



# Neural Network Learning and Optimization (NNLO)

Irena Veljanović, Vladimir Lončar,  
Maurizio Pierini, Jean-Roch Vlimant

IML Workshop  
May 2022



# Contents

Resources that our group uses

Motivation

What is the NNLO library

NNLO workflows

Unified distributed workflow

Simplified hyperparameter optimization

Model compression

Deployment on diverse platforms

Summary

# Resources we use in our group

## Local private mPP machines with GPUs

- The machines owned by mPP group, for the purposes of ML, but the single machines is not a way to go.
- It requires time to set it up and maintain it.

## Swan

- More than a ML training platform.
- It is a data analysis facility, it has ROOT integration.
- Main focus is on physics analysis.
- For long running just on CPUs.
- Certainly the most user-friendly entry point.
- Interesting for interactive work.

## Kubeflow service ml.cern.ch

- Interactive training environment with workflows and jupyter notebook service.
- Very limited number of GPUs at the moment.
- In the future more GPUs.
- Idea interesting and promising.



# Resources we use in our group

## lxplus-gpu.cern.ch

- Most familiar solution
- User is connecting through SSH connection
- User can take the software stack from cvmf, or use his own environments stored on eos or afs
- Tesla T4 GPUs
- No access to multiple GPUs at once (problem for advanced work with big models).
- No guarantee of exclusive access to a node.
- Some issue with Singularity containers.
- Limited memory usage (job killed when threshold exceeded)
- Need of some ssh-keep-alive setup, that has to be custom installed (e.g. screen)

## Clusters outside of CERN - Flatiron

- Flatiron is a cluster at Simons Foundation.
- It has hundreds of very powerful NVIDIA V100 and A100 GPUs
- Based on SLURM, user submits a job and specify the time limit and it runs the training
- Also possible to request an interactive node for development purposes where user can login with SSH.
- Gives exclusive GPU access.
- It has very high amount of RAM and has very fast access to the disk (without errors).
- No need to care for tokens to run trainings longer than 24h (as is needed for Kerberos)



# Motivation

Noticed issues:

- Deep learning becoming popular among LHC experiments, more data, longer training.
- Training and optimization need to be faster.
- Young scientist and new machine learning users face difficulties in upscaling their ML code to be run in distributed way.
- It is also not easy to switch between different kinds of resources.

Potential solution:

- Develop a library which will easen the workflows.



# What is the NNLO library?

NNLO is a library for distributed training and optimization.

