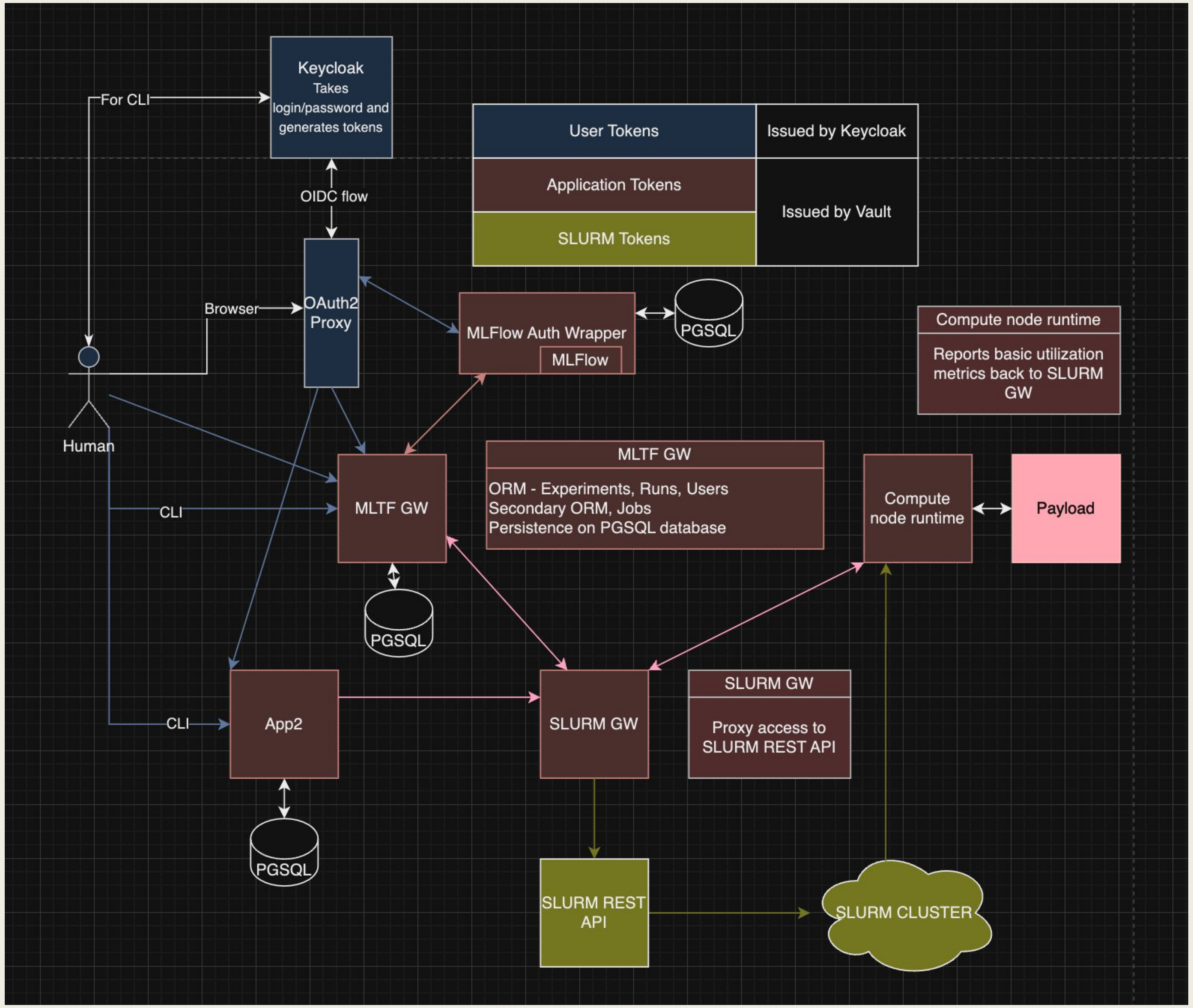
```json
{
"account": "mltf_acc",
"mem": "4G",
"partition": "pascal",
"time": "8:00:00",
"gres": "gpu:1",
"mail-user": "jethro.t.gaglione@vanderbilt.edu",
"mail-type": "FAIL",
"ntasks": "1",
"job-name": "MG_process_HiggsW",
"output": "G_NJL_tautau_2J.out",
"sbatch-script-file" : "batchFile",
"modules": ["GCCcore/.11.3.0", "Python/3.10.4"]
}
```

