Machine Learning Platform: Deploying and Managing Models in the CERN Control System

Jean-Baptiste de Martel - Nico Madysa, Roman Gorbonosov, Verena Kain

01/07/2022 – BE Seminar
Summary

- System overview
  - Motivation
  - Objectives
  - Workflow
  - Continuous retraining
- Standalone deployment prototype
- MLP in practice
ML for accelerator controls - motivation

- Some accelerators / processes difficult to model with means available in CCC
  -> no online analytical model available
  - Space charge dominated dynamics in LINACs
  - Transmission optimization during transition crossing
  - Collimator alignment, septa with many degrees of freedom
  - Multi-turn injection with accumulated intensity diagnostics only

- Usually solved by trial & error, (semi-)manual scans
  - Time consuming
  - Depends on experience of operator

- ML techniques outperform previous techniques and open new doors to automation

- But the approach today is ad hoc and heterogeneous
Finding a compromise

Volatile world of ML

• Code needs to run once

• Bleeding edge technology

• Used to own tools and comfort, cloud services

• Maintainability is not always the main concern

Reliable world of accelerator controls

• Need to run reliably 24/7/365: need reproducibility, robustness, traceability

• Use highly reliable, battle-tested tools

• Constraints of the technical network: no internet access, restricted tooling, security precautions

• Standardize and unify to minimize maintenance
Objectives

- Provide a common approach to storage, versioning, deployment and usage of models
- Accelerate and simplify the model lifecycle by abstracting infrastructural concerns
- Fulfill the specific needs of the accelerator control system
  - Reliability
  - Traceability
  - Security
  - Standardization
- Stay out of the user’s way
  - Minimize constraints on model developer’s workflow
  - Avoid constraints on choice of tools
Model type:
Layout/architecture of the neural network – i.e., number of neurons, how they are connected, etc…

Model parameters:
“Trained weights” - values assigned to the neurons and connections after training

Model:
Combination of a model type and model parameters
Artificial Intelligence

Machine Learning

Neural Nets

Deep Learning

dozens of different ML methods

Source: https://vas3k.com/blog/machine_learning/
Development workflow

- Training environment
- VCS, CI
- Controls application
- Model registry server
- Accelerators
- Develop model
- Publish model type
- Inference
- Publish parameters
- Train model
Advantages

- Access control and traceability for model types
- Quick & easy, no need to learn new tools, complexity is hidden
Publishing model parameters

**Usage**

- Choose parameters name and version
- Use the client library to publish

```python
model.fit(training_data)
client.publish_parameters_version(
    model,
    name = "proton_beam_config",
    version = "2.0.3"
)
```

**Advantages**

- All parameters stored centrally and reliably
- Compatibility is fully managed
Inference

Usage

• Use the MLP client library to instantiate the model

• Provide model type, parameters name and version

Advantages

• Parameters retrieved and loaded transparently

• Parameter traceability

```python
model = client.create_model(
    model_type = BeamLineModel,
    model_parameters = "proton_beam",
    params_version = "2.0.3"
)
result = model.predict(input)
```
Continuous retraining - motivation

Example: stripper foil degradation

- The stripper foil is an essential component of our linacs

- It degrades over time and is replaced regularly

- Beam characteristics vary

- Machine parameters need to adapt

-> need to re-train model continuously to keep it up to date
Continuous retraining - implementation

```
model = client.create_model(
    model_type = BeamLineModel,
    model_parameters = "ion_beam_config",
    params_version = AUTO
)
result = model.predict(input)
```
Standalone deployment prototype

Calling models remotely from any language
Inference - Embedded vs standalone

**Embedded**
- Controls application
- Python only
- Model type must be installed
- Parameters retrieved then stored locally

**Standalone**
- Controls application
- Language-agnostic approach
- No local model installation needed
- Everything happens remotely

```
$ pip install mymodel
```

```
model = client.create_model(parameters = "proton_beam")
```
Inference
(Standalone)

Usage

• Use the MLP client library to instantiate the model

• Provide model type and parameters name

Advantages

• Call models from any language

• Seamless model updates

model = client.create_standalone_model(
    model_type = "beam_line.Model",
    model_parameters = "proton_beam"
)
result = model.predict(input)
Behind the scenes

Physicist

Publish model type

Model registry server

Kaniko
BeamLineModel/Protons

Kaniko
BeamLineModel/Ions

Image building cluster (TN)

MLP
Container registry

Controls application

Serving cluster (TN)

Accelerators
MLP in practice
MLP in Practice

- **Adapting a model**
  - Implementing the model interface
  - Model declaration
  - CI template

- **Using a model**
  - Publishing trained parameters
  - Loading parameters
  - Standalone prediction
MLP in practice: Implementing a model