NNLO - **N**eural **N**etwork **L**earning and **O**ptimization

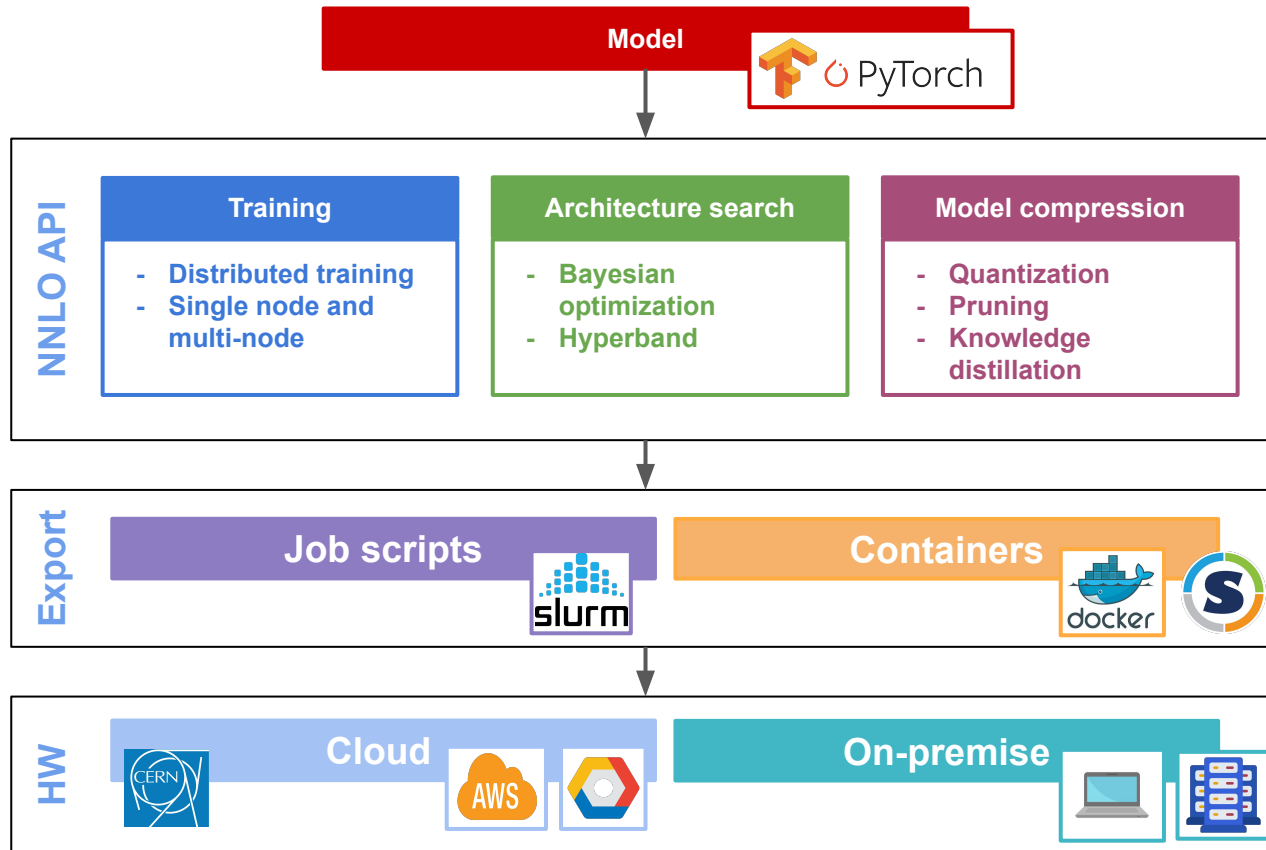
Main goals:

1. Unified distributed deep learning workflow
2. Simplified hyperparameter optimization
3. Model compression
4. Deployment on diverse platforms

This project is funded by Knowledge Transfer department (KT).

It is a project under development.

# NNLO workflows



# Unified distributed workflow




NNLO aims to have the same API for TensorFlow and PyTorch



The API takes model builder function, dataset, desired loss function, optimizer, distributed strategy.

The user has to make minimal changes, sometimes only 3 lines are needed to run distributed training using the NNLO library.

Definition of the distribution strategy:

- Single GPU 
- Single node with multiple GPUs 
- Multiple nodes with single/multiple GPUs each 

Optional dataset management

- Helps with optimal access pattern in distributed environment.

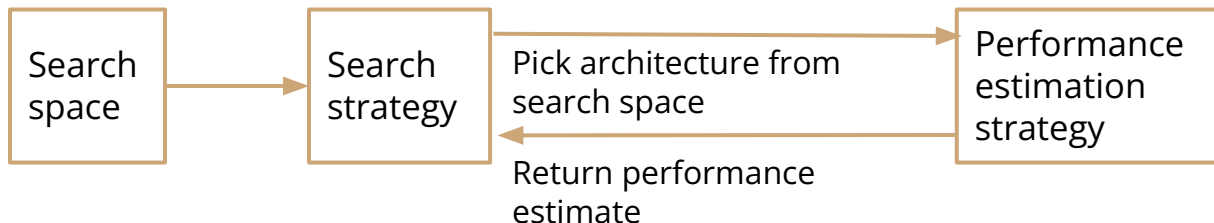


# Simplified hyperparameter optimization

The idea is to integrate well known frameworks ([Optuna](#), [Keras tuner](#), ...), in order to use them easily through NNLO library.

For example, an architecture search can be done in this way, searching for optimal numbers of layers, neurons etc.

By hyperparameter optimization, the user can get optimal values for desired hyperparameters according to the criteria which is set. The criteria can be a loss function or some of the desired metrics.