MLflow Gateway



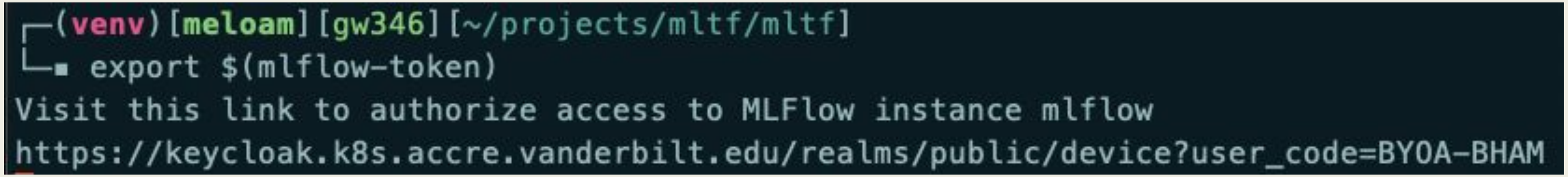
- Remote submission of MLflow projects opens interesting use-cases
  - *No need to get local accounts @ target cluster*
  - *CI/CD applications*
- MLflow has built-in support for remote submission via DataBricks
  - *Extension API allows this to be pluggable*
  - *Support for SLURM submission exists due to MLflow-SLURM*
- We are developing MLflow Gateway, a Client/Server interface to job submission
  - *Client side is an MLflow backend plugin*
  - *Server side is a REST service which interacts with local cluster*
- Since the gateway is an "intermediary" between the client and the facility, this provides a hook to add customization to the jobs before they are submitted to the batch system
  - *Difficult to do in direct SLURM submission*
  - *A task requesting a Rucio dataset as an input can trigger the Rucio rule, and wait for the replication to complete before submitting jobs to batch system*

Mlflow Gateway

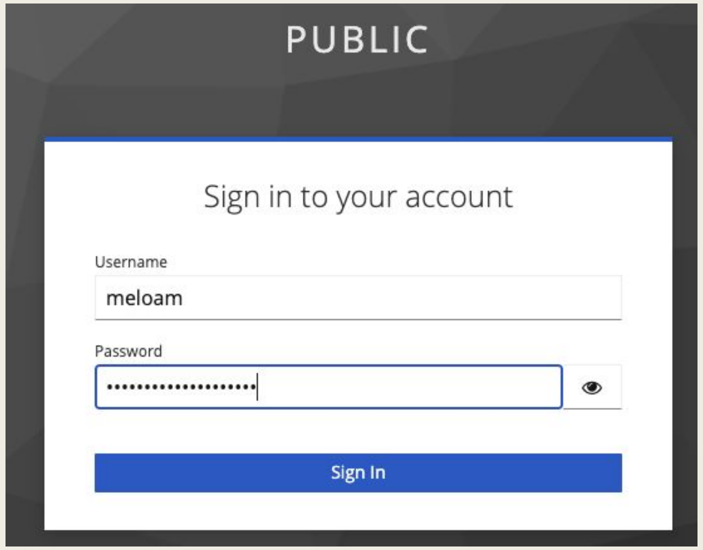


- Most components are existing, available, open-source projects.
- Authentication is token-based, using familiar SSO pages.
- All user code is run within a container to ensure isolation.
  - Either user-provided container or user python env within a container

Authentication Flow



```
┌─(venv)[meloam][gw346][~/projects/mltf/mltf]
└─▪ export $(mlflow-token)
Visit this link to authorize access to MLFlow instance mlflow
https://keycloak.k8s.accre.vanderbilt.edu/realms/public/device?user_code=BYOA-BHAM
```