- We define a common API for all controls models
  - shared abstraction layer
- Interface defines 4 methods:
  - *Fit* – train the model on the provided data
  - *Export parameters* – extract current values of all model parameters
  - *Load parameters* – configure the model using the provided parameters
  - *Predict* – return a prediction from the input data
- Default extensible implementations for common frameworks
MLP in practice: Implementing a model

```python
class Model(mlptf.TensorFlowModel):
    """ANN that fits a sine function."""
    def __init__(self):
        model = tf.keras.Model = tf.keras.models.Sequential([tf.keras.layers.Dense(16, activation="relu"),
                                                             tf.keras.layers.Dense(16, activation="relu"),
                                                             tf.keras.layers.Dense(1)]
                                      model.compile(loss='mean_squared_error',
                                                        optimizer=tf.keras.optimizers.Adam(0.01))
        model.build(input_shape=self.input_shape)
        super().__init__(model)
```
MLP in practice: Implementing a model

```python
def predict(self, input_data):
    inputs = np.asarray(input_data)[api.INPUTS]
    outputs = self.model.predict(inputs[:, np.newaxis])
    outputs = outputs[:, 0]
    assert np.ndim(outputs) == 1
    return {api.OUTPUTS: outputs}

def fit(self, input_data) -> api.FitHistory:
    inputs = np.asarray(input_data[api.INPUTS])
    targets = np.asarray(input_data[api.TARGETS])
    if np.ndim(targets) == 2:
        targets = np.squeeze(targets, 1)
    history = self.model.fit(inputs[:, np.newaxis], targets, epochs=500)
    return mlptf.convert_to_common_history(history)
```
MLP in practice: Implementing a model

Model declaration
- `mlp-models.toml`

CI configuration
- `gitlab-ci.yml`

```toml
[[model]]
name = "mlp_tensorflow_model_demo:Model"
standalone = false

[[model]]
name = "mlp_tensorflow_model_demo:SecondModel"
standalone = false
```
MLP in practice: Using a model

Publish trained parameters

```python
model = Model()
history = model.fit(dict(inputs=inputs, targets=targets))
client = Client(Profile.DEV)
mpv = client.publish_model_parameters_version(
    model, name="tf demo.tanh", version=AUTO)
```

Load remote parameters

```python
client = Client(Profile.DEV)
model = client.create_model(Model, "tf demo.sin", VersionFlag.AUTO)
preds = model.predict(dict(inputs=inputs))['outputs']
```

Call standalone model (prototype)

```python
client = Client(Profile.DEV)
model = client.create_standalone_model("mlp_tensorflow model demo:Model", "tf demo.sin")
preds = model.predict(dict(inputs=inputs))['outputs']
```
Conclusion

• We want to help physicists develop models faster and unburden them from infrastructural concerns while minimizing constraints

• We also want to apply software engineering best practices to ensure reliability and maintainability of the control system

• MLP provides a basis to achieve these goals and is ready for production (standalone deployment in beta)

• Could not cover everything, simplified a lot – please contact us offline!
  • MLP Wikis - ICALPCS paper – machine-learning-platform-support@cern.ch
  • jean-baptiste.de.martel@cern.ch
  • nico.madywa@cern.ch
  • roman.gorbonosov@cern.ch
model = client.create_model(
    model_type = BeamLineModel,
    model_parameters = "ion_beam",
    params_version = AUTO
)
result = model.predict(input)
Reserve slides
Use cases

- SPS eddy current effect – V. Kain
- SPS deep hysteresis compensation – N. Madysa
- LEIR Schottky computer vision – N. Madysa
- SPS MKD dump pattern analysis – F. M. Velotti
- Awake auto-matching - F. M. Velotti
Embedded use case

Physicist

VCS, CI

Controls application

Model registry server

Metadata DB

Object Storage DB

Accelerators

CI/CD

Training environment

python

Mouse

Spring boot

Oracle Database

SWIFT

Jean-Baptiste de Martel | Machine Learning Platform
Standalone use case
Publishing model types

Custom jobs

- ...
- test
- lint

Model developer

triggers

push tag v1.0.2

CI/CD

MLP jobs

build Python wheel

Publish wheel to artifact registry

scan entry points

- modelA
  - register MTV modelA v1.0.2

- modelB
  - register MTV modelB v1.0.2

Packaging & publication

Model registration
# Model parameters version number generation

<table>
<thead>
<tr>
<th>Model type version</th>
<th>Highest existing parameters version</th>
<th>-&gt;</th>
<th>Generated parameters version</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.0.0</td>
<td>None exist yet</td>
<td>-&gt;</td>
<td>1.0</td>
</tr>
<tr>
<td>1.0.0</td>
<td>1.0</td>
<td>-&gt;</td>
<td>1.1</td>
</tr>
<tr>
<td>1.6.0</td>
<td>1.1</td>
<td>-&gt;</td>
<td>1.2</td>
</tr>
<tr>
<td>2.0.0</td>
<td>1.2</td>
<td>-&gt;</td>
<td>2.0</td>
</tr>
<tr>
<td>3.3.0</td>
<td>4.0 (no 3.x)</td>
<td>-&gt;</td>
<td>ambiguity</td>
</tr>
<tr>
<td>3.3.0</td>
<td>4.0 (3.3 exists)</td>
<td>-&gt;</td>
<td>3.4</td>
</tr>
</tbody>
</table>
Compatibility

- Type v1.0
- Type v2.0
- Type v2.1

- Parameters v1.0
- Parameters v1.1
- Parameters v2.0

- Param. for other settings v1.0
# Standalone deployment CI

<table>
<thead>
<tr>
<th>Test</th>
<th>Deploy</th>
<th>Register</th>
<th>Standalone</th>
<th>Downstream</th>
</tr>
</thead>
<tbody>
<tr>
<td>test_dev</td>
<td>acc_py_relea...</td>
<td>register mode...</td>
<td>deploy stand...</td>
<td>standalone-d...#2758636 Multi-project</td>
</tr>
<tr>
<td>test_install</td>
<td>acc_py_relea...</td>
<td></td>
<td></td>
<td>standalone-d...#2758637 Multi-project</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>standalone-d...#2758636 Multi-project</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>standalone-d...#2758635 Multi-project</td>
</tr>
</tbody>
</table>