# Model compression

The goal is to support model compression through different methods:

- Quantization
  - Reducing the precision of the weights, biases, and activations such that they consume less memory.
- Pruning
  - Involves removing weights from a trained model.
- Knowledge distillation
  - The process of transferring the knowledge from a large model to a smaller one.
  - While large models have higher knowledge capacity than small models, this capacity might not be fully utilized.
  - Distillation in Keras: [https://keras.io/examples/vision/knowledge\\_distillation/](https://keras.io/examples/vision/knowledge_distillation/)

Potentially in the future also train [hls4ml](#) optimized compressed models.

# Deployment on diverse platforms

## Local resources

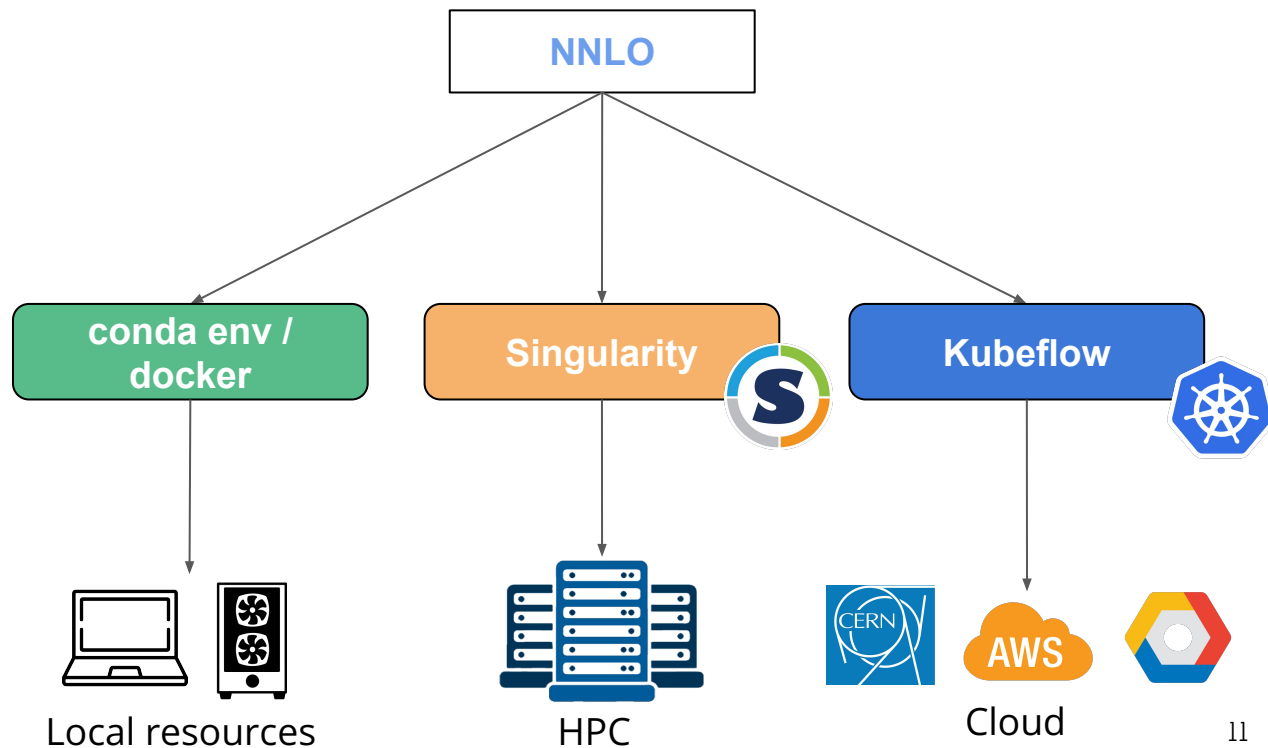
- Should imply SSH connections and running on bare metal.

## HPC

- Should imply the use of job scheduler.
- On bare metal or via singularity.