PUBLIC

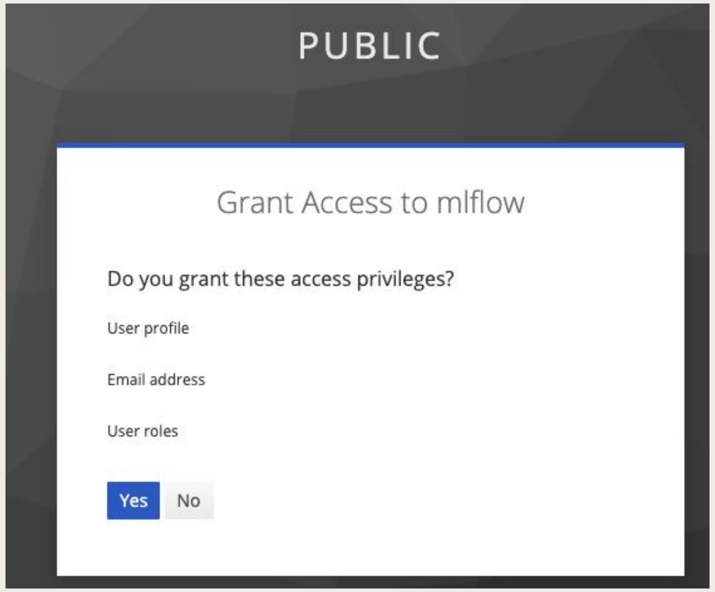
Sign in to your account

Username

meloam

Password

••••••••••••••••••••



PUBLIC

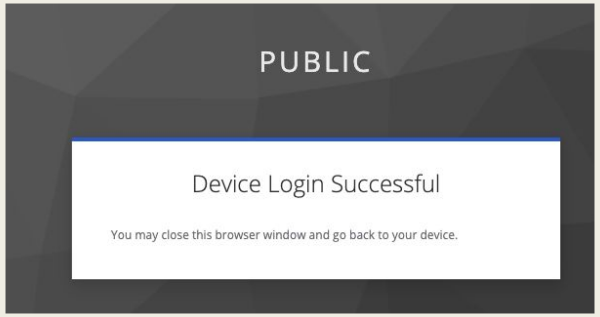
Grant Access to mlflow

Do you grant these access privileges?

User profile

Email address

User roles



PUBLIC

Device Login Successful

You may close this browser window and go back to your device.

MLflow Deployement

- Central components (MLFlow tracking, databases, artifact storage) are deployed in k8s, while GPU remains in SLURM cluster
- Currently we have SLURM submission functional, which requires access to Vanderbilt's cluster
- Goal is to have the whole stack (MLflow deployment + Gateway) packaged as Helm chart so it can be deployed elsewhere w/a k8s cluster and SLURM

Training

Submitting Jobs and Viewing Results

- After setting the tracking server,

```
export MLFLOW_TRACKING_URI=https://mlflow-test.mltf.k8s.accre.vanderbilt.edu
```

activating the required token,

```
export $(mlflow-token)
```

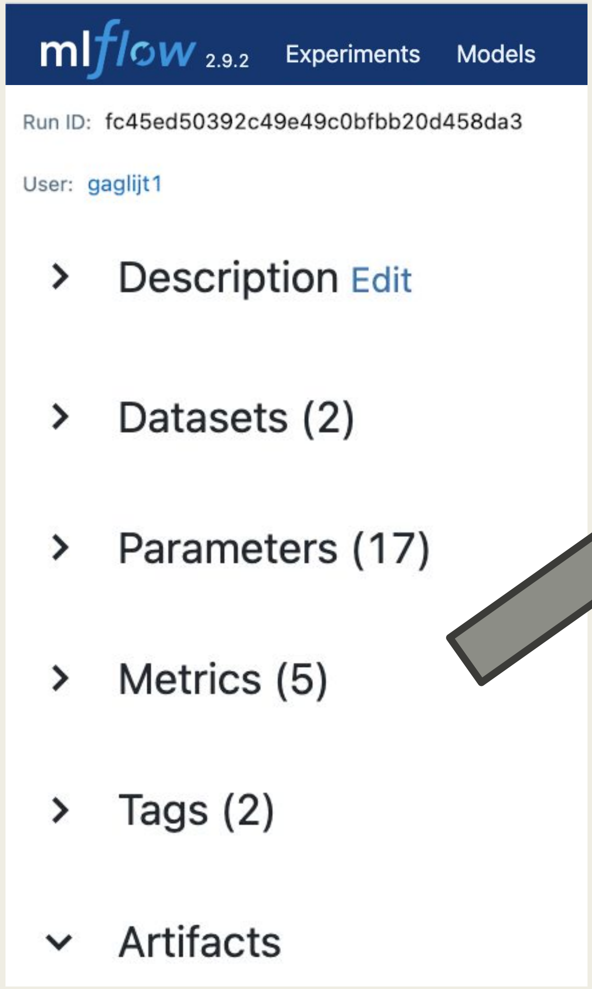
and running:

```
mlflow run --backend slurm --backend-config <path to slurm_config.json> <path to MLflow Project dir.>
```

- The resulting model weights/parameters, tracked model metrics, system metrics, and a range of other selected artifacts can be viewed/accessed in the MLflow UI.
- Code to activate/customize tracking and logging functionalities is inserted in the training script. Popular ML packages (Scikit-learn, Keras, Pytorch Lightning) have autolog support, used by simple adding the following to the Python training script:

```
import mlflow
mlflow.autolog()
```

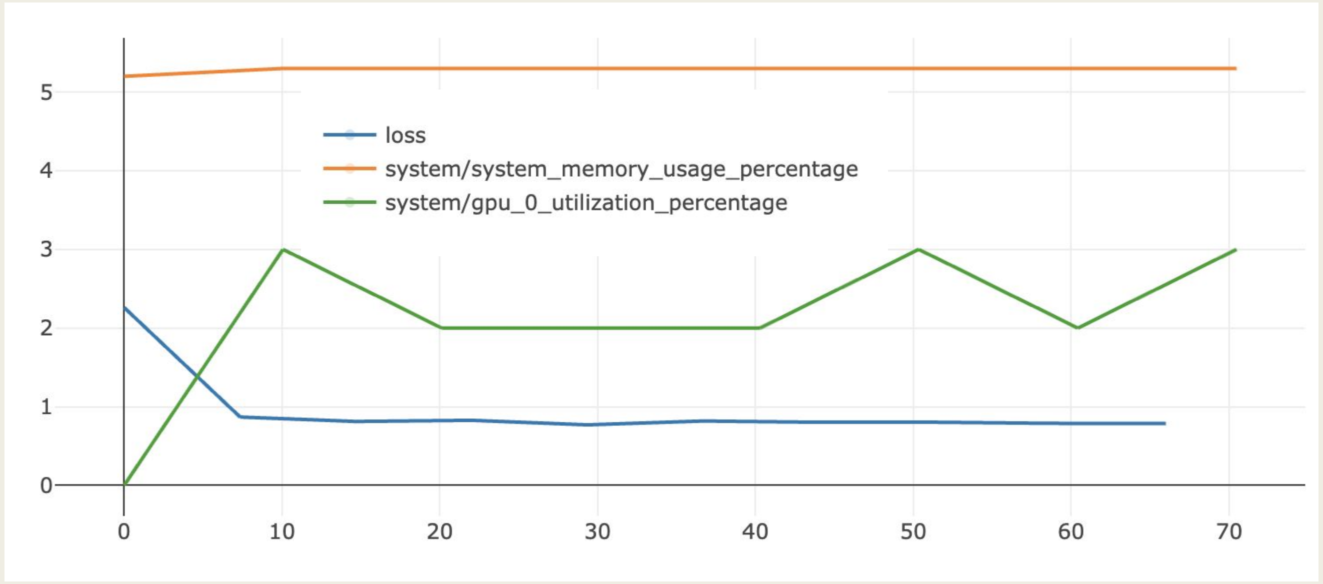
Mlflow User Interface



Run ID: fc45ed50392c49e49c0bfbb20d458da3

User: gaglijt1

- Description Edit
- Datasets (2)
- Parameters (17)
- Metrics (5)
- Tags (2)
- Artifacts