## Cloud

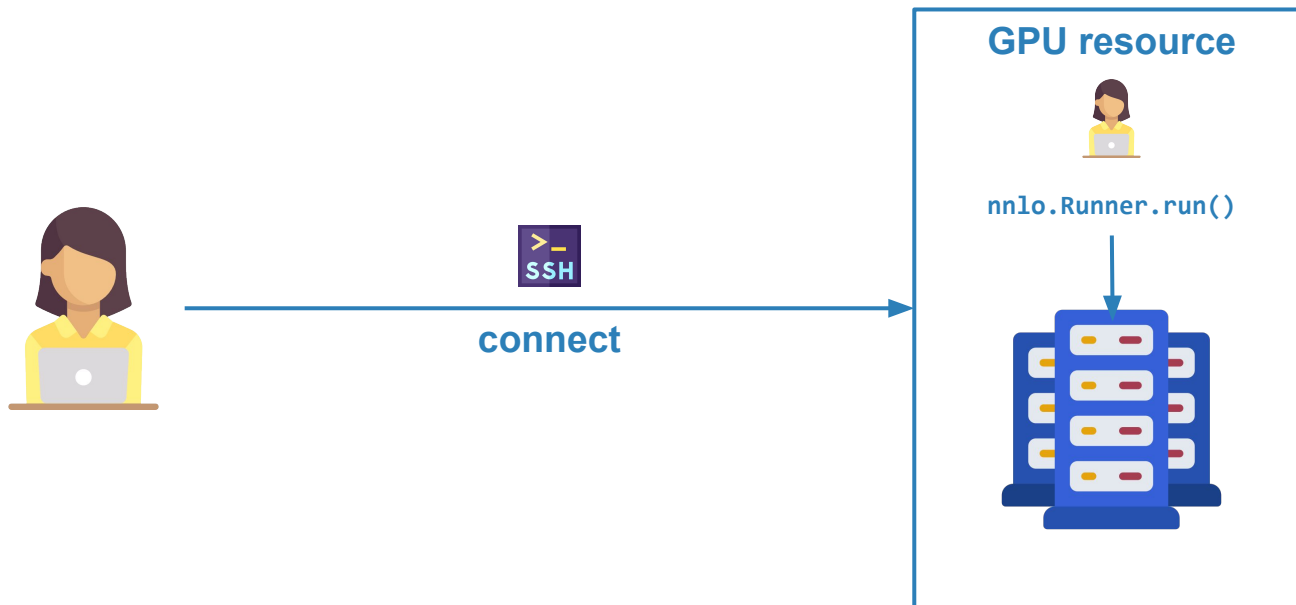
- Should imply SSH connection or have the Kubeflow service set up there.



# Local resources workflow

Anything reachable through SSH with full control

- Laptops, workstations, local machines (mPP planets), Caltech cluster,...



# Local resources workflow

By using NNLO library, user can easily scale up the training from single GPU, to multiple GPUs by adding only few lines of code.

NNLO is based on the concepts of:

- Driver
  - Defines what should be done.
- Runner
  - Does the job on one of the supported platforms.

In order to run the code in distributed manner, it is crucial to instantiate driver and runner classes.

The example for MobileNetV2 and CIFAR10: →

```
def model_builder():  
    return MobileNetV2(input_shape=(32, 32, 3),  
                       alpha=1.0, weights=None, classes=10)  
  
data_generator = Cifar10InMemoryDataSetGenerator(32)  
  
driver = TensorflowTrainingDriver(  
    'nnlo_mobile_net',  
    model_builder,  
    data_generator,  
    loss='categorical_crossentropy',  
    optimizer='adam',  
    dist_trategy=SingleNodeStrategy())  
  
runner = TensorflowTrainingRunner(driver)  
runner.run()
```



# Local resources workflow - example

Without NNLO:

```
batch_size = 32
epochs = 3

(x_train, y_train), (x_test, y_test) = tf.keras.datasets.cifar10.load_data()

y_train = y_train / 127.5 - 1.0
y_train = tf.keras.utils.to_categorical(y_train, 10)
train_dataset = tf.data.Dataset.from_tensor_slices((x_train, y_train)).batch(batch_size)

y_test = y_test / 127.5 - 1.0
y_test = tf.keras.utils.to_categorical(y_test, 10)
validation_dataset = tf.data.Dataset.from_tensor_slices((x_test, y_test)).batch(batch_size)

strategy = tf.distribute.MirroredStrategy()

with strategy.scope():
    model = MobileNetV2(input_shape=(32, 32, 3), alpha=1.0, weights=None, classes=10)
    model.compile(
        optimizer='adam',
        loss='categorical_crossentropy',
        metrics=None)
    model.fit(
        train_dataset,
        batch_size=batch_size,
        epochs=epochs,
        validation_data=validation_dataset,
        callbacks=None)
```