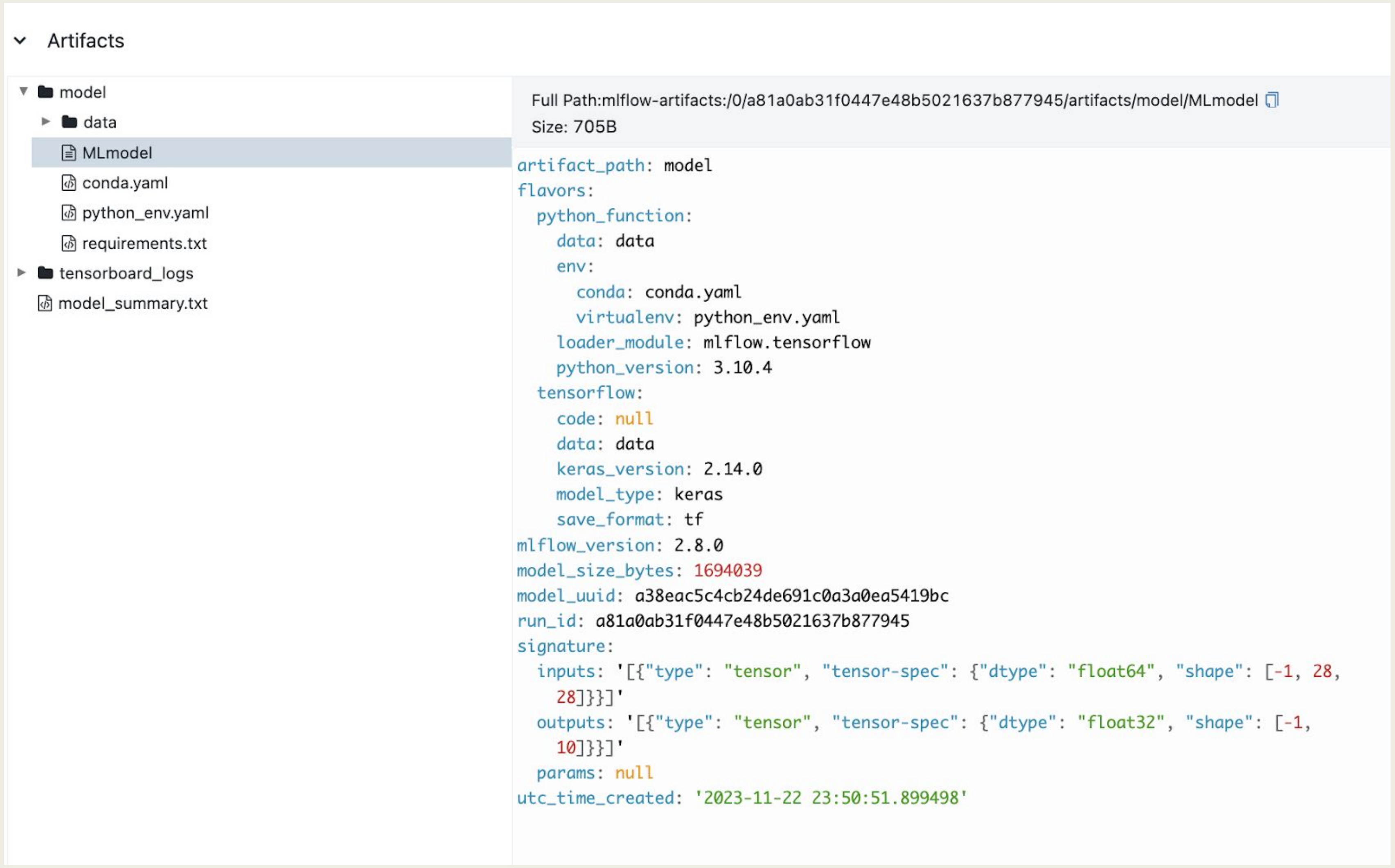


- Detailed description of training conditions, e.g. date/time, duration, source code filename, user…
- Option to save datasets
- Automatic or custom logging of parameters and metrics of interest.

Mlflow User Interface

- Complete set of information on the trained model.

- Environment information along with "requirements.txt" file.



Artifacts

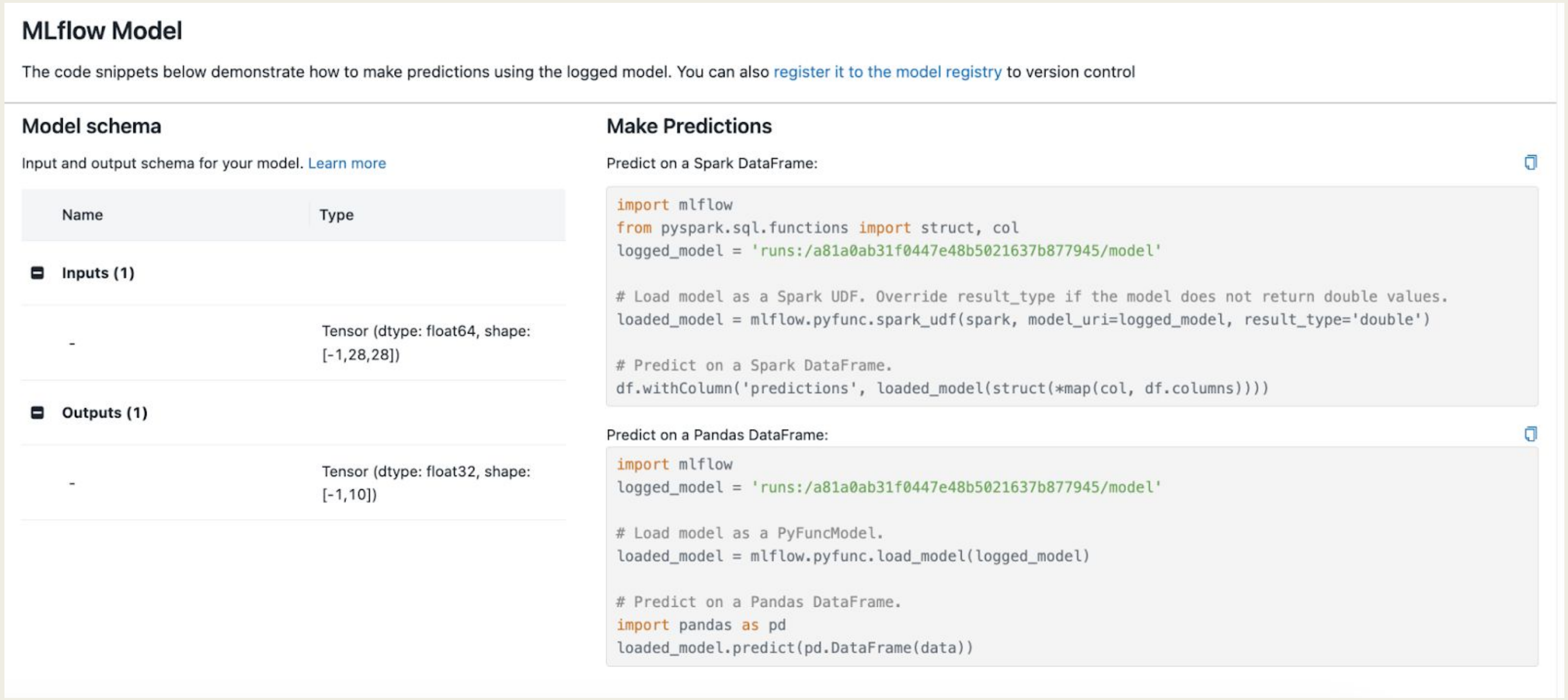
- model
  - data
  - MLmodel
  - conda.yaml
  - python_env.yaml
  - requirements.txt
- tensorboard_logs
- model_summary.txt