With NNLO:

```
def model_builder():
    return MobileNetV2(input_shape=(32, 32, 3), alpha=1.0, weights=None, classes=10)

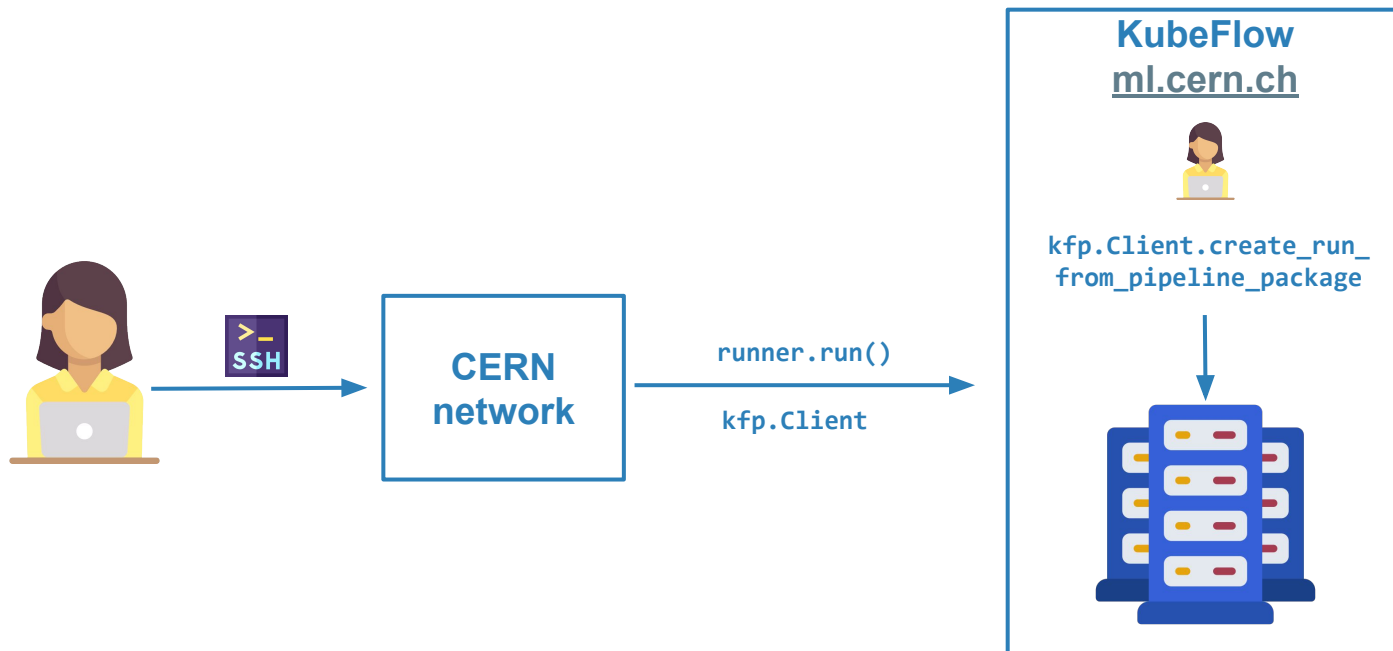
data_generator = Cifar10InMemoryDataSetGenerator(32)

driver = TensorflowTrainingDriver('nnlo_mobile_net',
    model_builder,
    Data_generator,
    loss='categorical_crossentropy',
    optimizer='adam',
    dist_trategy=SingleNodeStrategy())

runner = TensorflowTrainingRunner(driver)
runner.run()
```

# Remote resources workflow - KubeFlow

Remote execution on KubeFlow premises at CERN from local machine





# Remote resources - KubeFlow - ml.cern.ch

NNLO supports running a training as a pipeline on KubeFlow platform [ml.cern.ch](https://ml.cern.ch) developed by IT department.

It can be run by adding few lines of code:

```
from nnlo.export.kubeFlow_exporter import CERNKubeFlowExporter
from nnlo.train import TensorflowTrainingDriver
from nnlo.train import SingleNodeStrategy
```

```
data_generator = JetDataSetGenerator()

driver = TensorflowTrainingDriver('nnlo_conv_net', model_builder, data_generator,
                                  loss='categorical_crossentropy',
                                  optimizer='adam',
                                  dist_strategy=SingleNodeStrategy())

exporter = CERNKubeFlowExporter()
runner = exporter.export(driver, "nnlo-pipeline", "irena-veljanovic")
runner.run()
```

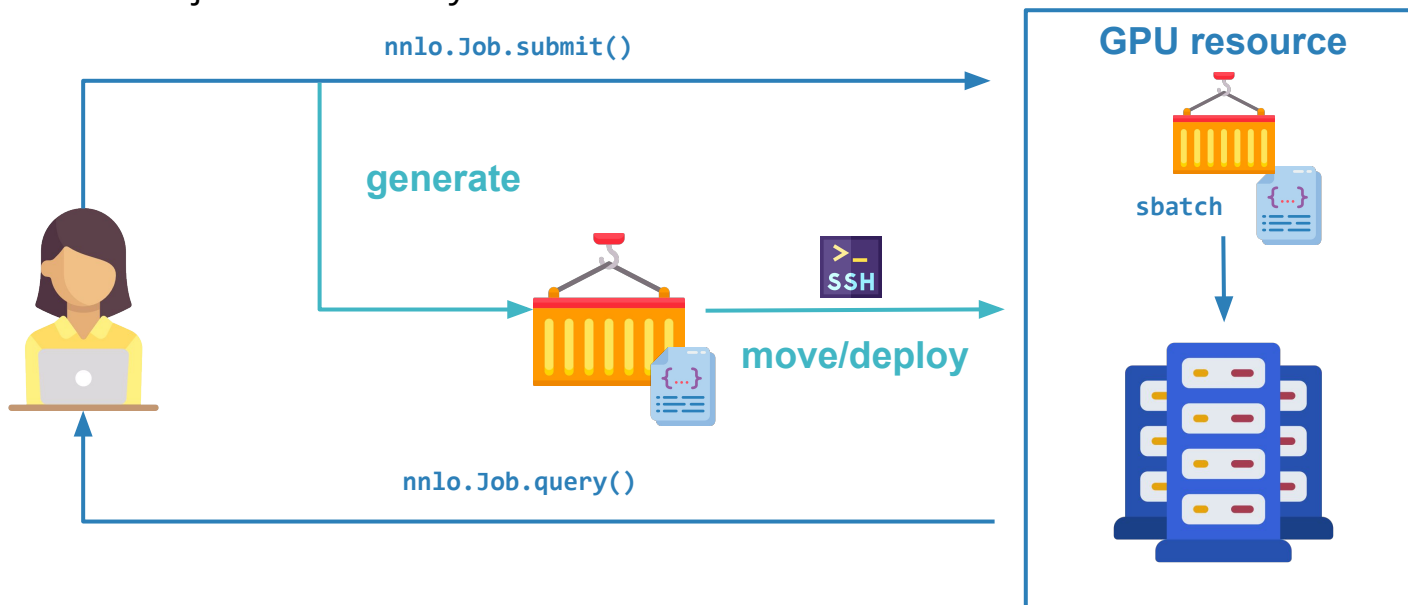
The screenshot displays the KubeFlow web interface. At the top, the KubeFlow logo and the user 'irena-veljanovic (Owner)' are visible. The main content area shows a navigation menu on the left with options like Pipelines, Experiments, Artifacts, Executions, Archive, Documentation, Github Repo, and AI Hub Samples. The right pane is titled 'Experiments > nnlo-experiment' and shows a specific experiment named 'nnlo-pipeline' with a green checkmark. Below the experiment name, there are tabs for 'Graph', 'Run output', and 'Config'. The 'Graph' tab is active, showing a single step named 'Train wrapper' with a green checkmark, indicating successful completion.