Full Path:mlflow-artifacts:/0/a81a0ab31f0447e48b5021637b877945/artifacts/model/MLmodel

Size: 705B

```
artifact_path: model
flavors:
  python_function:
    data: data
    env:
      conda: conda.yaml
      virtualenv: python_env.yaml
    loader_module: mlflow.tensorflow
    python_version: 3.10.4
  tensorflow:
    code: null
    data: data
    keras_version: 2.14.0
    model_type: keras
    save_format: tf
mlflow_version: 2.8.0
model_size_bytes: 1694039
model_uuid: a38eac5c4cb24de691c0a3a0ea5419bc
run_id: a81a0ab31f0447e48b5021637b877945
signature:
  inputs: '[{"type": "tensor", "tensor-spec": {"dtype": "float64", "shape": [-1, 28,
    28]}}]'
  outputs: '[{"type": "tensor", "tensor-spec": {"dtype": "float32", "shape": [-1,
    10]}}]'
  params: null
utc_time_created: '2023-11-22 23:50:51.899498'
```

Mlflow User Interface



MLflow Model

The code snippets below demonstrate how to make predictions using the logged model. You can also register it to the model registry to version control

Model schema

Input and output schema for your model. Learn more

Name	Type
Inputs (1)	
-	Tensor (dtype: float64, shape: [-1,28,28])
Outputs (1)	
-	Tensor (dtype: float32, shape: [-1,10])

Make Predictions

Predict on a Spark DataFrame:

```
import mlflow
from pyspark.sql.functions import struct, col
logged_model = 'runs:/a81a0ab31f0447e48b5021637b877945/model'

# Load model as a Spark UDF. Override result_type if the model does not return double values.
loaded_model = mlflow.pyfunc.spark_udf(spark, model_uri=logged_model, result_type='double')

# Predict on a Spark DataFrame.
df.withColumn('predictions', loaded_model(struct(*map(col, df.columns))))
```

Predict on a Pandas DataFrame:

```
import mlflow
logged_model = 'runs:/a81a0ab31f0447e48b5021637b877945/model'

# Load model as a PyFuncModel.
loaded_model = mlflow.pyfunc.load_model(logged_model)

# Predict on a Pandas DataFrame.
import pandas as pd
loaded_model.predict(pd.DataFrame(data))
```

- Easy access and training execution of logged models for efficient and accurate reproducibility.

Preliminary Testing and Performance

We've tested our software infrastructure for flexibility across various training scenarios with :

- Data-distributed training (PyTorch DDP)
- Model-distributed training.
- Hyperparameter Optimization (Optuna)
- In conjunction with established full-suite training frameworks
    - *We've wrapped MLTF functionality around the CMS BTV POG's b-hive modular object-tagging framework.*
    - *Aside from the benefit of being able to train at MLTF, this also brings the enhancement of the MLflow suite to log system metrics, track models and parameters, and stage the trained model for inference/reproduction.*

Documentation

- Our goal is to simplify the user experience – a system that is hard to use will often remain unused
- We have begun to develop user-facing documentation for the facility - https://docs.mltf.vu/
  - We currently cover PyTorch-based training.
  - Plan to eventually explicitly document all popular frameworks.
- Still a lot of work to be done on this front

Additional Plans and Future Work

(in no particular order)

- We plan to test out interesting features like GPUDirect Storage (NVME<->GPU communication) and large-scale distributed training (10s of GPUs spread across multiple nodes).
- Development of MLflow Gateway to facilitate MLTF usage for external users.
- Add Luigi/Dask support for parallelization
  - *e.g. the user can request 2 Dask jobs, and MLflow will handle requesting them from SLURM properly*
- Deploy and demonstrate this deployment at another site
- Package and release Gateway client/server
- Develop a class/tutorial to train new users on how to use this facility

We want to thank U.S. CMS S&C HL-LHC R&D program, Vanderbilt Office of Vice Provost of Research Initiatives, and NSF Award PHY-2111554 for supporting this work