# Remote resources workflow - HPC

## Interactive cloud/HPC workflow

- Prepare the training/hyperparameter optimization locally
- Then schedule a job interactively and observe its state



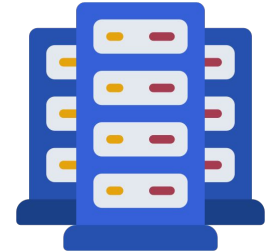


# Remote resources workflow - HPC

Depending on the security policy, sometimes it may not be possible to use TensorFlow's multi-node strategy on HPCs, therefore we would like to include inter-node communication through MPI protocol which [Horovod](#) supports.



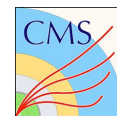
Horovod is a distributed deep learning framework for TensorFlow, Keras, PyTorch. It was originally developed by Uber to make distributed deep learning fast and easy to use. It brings model trainings time down from days and weeks to hours and minutes.



We applied for the HPC resources through [Prace](#) and we hope we will be able to start using them soon for the purpose of the development of the library.



# Summary



Main goals for the NNLO library:

Different workflows:

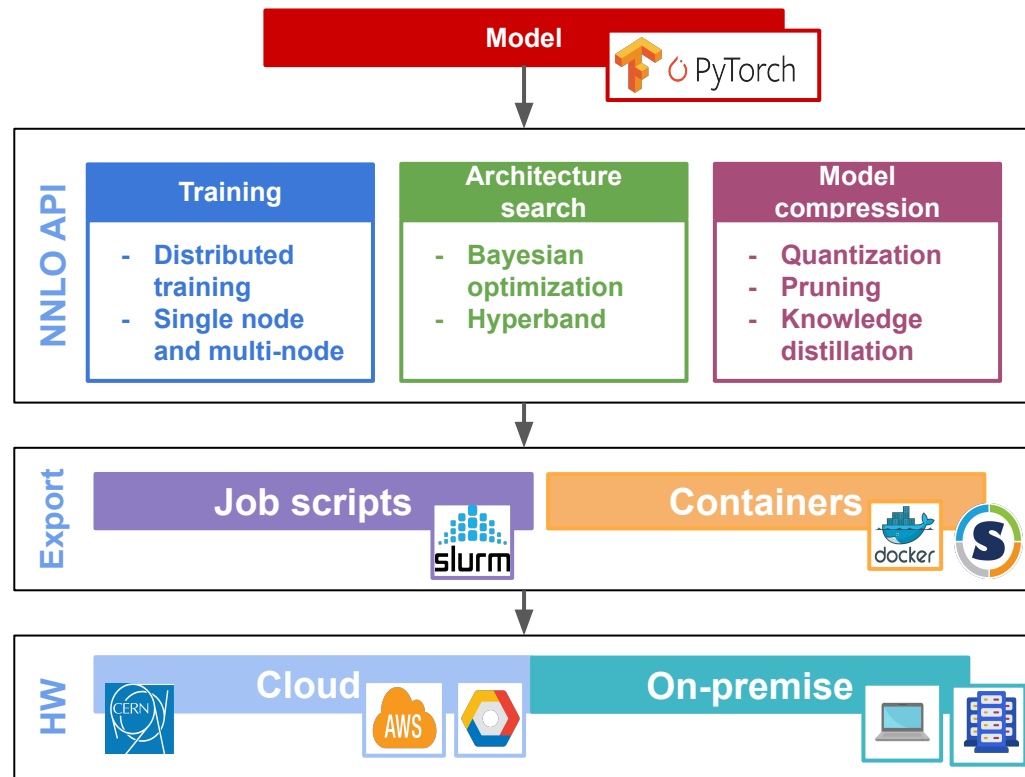
- Distributed training
- Hyperparameter optimization
- Model compression

Supports both models developed using:

- TensorFlow
- PyTorch

Running the code on different hardware resources:

- Locally
- Cloud
- HPC



# Thank you!

irena.veljanovic@cern.ch



Knowledge Transfer